# **TÓM TẮT**

Tên đề tài: THIẾT KẾ PHẦN MỀM CƠ SỞ DỮ LIỆU CAN

Sinh viên thực hiện: HOÀNG NGỌC THANH NGÂN

Số thẻ sinh viên: 106120034 Lớp: 12DT1

Sinh viên thực hiện: NGUYỄN ĐÌNH TUYẾN

Số thẻ sinh viên: 106120097 Lớp: 12DT2

Đề tài “Thiết kế phần mềm cơ sở dữ liệu CAN” thuộc hệ thống “ Phần mềm mô phỏng CAN BUS” được xây dựng dựa theo mong muốn trình chiếu được tất cả các dữ liệu từ các ECUs để có thể biết được dữ liệu hoạt động như thế nào. Các Message và Signal được load lên và thể hiện được tất cả các thông số đi kèm, ngoài ra phần mềm còn cho phép thêm, chỉnh sửa và xóa các Message, Signal theo ý muốn để có được một cơ sở dữ liệu hoàn thiện. Dữ liệu được thể hiện trên giao diện người dùng dễ sử dụng trên 2 nền tảng là WinDow và Linux, giao diện được viết bằng ngôn ngữ C++ nhờ sự hỗ trợ của Qt Framework.

|  |  |
| --- | --- |
| ĐẠI HỌC ĐÀ NẴNG  **TRƯỜNG ĐẠI HỌC BÁCH KHOA**  KHOA ………………………………………… | **CỘNG HÒA XÃ HÔI CHỦ NGHĨA VIỆT NAM**  Độc lập - Tự do - Hạnh phúc |

**NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP**

Họ tên sinh viên: …..…………….………….…….. Số thẻ sinh viên: ………………...

Lớp:…………… Khoa:....................................... Ngành: ……………….......................

Họ tên sinh viên: …..…………….………….…….. Số thẻ sinh viên: ………………...

Lớp:…………… Khoa:....................................... Ngành: ……………….......................

1. *Tên đề tài đồ án:*

………………………………………………..…………………………………………

…………………………………………………………………………………………..

1. *Đề tài thuộc diện:*  *Có ký kết thỏa thuận sở hữu trí tuệ đối với kết quả thực hiện*
2. *Các số liệu và dữ liệu ban đầu:*

……………………………………..……………………………………………..……......…………………………………………………………………………………………

1. *Nội dung các phần thuyết minh và tính toán:*

…...………………………………………………………………………………………

…...………………………………………………………………………………………

…...………………………………………………………………………………………

1. *Các bản vẽ, đồ thị ( ghi rõ các loại và kích thước bản vẽ ):*

…...………………………………………………………………………………………

…...………………………………………………………………………………………

…...………………………………………………………………………………………

1. *Họ tên người hướng dẫn:* …………………………………..……………………
2. *Ngày giao nhiệm vụ đồ án:*  *……../……./201…..*
3. *Ngày hoàn thành đồ án: ……../……./201…..*

|  |  |
| --- | --- |
|  | *Đà Nẵng, ngày tháng năm 201* |
| **Trưởng Bộ môn** …………………….. | **Người hướng dẫn** |

# **LỜI NÓI ĐẦU VÀ CẢM ƠN**

Trong thời gian hơn 4 tháng thực hiện đồ án tốt nghiệp dưới hình thức Capstone Project tại FPT Software, chúng tôi đã được tiếp xúc, làm quen với môi trường làm việc tại công ty. Học hỏi được nhiều kiến thức bổ ích, cũng như tích lũy thêm kinh nghiệm để chuẩn bị những hành trang cho bản thân nhằm đáp ứng nhu cầu tuyển dụng của các doanh nghiệp sau khi ra trường.

Chúng tôi xin chân thành cảm ơn FPT Software và lãnh đạo Khoa Điện tử - Viễn thông đã tạo điều kiện để chúng tôi được tham gia và thực hiện đồ án . Chúng tôi xin chân thành cảm ơn giảng viên hướng dẫn - cô Trần Thị Hương và hơn hết là anh Nguyễn Văn Thành – FPT Software đã quan tâm trong suốt quá trình thực hiện, hướng dẫn nhiệt tình để chúng tôi có thể hoàn thành được đồ án này.

# **LỜI CAM ĐOAN**

Kính gửi: Hội đồng bảo vệ đồ án tốt nghiệp khoa Điện tử - Viễn thông, Trường Đại học Bách Khoa-Đại học Đà Nẵng

Chúng tôi gồm: Hoàng Ngọc Thanh Ngân Lớp 12DT1

Nguyễn Đình Tuyến Lớp 12DT2

Sinh viên Khoa Điện tử -Viễn thông, Trường Đại học Bách Khoa –Đại học Đà Nẵng.

Chúng tôi xin cam đoan nội dung của Đồ án này không phải là bản sao chép của bất cứ Đồ án hoặc Công trình đã có từ trước. Nếu vi phạm chúng tôi xin chịu mọi hình thức kỷ luật của Khoa.

Đà Nẵng, tháng 05 năm 2017

Sinh viên thực hiện Sinh viên thực hiện

Nguyễn Đình Tuyến Hoàng Ngọc Thanh Ngân

# **MỤC LỤC**

[**TÓM TẮT**](#_Toc482643425)

[**NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP**](#_Toc482643426)

[**LỜI CẢM ƠN** i](#_Toc482643427)

[**LỜI CAM ĐOAN** ii](#_Toc482643428)

[**MỤC LỤC** iii](#_Toc482643429)

[**DANH MỤC CÁC BẢNG, HÌNH VẼ** vi](#_Toc482643430)

[**DANH MỤC CÁC TỪ VIẾT TẮT** vii](#_Toc482643431)

[**MỞ ĐẦU** 1](#_Toc482643432)

[**Chương 1: TỔNG QUAN HỆ THỐNG MÔ PHỎNG CAN BUS** 2](#_Toc482643433)

[**1.1.** **Giới thiệu chương** 2](#_Toc482643434)

[**1.2.** **Tổng quan về CAN BUS** 2](#_Toc482643435)

[**1.3.** **Hệ thống mô phỏng CAN BUS** 4](#_Toc482643436)

[**1.4.** **Các thành phần của hệ thống.**](#_Toc482643437) 6

[1.4.1. CAN Software 6](#_Toc482643438)

[1.4.1.1. Phần mềm mô phỏng CAN. 6](#_Toc482643439)

[1.4.1.2. Phần mềm cơ sở dữ liệu CAN. 7](#_Toc482643440)

[1.4.2. CAN Device 7](#_Toc482643441)

[1.4.3. PC Driver và Protocol 7](#_Toc482643442)

[**1.5.** **Kết luận chương** . 8](#_Toc482643443)

[**Chương 2: CƠ SỞ LÝ THUYẾT VÀ MÔI TRƯỜNG THỰC HIỆN** 9](#_Toc482643444)

[**2.1.** **Giới thiệu chương.** 9](#_Toc482643445)

[**2.2.** **Cơ sở dữ liệu hệ thống CAN.** 9](#_Toc482643446)

[2.2.1. Cơ sở dữ liệu. 9](#_Toc482643447)

[2.2.2. Cơ sở dữ liệu CAN. 9](#_Toc482643448)

[**2.3. Các phương pháp lập trình.** 10](#_Toc482643449)

[2.3.1. Lập trình tuyến tính. 10](#_Toc482643450)

[2.3.2. Lập trình hướng thủ tục 10](#_Toc482643451)

[2.3.3. Lập trình hướng đối tượng. 12](#_Toc482643452)

[**2.4. Môi trường phát triển.** 14](#_Toc482643453)

[2.4.1. Hệ điều hành Linux. 14](#_Toc482643454)

[2.4.1.1. Khái niệm. 14](#_Toc482643455)

[2.4.1.2. Ưu điểm của hệ điều hành. 15](#_Toc482643456)

[2.4.2. QT software framework. 16](#_Toc482643457)

[2.4.2.1. Giới thiệu chung về QT framework. 16](#_Toc482643458)

[2.4.2.2. Mô hình Model/View trong QT framework. 17](#_Toc482643459)

[2.4.2.3. Cơ chế Signal và Slot trong QT framework. 19](#_Toc482643460)

[2.4.2.4. Các lớp Qt được sử dụng để xây dựng Can Database. 20](#_Toc482643461)

[**2.5.** **Kết luận chương.** 25](#_Toc482643468)

[**Chương 3:**  **KẾT QUẢ THỰC HIỆN VÀ ĐÁNH GIÁ** 26](#_Toc482643469)

[**3.1.** **Giới thiệu chương.** 26](#_Toc482643470)

[**3.2. Thiết kế và thi công.** 26](#_Toc482643471)

[3.2.1. Phần giao diện. 26](#_Toc482643472)

[3.2.1.1. Giao diện chính. 26](#_Toc482643473)

[3.2.1.2. Giao diện của cửa sổ chỉnh sửa 28](#_Toc482643474)

[3.2.1.3. Đánh giá 31](#_Toc482643475)

[3.2.2. Phần xử lí back-end 31](#_Toc482643476)

[3.2.3. Phần edit cơ sở dữ liệu 31](#_Toc482643477)

[3.2.3.1.Kết quả 34](#_Toc482643478)

[3.2.3.2. Đánh giá 42](#_Toc482643479)

[**3.3.Hướng phát triển** 42](#_Toc482643480)

[**3.4. Kết luận chương.** 42](#_Toc482643481)

[**KẾT LUẬN** 43](#_Toc482643482)

[**TÀI LIỆU THAM KHẢO** 45](#_Toc482643484)

[**PHỤ LỤC**  1](#_Toc482643485)

# **DANH MỤC CÁC BẢNG, HÌNH VẼ**

[*HÌNH 1. 1. Sơ đồ khối mô hình kết nối CAN BUS* 2](#_Toc482863541)

[*HÌNH 1. 2. BUS CAN với 3 node* 3](#_Toc482863542)

[*HÌNH 1.3. Sơ đồ khối mô tả giao tiếp hệ thống CAN BUS* 5](#_Toc482863543)

[*HÌNH 1.4. Sơ đồ khối hệ thống CAN BUS* 6](#_Toc482863544)

[*HÌNH 1.5. CAN Software bao gồm: Database Edit Software và Simulator software.* 6](#_Toc482863545)

[*HÌNH 2. 1. Kiến trúc hệ điều hành Linux* 15](#_Toc482863546)

[*HÌNH 2.2. Mô hình tổng quan Qt framework* 17](#_Toc482863547)

[*HÌNH 2.3. Kiến trúc mô hình MVC* 18](#_Toc482863548)

[*HÌNH 2.4. Cấu trúc model/view trong QT* 19](#_Toc482863549)

[*HÌNH 2.5. Cấu trúc của cửa sổ QMainWindow* 21](#_Toc482863550)

[*HÌNH 2.6. Cửa sổ giao diện chính sử dụng QWindow và QMenu* 21](#_Toc482863551)

[*HÌNH 2.7. Giao diện menu khi click chuột phải* 22](#_Toc482863552)

[*HÌNH 2.8. Các cửa sổ con được tạo ra trong cửa sổ chính khi gọi lớp QMdiArea* 23](#_Toc482863553)

[*HÌNH 2.9. Hộp thoại QDialog* 23](#_Toc482863554)

[HÌNH 2.10*. Table được tạo khi gọi lớp QTableWidget, QTableWidgetItem* 24](#_Toc482863555)

[*HÌNH 2.11. Ví dụ về TabWidget* 25](#_Toc482863556)

[*HÌNH 2.12. Các loại Layout trong Qt* 26](#_Toc482863557)

[*HÌNH 3.1. Giao diện chính CAN database* 28](#_Toc482864314)

[*HÌNH 3.2. Giao diện CAN database sau khi load dữ liệu lên* 29](#_Toc482864315)

[*HÌNH 3. 3. Giao diện cửa sổ message editor: tab definition* 30](#_Toc482864316)

[*HÌNH 3.4. Giao diện cửa sổ message editor: tab signal* 31](#_Toc482864317)

[*HÌNH 3.5. Sequence diagram LoadDB* 32](#_Toc482864318)

[*HÌNH 3.6. Sequence diagram SaveDB* 34](#_Toc482864319)

[*HÌNH 3.7. Thông tin Signal trước khi edit* 36](#_Toc482864320)

[*HÌNH 3.8. Thông tin Signal sau khi edit* 37](#_Toc482864321)

[*HÌNH 3.9. Delegate dữ liệu* 38](#_Toc482864322)

[*HÌNH 3.10. List Signal trước khi Add* 39](#_Toc482864323)

[*HÌNH 3.11. Bảng Thêm Signal* 40](#_Toc482864324)

[*HÌNH 3.12. List Signal sau khi được thêm* 40](#_Toc482864325)

[*HÌNH 3.13. List Signal được update tại TreeView và TableView* 41](#_Toc482864326)

[*HÌNH 3.14. Remove message* 41](#_Toc482864327)

[*HÌNH 3.15. List Signal trước khi remove Signal* 42](#_Toc482864328)

[*HÌNH 3.16. List Signal sau khi remove Signal* 42](#_Toc482864329)

[*HÌNH 3.17. List Signal sau khi đã xóa tại TreeView* 43](#_Toc482864330)

**DANH MỤC CÁC TỪ VIẾT TẮT**

CHỮ VIẾT TẮT:

CAN : Controller Area Network

ECU : Electronic Control Unit

PC : Personal Computer

CSMA : Carrier-Sense Multiple Access

CD+AMP : Collision Detection Arbitration on Message Priority

IG : Interactive Generator

EV : Environment Variable

GPL : General Public Licence

MVC : Model-View-Controller

# **MỞ ĐẦU**

Cùng với sự phát triển không ngừng của khoa học công nghệ, công nghiệp ôtô cũng đã và đang được hoàn thiện nhằm mục đích đem đến cho người dùng những trải nghiệm ngày càng thoải mái và tiện nghi hơn. Sự ra đời của các công nghệ, thiết bị hiện đại như cảnh báo chệch làn đường, cảnh báo tiền va chạm, hệ thống hỗ trợ phanh và điều khiển động cơ bánh lái…đã mang lại cho người dùng những chuyến hành trình ngày càng thú vị. Tuy nhiên, để áp dụng những công nghệ mới này lên trên một sản phẩm xe ô tô thực tế để đưa ra ngoài thị trường thì từ khâu kiểm thử, vận hành cho đến phát triển các ứng dụng về sau thì đây cũng là một khó khăn đối với các nhà nghiên cứu, cũng như sản xuất ô tô công nghiệp. Việc sử dụng cả hệ thống thực tế chỉ để kiểm thử với một ECU (Electronic Control Unit) mới nhằm hướng tới sự tương thích giữa các ECU với nhau có thể gây ra sự phức tạp cho toàn bộ hệ thống CAN BUS (Controller Area Network).

Chính vì vậy, việc xây dựng nên một phần mềm có thể mô phỏng hoạt động như một hệ thống hoàn chỉnh nhằm thay thế các ECU thực tế là thật sự cần thiết. Nó sẽ giúp cho việc phát triển thêm ứng dụng đơn giản hơn, với độ tin cậy và hiệu quả cao.

Hiểu được tầm quan trọng của nó, nhóm chúng tôi đã đăng ký tham gia Đồ án tốt nghiệp dưới hình thức Capstone Project cùng FPT Software với đề tài “THIẾT KẾ VÀ THI CÔNG HỆ THỐNG MÔ PHỎNG CAN BUS”. Hệ thống bao gồm: Phần mềm mô phỏng CAN, phần mềm cơ sở dữ liệu CAN, CAN Device và PC Driver.

Phạm vi của chúng tôi thực hiện trong đề tài là “THIẾT KẾ PHẦN MÊM CƠ SỞ DỮ LIỆU CAN”.

Nội dung báo cáo gồm các chương:

Chương 1: Tổng quan hệ thống mô phỏng CAN Bus

Chương 2: Cơ sở lý thuyết và môi trường thực hiện

Chương 3: Kết quả thực hiện và đánh giá

Trong quá trình thực hiện đồ án, chúng tôi đã cố gắng rất nhiều song không khỏi mắc những sai sót, kính mong quý thầy cô thông cảm và đóng góp ý kiến để đồ án được hoàn thiện hơn.

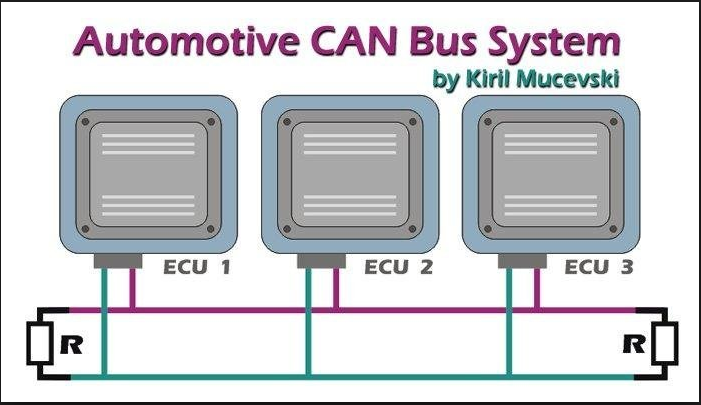
# **Chương 1: TỔNG QUAN HỆ THỐNG MÔ PHỎNG CAN BUS**

# **Giới thiệu chương**

Chương này sẽ giới thiệu về đường truyền CAN và hệ thống mô phỏng CAN BUS, bao gồm: Phần mềm mô phỏng CAN, phần mềm cơ sở dữ liệu CAN, CAN Device và PC Driver.

# **Tổng quan về CAN BUS**

Mạng khu vực điều khiển CAN – Controller Area Network (hay còn gọi là CAN BUS) là mạng lưới đường truyền thông tin các bộ điều khiển độc lập dưới dạng đường truyền BUS. Trong đó các thành phần nối với mạng có quyền ngang nhau trong việc truyền và nhận thông tin (multi master), bất kỳ các thành phần đó đều có thể truyền cũng như nhận thông tin mà chúng cần từ các thành phần khác. Vì vậy, đây là một giao thức truyền thông nối tiếp có hỗ trợ hiệu quả phân phối thời gian thực kiểm soát với mức độ bảo mật rất cao.



*HÌNH 1. 1. Sơ đồ khối mô hình kết nối CAN BUS*

Tiêu chuẩn Bus CAN được phát triển bởi Bosch (một nhà sản xuất thiết bị điện tại Đức) và Intel vào đầu những năm 1980. Sau đó, CAN đã được chuẩn hóa theo tiêu chuẩn ISO-11898 và ISO-11519, thiết lập chính nó như là giao thức chuẩn để giao

tiếp kết nối mạng trong ngành công nghiệp ô tô. Trong những ngày đầu của ngành công nghiệp ô tô, bộ điều khiển độc lập cục bộ đã được sử dụng để quản lý cơ cấu truyền động khác nhau và các hệ thống cơ điện phụ. Bởi kết nối mạng các thiết bị điện tử trong xe theo chuẩn CAN, và được được điều khiển từ một điểm trung tâm, bộ điều khiển động cơ (ECU), do đó làm tăng chức năng, thêm mô đun, và làm cho quá trình chẩn đoán hiệu quả hơn.

Thời gian đầu giao tiếp CAN được phát triển chủ yếu hỗ trợ cho ngành công nghiệp xe, vì vậy nó đã được sử dụng trong xe ô tô chở khách, tàu thuyền, xe tải, và trên nhiều loại xe khác. Ngày nay giao thức CAN  đang được sử dụng trong nhiều lĩnh vực khác nhau trong các ứng dụng với tên gọi mạng điều khiển nhúng (Networked Embedded Control), bao gồm cả tự động hóa công nghiệp, ứng dụng y tế, xây dựng tự động hóa, máy dệt, máy móc sản xuất. CAN cung cấp một giao thức truyền thông hiệu quả giữa các bộ cảm biến, cơ cấu truyền động, điều khiển, và các nút khác trong các ứng dụng thời gian thực, và được biết đến với sự đơn giản, độ tin cậy và hiệu suất cao.



*HÌNH 1. 2. BUS CAN với 3 node*

Hình trên cho thấy một bus CAN với ba node. Các giao thức CAN được dựa trên giao thức CSMA / CD+AMP (Carrier-Sense Multiple Access:  Đa truy cập có tránh xung đột / Collision Detection Arbitration on Message Priority: Giám sát phát hiện xung đột dựa trên thông điệp ưu tiên), nó tương tự như các giao thức được sử

dụng trong mạng Ethernet LAN. Khi mạng Ethernet phát hiện một vụ xung đột, các nút đang gửi dữ liệu chỉ đơn giản là dừng truyền và đợi một khoảng thời gian ngẫu nhiên trước khi cố gắng để gửi một lần nữa. Tuy nhiên, đối với giao thức CAN có một chúc khác biệt đó là các vấn đề xung đột được giải quyết  bằng cách sử dụng nguyên tắc nơi chỉ có các nút ưu tiên cao nhất được trao quyền để gửi dữ liệu của nó.

**Một số các tính năng giao thức CAN:**

* CAN bus là đa chủ. Khi bus dữ liệu rỗi, bất kỳ thiết bị gắn vào bus đều có thể bắt đầu gửi một thông điệp.
* CAN giao thức bus là linh hoạt. Các thiết bị kết nối với bus không có địa chỉ, có nghĩa là tin nhắn không được truyền đi từ một nút này đến một nút khác dựa trên các địa chỉ. Thay vào đó, tất cả các nút trong hệ thống nhận mọi tin nhắn truyền đi trên bus, và nó là đến mỗi nút để quyết định xem đã nhận tin nhắn sẽ được lưu giữ hay loại bỏ. Một tin nhắn duy nhất có thể được dành cho một nút cụ thể hoặc cho nhiều nút, tùy thuộc vào cách hệ thống được thiết kế.
* Một lợi thế của việc không có địa chỉ là khi một thiết bị được thêm vào hay lấy đi khỏi bus, không cần phải thay đổi cấu hình dữ liệu  (tức là, các bus "có thể gắn nóng (hot pluggable)").
* Tốc độ giao tiếp bus CAN là không cố định. Bất kỳ tốc độ truyền thông có thể được thiết lập cho các thiết bị gắn vào một bus.
* Tất cả các thiết bị trên bus có thể phát hiện một lỗi. Các thiết bị đã phát hiện một lỗi ngay lập tức thông báo cho tất cả các thiết bị khác.
* Nhiều thiết bị có thể được kết nối bus tại cùng một thời điểm, và không có được giới hạn hợp lý về số lượng các thiết bị có thể được kết nối. Trong thực tế, số thiết bị có thể được gắn vào một bus bị hạn chế bởi thời gian trễ của bus và tải điện.

# **Hệ thống mô phỏng CAN BUS**

Hệ thống mô phỏng CAN Bus giao tiếp các ECU trong ô tô sử dụng đường truyền CAN bus, bao gồm CAN Device thực hiện hoạt động truyền nhận thông điệp tương tự như một ECU thực tế,PC Driver có nhiệm vụ giao tiếp giữa Hardware và Software và CAN Simulator là phần mềm mô phỏng, hiển thị định dạng của thông

điệp, tín hiệu, đồng thời có thể cấu hình chọn lựa cổng CAN, tốc độ baud…với cửa sổ thống kê hiển thị tương tự như trên phần mềm CANoe Vector đã và đang được các nhà phát triển, sản xuất ô tô trên thế giới sử dụng hiện nay.



*HÌNH 1.3. Sơ đồ khối mô tả giao tiếp hệ thống CAN BUS*

Để có thể kiểm tra được ECU có hoạt động đúng yêu cầu thiết kế hay không, ta sẽ xây dựng mô hình kiểm tra như trên , trong đó ý tưởng chính dự trên việc giả lập một hệ thống ô tô hoàn chỉnh trên máy tính và kết nối nó với ECU thực tế để kiểm tra bằng cách truyền các thông điệp yêu cầu ECU thực hiện và nhận lại các thông điệp phản hồi từ ECU, sau đó xử lý nội dung của phản hồi và hiển thị lên màn hình thống kê một cách trực quan với độ chính xác cao.

ECU cần kiểm tra sẽ được kết nối với CAN Device thông qua cổng CAN. CAN Device kết nối với Software, nơi chứa hệ thống giả lập gồm các ECU (Electronic Control Unit) mô phỏng hệ thống thực tế thông qua cổng USB nhờ PC driver .PC driver đóng vai trò cầu nối giao tiếp giữa hardward và software thông qua cơ chế truyền (ghi) và nhận (đọc) tin từ các file descriptor.



*HÌNH 1.4.* ***Sơ đồ khối hệ thống CAN*** *BUS*

Sau khi kết nối giữa ECU và Software , mô hình hệ thống sẽ gồm nhiều ECU được kết nối với nhau thông qua CAN BUS như là một hệ thống CAN thực tế hoàn chỉnh.

# **Các thành phần của hệ thống.**

Hệ thống mô phỏng CAN Bus bao gồm: Phần mềm mô phỏng CAN, phần mềm cơ sở dữ liệu CAN, CAN Device và PC Driver

# ***CAN Software***

.

*HÌNH 1.5. CAN Software bao gồm: Database Edit Software và Simulator software.*

# ***Phần mềm mô phỏng CAN***

Là một phần mềm độc lập, có nhiệm vụ xử lí việc truyền nhận dữ liệu cũng như phân tích và hiển thị nội dung của các thông điệp, mà định dạng của các thông điệp đó được lấy từ D atabase edit software. Bao gồm các chức năng:

* Cấu hình hệ thống, cổng CAN nào đang hoạt động , tốc độ Baud…
* Load được database vào test model.
* Add/remove được sơ đồ khối kết nối đồng thời cập nhật cấu hình và lưu lại.
* Add/remove được các message vào trong IG. (interactive generator).
* Trong IG có thể setting cycle của từng message.
* Trace window hiển thị các thuộc tính của message: ID, length, direction, data, signal value.

# ***Phần mềm cơ sở dữ liệu CAN***

Database Edit Software là phần mềm độc lập quản lí cơ sở dữ liệu của hệ thống, lưu giữ các thông điệp (message) và các tín hiệu (signal), hỗ trợ các loại format Little Endian và Big Endian , MSB và LSB. Các chức năng chính của Database Software là: Tạo mới database và mở 1 database có sẵn.

Hổ trợ các chức năng:

- Thêm/sửa/xóa các thông điệp và tín hiệu trong các thông điệp.

- Thêm/sửa/xóa các nốt mạng (node network) và các biến môi trường EV (environment variable).

Vai trò chính của Database edit software là tạo dựng một cơ sở dữ liệu gồm các node network và các thông điệp gửi đi cũng như nhận về từ chính node đó để cung cấp dữ liệu cho việc mô phỏng ở simulator software.

# ***CAN Device***

**CAN Device:** Chức năng chính của CAN Device là tạo một node CAN có thể tùy chỉnh được tốc độ của node để có thể kết nối được với một CAN BUS bất kỳ để lấy các CAN\_Frame trong CAN BUS và gởi lên PC.

Để thực hiện các chức năng đó chúng ta sử dụng giao thức USB và CAN. Board DK-TM4C123G hỗ trợ

* Nhiệm vụ chính:
* Giao tiếp USB nhận và gởi message để giao tiếp giữa board và PC
* Giao Tiếp CAN
* Nhận CAN Frame từ CAN Bus và gởi dữ liệu thu được từ CAN Frame lên PC.
* Kiểm tra thiết bị đã được kết nối với CAN Bus hay chưa, trả kết quả lại cho PC.
* Cấu hình được tốc độ của thiết bị để có thể kết nối được với Bus CAN bất kỳ.

# ***PC Driver và Protocol***

**PC driver**:

PC Driver có nhiệm vụ giao tiếp giữa hardware và software .Khi một device được kết nối với PC, driver nhận nhiệm vụ tạo ra vùng nhớ đệm thông qua các file descriptor. Khi ta muốn truyền một thông điệp từ PC xuống Hardware thì Driver sẽ nhận biết được thông điệp, sau đó ghi vào file descriptor tương ứng. Hardware sẽ đọc dữ liệu từ file descriptor và thực hiện nội dung thông điệp. Thông điệp phản hồi sẽ được truyền theo chiều ngược lại đến Software.

* Nhận biết được thiết bị connect, disconnect thông qua cổng USB, đưa ra thông tin, trạng thái của thiết bị.
* CAN software có thể đóng/ mở thiết bị thông qua device node.
* CAN software có thể cấu hình baurd rate thông qua file descriptor.
* CAN software có thể gửi/nhận message thông qua file descriptor

**Protocol:**

Giao thức truyền dữ liệu giữa cổng software và thiết bị TM4C123G là tập các ngữ cảnh để thực hiện việc truyền nhận dữ liệu khi phần mềm được khởi động, giao thức trong phạm vi đề tài này chỉ đề cập đến các ngữ cảnh đơn giản như : kiểm tra hoạt động kit TM4C123G khi kết nối với CAN bus, truyền nhận dữ liệu chứa cấu hình thiết bị, thực hiện truyện nhận dữ liệu CAN frame khi phần mềm thực hiện chức năng mô phỏng để giám sát dữ liệu từ CAN bus. Vì thiết bị đóng vai trò như một node CAN trên mạng CAN bus, phần mềm trên PC đóng vai trò hiển thị

message để kiểm tra, nên cần thiết phải thiết kế một chuẩn giao thức gồm nhiều trường hợp khác nhau để hình thành quy luật truyền nhận dữ liệu từ PC xuống thiết bị và ngược lại, theo các chức năng mà người giám sát mạng CAN cần đến.

# **Kết luận chương**

Qua chương này, hiểu rõ được tổng quan về hệ thống mô phỏng CAN Bus, cách thức hoạt động cũng như các thành phần chính của hệ thống. Chương tiếp theo sẽ trình bày cụ thể hơn về phần thực hiện Phần mềm cơ sở dữ liệu CAN của chúng tôi.

# **Chương 2: CƠ SỞ LÝ THUYẾT VÀ MÔI TRƯỜNG THỰC HIỆN**

* 1. **Giới thiệu chương**

Chương này sẽ trình bày cơ sở lí thuyết, những kiến thức cơ bản về cơ sở dữ liệu, kỹ thuật lập trình cũng như môi trường phát triển của phần mền cơ sở dữ liệu CAN, bao gồm lựa chọn hệ điều hành và môi trường phát triển tích hợp IDE.

* 1. **Cơ sở dữ liệu hệ thống CAN**
     1. ***Cơ sở dữ liệu***

**Định nghĩa dữ liệu:**

Dữ liệu là các thông tin của đối tượng, chúng ta có thể truy xuất vào dữ liệu để lấy thông tin.

Dữ liệu có thể được mô tả dưới nhiều dạng khác nhau: các kí tự, hình ảnh, số liệu, âm thanh…ứng với mỗi cách mô tả là một ngữ nghĩa khác nhau.

**Định nghĩa cơ sở dữ liệu:**

Cơ sở dữ liệu (database) là một tập hợp dữ liệu được tổ chức có cấu trúc liên quan với nhau và được lưu trữ trong máy tính.

Cơ sở dữ liệu được thiết kế, xây dựng, cho phép người dùng lưu trữ dữ liệu, truy xuất thông tin hoặc cập nhật dữ liệu.

* + 1. ***Cơ sở dữ liệu CAN***

Cơ sở dữ liệu CAN là cơ sở dữ liệu khá đơn giản, nơi mà thông tin được lưu trữ dưới dạng văn bản thuần.

Dữ liệu trong cơ sở dữ liệu CAN sẽ chứa thông tin mô tả cấu trúc hệ thống CAN (Controller Network Area). Các thông tin đó mô tả một hệ thống CAN hoàn chỉnh có phân cấp rõ ràng:

* Trong mạng sẽ chứa các danh sách các node mạng và danh sách các Signal/Message trong các node đó. Hệ thống hoạt động theo đơn vị dữ liệu là các message(có thể chứa 1 hoặc nhiều signal)
* Danh sách các ECUs(Electronic Controll Unit) chứa các biến môi trường(EVs) và các node mạng
* Trong mỗi node mạng sẽ phân biệt 2 loại message: Tx message và Rx message tương ứng với thông điệp phát đi và nhân được từ mỗi ECU.
* Một danh sách các message sẽ lưu thông tin các message của toàn bộ hệt thống, có thể đã được dùng hoặc chưa được dùng. Mỗi message sẽ chứa danh sách các signal trong message đó. Mỗi message có thể có 1 hoặc nhiều signal, tuy nhiên dữ liệu mà các signal chứa không được vượt quá dữ liệu mà message chứa. Một message không chứa signal sẽ không có ý nghĩa trong hệ thống (thường là các message vứa được tạo mới).
* Một danh sách các signal sẽ lưu thông tin tất cả các signal, kể cả những signal được tạo ra nhưng chưa sử dụng trong bất kì message nào.
  1. **Các phương pháp lập trình**
     1. ***Lập trình tuyến tính***

Đặc trưng cơ bản của lập trình tuyến tính là tư duy theo lối tuần tự. Chương trình sẽ được thực hiện từ đầu đến cuối, lệnh này kế tiếp lệnh kia cho đến hết chương trình.

**Đăc trưng:**

* Đơn giản: chương trình được thực hiện theo lối tuần tự, không phức tạp
* Đơn luồng: chỉ có một việc công việc duy nhất, và các công việc được thực hiện trong luồng đó

**Tính chất:**

* Ưu điểm: do tính chất đơn giản, dễ hiểu, lập trình tuyến tính được ứng dụng trong các chương tình đơn giản.
* Nhược điểm: không thể ứng dụng lập trình tuyến tính trong các ứng dụng lớn. Ngày nay, lập trình tuyến tính chỉ còn tồn tại trong các module nhỏ nhất của các phương pháp lạp trình khác.
  + 1. ***Lập trình hướng thủ tục (hay hướng cấu trúc)***

Phương pháp lập trình thủ tục (*procedural programming*) chia một chương trình lớn thành các khối chức năng hay hàm (thủ tục) đủ nhỏ để dễ lập trình và kiểm tra. Mỗi hàm có một điểm bắt đầu và một điểm kết thúc và có dữ liệu và logic riêng. Trong một hệ thống chương trình, các biến có các phạm vi nhìn thấy nhất định. Trong chương trình, các hàm làm việc độc lập với nhau. Dữ liệu được chuyển đổi qua lại thông qua các tham số gọi hàm. Việc chia chương trình thành các hàm cho phép nhiều

người có thể tham gia vào việc xây dựng chương trình. Mỗi người xây dựng một hay một số các hàm độc lập với nhau. Phương pháp này dẫn đến một khái niệm mới – sự trừu tượng hóa. Sự trừu tượng hóa có thể xem như khả năng quan sát một sự việc mà không cần xem xét đến các chi tiết bên trong của nó. Trong một chương trình thủ tục, chúng ta chỉ cần biết một hàm nào đó có thể làm được những công việc cụ thể gì là đủ. Còn làm thế nào để thực hiện công việc đó là không quan trọng, chừng nào hàm còn tin cậy được thì còn có thể dùng nó mà không cần phải biết nó thực hiện đúng đắn chức năng của mình như thế nào. Điều này gọi là sự trừu tượng hóa theo chức năng (*functional abstraction*) (hay còn gọi là sự chuyên môn hóa) và là nền tảng của lập trình thủ tục.

**Đặc trưng:**

Đặc trưng cơ bản nhất của lập trình hướng cấu trúc thể hiện ở mối quan hệ:

**Chương trình = Cấu trúc dữ liệu + Giải thuật**

Cấu trúc dữ liệu: là cách tổ chức dữ liệu cho việc xử lí của một hay nhiều chương trình nào đó.

Giải thuật: là một quy trình để thực hiện một công việc cụ thể

Mối quan hệ giữa giải thuật và cấu trúc dữ liệu:

* Một cấu trúc dữ liệu chỉ phù hợp với một số lượng hạn chế các giải thuật
* Nếu thay đổi cấu trúc dữ liệu thì phải thay đổi giải thuật cho phù hợp
* Một giải thuật thường phải đi kèm với một cấu trúc dữ liệu nhất định

**Tính chất:**

* Mỗi chương trình con có thể được thực hiện nhiều lần trong chương trình chính
* Các chương trình con có thể được gọi trong chương trình chính để thực hiện theo một thứ tự bất kì, tùy thuộc vào giải thuật mà không phụ thuộc vào thứ tự khai báo các chương trình con.

**Ưu điểm:**

* Chương trình rõ ràng, dễ hiểu, dễ theo dõi
* Tư duy giải thuất rõ ràng

**Nhược điểm:**

Lập trình cấu trúc không hổ trợ mạnh mẽ việc sử dụng lại mã nguồn. Giải thuật luôn phụ thuộc chặt chẽ vào cấu trúc dữ liệu, do đó khi thay đổi cấu trúc dữ liệu phải thay đổi giải thuật, nghĩa là viết lại chương trình.

Không phù hợp với phần mềm lớn: tư duy cấu trúc chỉ phù hợp với các bài toán nhỏ, nằm trong phạm vi một module của chương trình.Với các dự án phần mềm lớn, lập trình hướng cấu trúc tỏ ra không hiệu qủa.

* + 1. ***Lập trình hướng đối tượng***

Trong lập trình hướng đối tượng, lập trình viên coi các thực thể trong chương trình là các đối tượng sau đó trưu tượng hóa đối tượng thành lớp đối tượng.

Dữ liệu được tổ chức thành các thuộc tính của lớp. Ta chỉ có thể truy cập dữ liệu thông qua đối tượng và các phương thức của đối tượng

**Tính chất:**

* Tính kế thừa: Chúng ta có thể kế thừa các lớp để xây dựng các lớp dẫn xuất, lớp được kế thừa được gọi là lớp cơ sở. Lớp dẫn xuất sẽ có tất cả các thuộc tính và phương thức của lớp cơ sở mà nó kế thừa. Như vậy ta không cần phải xây dựng lại một đối tượng mới hoàn toàn.

Tính kế thừa của lập trình hướng đối tượng cho phép một lớp có thể kế thừa từ các lớp cơ sở. Khi đó lớp dẫn xuất sẽ có thể sử dụng những thuộc tính và phương thức của lớp cơ sở như là của mình.Ngoài ra, lớp dẫn xuất còn có thể bổ sung thêm các thuộc tính phương thức mới giúp giảm được khối lượng công việc trong lập trình hướng đối tượng.

C++ còn hổ trợ đa kế thừa, nghĩa là một lớp có thể kế thừa từ nhiều lớp khác nhau giúp kế thừa các thuộc tính, phương thức cần thiết từ nhiều lớp.

* Tính đa hình: Đa hình là tính chất đi kèm với kế thừa, do việc kế thừa, một lớp có thể sử dụng lại các phương thức của lớp khác. Tuy nhiên, nếu cần thiết, lớp dẫn xuất có thể định nghĩa lại phương thức của lớp cơ sở. Đó là sự nạp chồng phương thức trong kế thừa. Nhờ sự nạp chồng phương thức này ta chỉ cần gọi tên phương thức từ đối tượng mà không cần biết đối tượng thuộc lớp nào. Chương trình sẽ tự kiểm tra đối tượng thuộc lớp nào và thực hiện phương thức tương ứng.
* Tính trừu tượng: Trừu tượng hóa dữ liệu (Data abstraction) liên quan tới việc chỉ cung cấp thông tin cần thiết tới bên ngoài và ẩn chi tiết cơ sở của chúng, ví dụ: để biểu diễn thông tin cần thiết trong chương trình mà không hiển thị chi tiết về chúng. C++ cung cấp đủ các phương thức public tới bên ngoài để thao tác với tính năng của đối tượng và để thao tác dữ liệu đối tượng mà không cần thực sự biết về cách lớp đó đã được triển khai nội tại.
* Tính đóng gói: Các dữ liệu được đóng gói vào trong đối tượng. Mỗi đối tượng có một phạm vi truy nhập riêng.

Không thể truy nhập đến dữ liệu một cách tự do như lập trình cấu trúc.

Muốn truy nhập vào các thuộc tính được bảo vệ, phải thông qua các đối tượng, nghĩa là phải thông qua các phương thức mà đối tượng đó cung cấp mới có thể truy nhập được dữ liệu đã được bảo vệ.

* Ưu điểm:Không còn nguy cơ dữ liệu bị thay đổi trong chương trình vì dữ liệu đã được đóng gói trong các đối tượng. Việc truy xuất dữ liệu phải thông qua các phương thức được cho phép bởi đối tượng.

Khi thay đổi cấu trúc dữ liệu của đối tượng, không cần thay đổi mã nguồn của các đối tượng khác, mà chỉ cần thay đổi một số thành phần của lớp dẫn xuất. Tránh được việc phải thay đổi quá nhiều mã khi thay đổi cấu trúc dữ liệu.

Có thể sử dụng lại mã nguồn, tiết kiệm đáng kể tài nguyên, thời gian và chi phí. Điều này có được nhờ tính kế thừa.

Phù hợp với các phần mềm lớn, phức tạp.

Hạn chế của lập trình hướng đối tượng C++:

Tính đa kế thừa trong C++ có thể dẫn đến xung đột phương thức nếu 1 lớp kế thwuaf từ 2 lớp khác mà 2 lớp đó lại có các phương thức với các API giống nhau. Lúc đó, đối tượng của lớp dẫn xuất sẽ không biết được đang gọi phương thức của lớp cơ sở nào. Vì vậy lập trình viên cần cẩn trọng không lạm dụng đa kế thừa.

**Kết luận:** Là ngôn ngữ nữa hướng đối tượng, nữa hướng cấu trúc do kế thừa từ ngôn ngữ C thuần cấu trúc vì vậy C++ có cả đặc tính hướng cấu trúc và hướng đối

tượng, tùy thuộc vào trường hợp cụ thể mà ta sẽ sử dụng phương pháp phù hợp.

Như vậy, C++ vừa có thể sử dụng cho lập trình phần cứng, tác động sâu vào từng vùng nhớ, vừa có tính chất hướng đối tượng giúp tiết kiệm thời gian và tài nguyên.

* 1. **Môi trường phát triển**
     1. ***Hệ điều hành Linux*** 
        1. ***Khái niệm***

Linux là hệ điều hành mô phỏng Unix, được xây dựng trên phần nhân (kernel) và các gói phần mềm mã nguồn mở. Linux được công bố dưới bản quyền của GPL (General Public Licence).

****

*HÌNH 2. 1. Kiến trúc hệ điều hành Linux*

Giống như Unix, Linux gồm 3 thành phần chính: kernel, shell và cấu trúc tệp. Kernel là chương trình nhân, chạy các chương trình và quản lý các thiết bị phần cứng như đĩa và máy in.

Shell (môi trường) cung cấp giao diện cho người sử dụng, còn được mô tả như một bộ biên dịch. Shell nhận các câu lệnh từ người sử dụng và gửi các câu lệnh đó cho nhân thực hiện. Nhiều shell được phát triển. Linux cung cấp một số shell như: desktops, windows manager, và môi trường dòng lệnh. Hiện nay chủ yếu tồn tại 3

shell: Bourne, Korn và C shell. Bourne được phát triển tại phòng thí nghiệm Bell, C shell được phát triển cho phiên bản BSD của UNIX, Korn shell là phiên bản cải tiến của Bourne shell. Những phiên bản hiện nay của Unix, bao gồm cả Linux, tích hợp cả 3 shell trên.

Cấu trúc tệp quy định cách lưu trữ các tệp trên đĩa. Tệp được nhóm trong các thư mục. Mỗi thư mục có thể chứa tệp và các thư mục con khác. Một số thư mục là các thư mục chuẩn do hệ thống sử dụng. Người dùng có thể tạo các tệp/thư mục của riêng mình cũng như dịch chuyển các tệp giữa các thư mục đó. Hơn nữa, với Linux người dùng có thể thiết lập quyền truy nhập tệp/thư mục, cho phép hay hạn chế một người dùng hoặc một nhóm truy nhập tệp. Các thư mục trong Linux được tổ chức theo cấu trúc cây, bắt đầu bằng một thư mục gốc (root). Các thư mục khác được phân nhánh từ thư mục này. Kernel, shell và cấu trúc tệp cấu thành nên cấu trúc hệ điều hành. Với những thành phần trên người dùng có thể chạy chương trình, quản lý tệp, và tương tác với hệ thống.

* + - 1. ***Ưu điểm của của hệ điều hành***
* **Bản quyền:**

Lợi thế của Linux chính là nền tảng mã nguồn mở và hoàn toàn miễn phí.

* **Linh hoạt**

Tính linh hoạt của Linux được thể hiện ở chỗ nó tương thích được với rất nhiều môi trường. Hiện tại, ngoài Linux dành cho server,máy tính để bàn...nhân Linux (Linux kernel) còn được nhúng vào các thiết bị điều khiển như máy tính palm, robot.....Phạm vi ứng dụng của Linux được xem là rất rộng rãi.

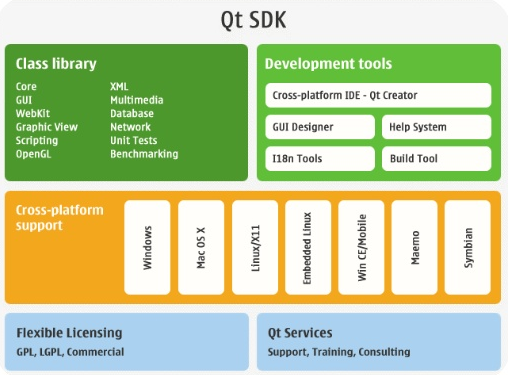
* **Độ an toàn cao**

Trước hết, trong Linux có một cơ cấu phân quyền hết sức rõ ràng. Chỉ có "root"(người dùng tối cao) mới có quyền cài đặt và thay đổi hệ thống. Ngoài ra Linux cũng có cơ chế để một người dùng bình thường có thể chuyển tạm thời chuyển sang quyền "root" để thực hiện một số thao tác. Điều này giúp cho hệ thống có thể chạy ổn định và tránh phải những sai sót dẫn đến đổ vỡ hệ thống. Trong những phiên bản Windows gần đây, cơ chế phân quyền này cũng đã bước đầu được áp dụng, nhưng so với Linux thì vẫn kém chặt chẽ hơn.

* **Chạy thống nhất trên các hệ thống phần cứng**

Dù cho có rất nhiều phiên bản Linux được các nhà phân phối khác nhau ban hành nhưng nhìn chung đều chạy khá ổn định trên mọi thiết bị phần cứng, từ Intel 486 đến những máy Pentium mới nhất, từ những máy có dung lượng RAM chỉ 4MB đến những máy có cấu hình cực mạnh (tất nhiên là tốc độ sẽ khác nhau nhưng về nguyên tắc vẫn có thể chạy được). Nguyên nhân là Linux được rất nhiều lập trình viên ở nhiều môi trường khác nhau cùng phát triển (không như Windows chỉ do Microsoft phát triển) và bạn sẽ bắt gặp nhiều người có "cùng cảnh ngộ" như mình và dễ dàng tìm được các driver tương ứng với thiết bị của mình. Tính chất này hoàn toàn trái ngược với Windows. Mỗi khi có một phiên bản Windows mới ra đời thì bao giờ kèm theo đó cũng là một cơn khát về phần cứng vì HĐH mới thường không hỗ trợ các thiết bị quá cũ.

* + 1. ***QT software framework*** 
       1. ***Giới thiệu chung về QT Framework***
* Định nghĩa framework: Là một thư viên các lớp được xây dựng hoàn chỉnh, là nền tảng để phát triển các phần mềm ứng dụng. Thay vì tự tay viết tất cả mã cho các lớp , hàm cho toàn bộ dự án phần mềm của mình, các lập trình viên dùng framework để tiết kiệm thời gian và công sức nhưng vẫn đạt hiệu quả mong muốn bằng cách kế thừa các lớp có sẵn từ các thư viện trong framework



*HÌNH 2.2. Mô hình tổng quan Qt framework*

* QT Framework là một nền tảng xây dựng các ứng dụng chạy trên nhiều hệ điều hành khác nhau.
* Mục tiêu của các nhà phát triển nên Qt chính là tạo ra một nền tảng có khả năng thiết kế những phần mềm có thể chạy trên nhiều nền tảng phần mềm lẫn phần cứng khác nhau mà không phải thay đổi nhiều về code.
* Qt hổ trợ các nền tảng sau: Windows, Linux, OS X, iOS, Android, WinRT…
* Qt hổ trợ ngôn ngữ C/C++/Java/Python/…
* Qt hổ trợ lập trình giao diện chuẩn theo mô hình model/view và hổ trợ các lớp lập trình giao diện động trong Qt Quick, giúp tạo giao diện chuyên nghiệp và sinh động hơn.
* Với QT chúng ta không cần thiết phải viết giao diện từ những dòng code phức tạp, mà hoàn toàn có thể vẽ giao diện trong tệp có phần mở rộng .ui như vậy việc thiết kế giao diện dễ dàng hơn nhiều. UI của Qt hổ trợ sẵn các công cụ như QPushButton, QLineEdit, QComboBox, QTreeView, QStackedWidget… giúp việc thiết kế dễ dàng hơn nhiều so với việc phải tự xây dựng tất cả các lớp trên.
* Với những tính năng nổi bật trên, nhóm em quyết định chọn Qt framework làm nền tảng để xây dựng phần mềm CAN database.
  + - 1. ***Mô hình Model/View trong QT Framework***
* Kiến trúc MVC và kiến trúc model/view trong QT:



*HÌNH 2.3. Kiến trúc mô hình MVC*

* Mô hình MVC (Model-View-Controller) là một kiến trúc phần mềm, nó giúp cho các nhà phát triển phần mềm tách các ứng dụng của họ ra thành 3 phần có nhiệm vụ riêng biệt và độc lập với các thành phần khác, trong đó:
* Model: chứa các phương thức xử lí và truy xuất dữ liệu.
* View: Đảm bảo việc hiển thị thông tin và tương tác với người dùng.
* Controller: là trung gian giữa Model và View, nhận các yêu cầu từ tương tác người dùng từ view và tác động làm thay đổi dữ liệu trong model.
* Như vậy, việc xây dựng phần mền dùng MVC sẽ dễ dàng, trực quan hơn so với việc xây dựng theo kiểu truyền thống. Ngoài ra, việc chia thành 3 phần riêng biệt giúp cho việc phát triển, nâng cấp, bảo trì và khắc phụ sự cố tiết kiệm thời gian, công sức và chi phí. Việc chia nhỏ như vậy cuãng thuận tiện để phân chia nhiệm vụ và chuyên biệt hóa các phần trong các dự án lớn.
* QT framework sử dụng mô hình model/view thay cho mô hình MVC, trong đó lớp Controller sẽ được tích hợp trong View thông qua Delegate.

*HÌNH 2.4. Cấu trúc model/view trong QT*

* Lí do lựa chọn mô hình model/view: mô hình này chia phần mền thành các module riêng biệt, nên sẽ dễ dàng hơn cho việc bảo trì, nâng cấp, phát triển sau này.
* Hoạt động của model/view:
* Ban đầu model sẽ sử dụng phương thức setModelData () của mình để lấy dữ liệu từ tập dữ liệu gốc (dataset), và View sẽ tự cập nhật dữ liệu được lấy từ model thông qua phương thức setModel ().
* Mỗi tương tác chỉnh sửa của người dùng trên giao diện sẽ gọi một đối tượng của 1 lớp Delegate tương ứng, thông qua delegate sẽ làm thay đổi dữ liệu tương ứng

trong model, khi đó model sẽ cập nhật hiển thị mới trên View cũng như phát đi một signal để làm thay đổi dữ liệu ở dữ liệu gốc, signal này được kết nối với một slot tương ứng ở lớp quản lí tập dữ liệu gốc, slot này sẽ làm thay đổi dữ liệu trong dataset.

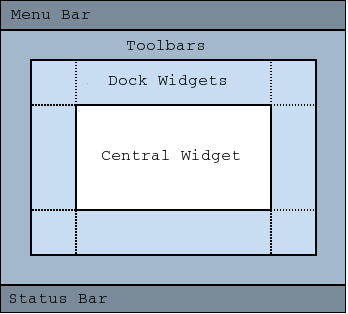
* + - 1. ***Cơ chế Signal và Slot trong QT Framework***
* Trong các phần mềm, nơi người dùng có thể tương tác được với giao diện, phải có một cơ chế nào đó để lúc người dùng tương tác( như kích chuột, di chuyển con trỏ chuột,…) với giao diện thì phần mềm sẽ biết được người dùng đang thực hiện thao tác đó. Qt sử dụng cơ chế signal/slot để đảm nhận nhiệm vụ này.
* Signal: khi một sự kiện nào đó xảy ra, một signal sẽ được phát đi, thực ra nó chỉ là một phương thức của một lớp nhưng không có phần thân hàm {}. Các lớp Widget có sẵn trong Qt có rất nhiều signal được định nghĩa sẵn, và chúng ta cũng có thể viết các signal riêng cho các lớp của tự định nghĩa. Signal không có kiểu trả về, kiểu trả về của signal luôn luôn là void.
* Slot:  là một phương thức bình thường của một lớp, các phương thức này sẽ được gọi khi có một signal nào đó được phát đi. Cũng giống như signal, các lớp Widget trong Qt cũng có sẵn rất nhiều slot và chúng ta cũng có thể viết slot cho lớp của riêng chúng ta.
* Connect: Signal và slot được kết nối qua từng đối tượng thông qua phương thức connect.

Connect (Đối tượng 1, SIGNAL (signal thuộc đối tượng 1), Đối tượng 2, SLOT (slot thuộc đối tượng 2));

Sau khi kết nối sẽ có một đối tượng phát ra signal và một đối tượng nhận signal đó và thực hiện một phương thức tương ứng, lúc này phương thức đó sẽ là một slot .Kết nối signal và slot là một cách tiện lợi để người dùng gửi đi các signal làm thay đổi dữ liệu tương ứng trên dữ liệu gốc cũng như trên model thay cho delegate.

* Để có thể kết nối signal và slot thì hoặc là phương thức slot của đối tượng nhận signal phải có thành phần tham số giống với signal hoặc là slot phải không có tham số.
  + - 1. ***Các lớp Qt được sử dụng để xây dựng CAN database***

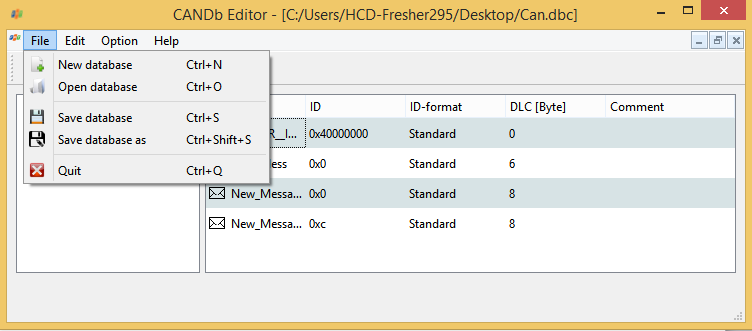
**QMainWindow:**



*HÌNH 2.5. Cấu trúc của cửa sổ QMainWindow*

Là cửa sổ chính cung cấp nền tảng cho việc xây dựng một ứng dụng tương tác người dùng. Qt tạo ra QMainWindow và các lớp liên quan để quản lí cửa sổ chính. QMainWindow có layout của chính nó để người dùng có thể thêm QToolBars, QDockWidgets, QMenuBar và QStatusBar.

**QWindow:**



*HÌNH 2.6. Cửa sổ giao diện chính sử dụng QWindow và QMenu*

* Cửa sổ chính cung cấp một khuôn khổ để xây dựng giao diện cho người dùng.
* QWindow có bố cục riêng mà bạn có thể thêm QToolBars, QMennuBar hay QStatusBar bằng các phương thức setMenuBar (), setStatusbar ()…

**QMenu:**



*HÌNH 2.7. Giao diện menu khi click chuột phải*

Lớp QMenu cung cấp một danh sách các chọn lựa của hành động để dùng trong thanh menu hoặc context menu (menu xuất hiện khi kích chuột vào một vùng nào đó trong QWindow).

**QAction:**

Trong 1 menu chứa một hoặc nhiều QAction, chính là các hành động để người dùng chọn lựa. Khi kích chuột vào một action thì một signal sẽ được giải phóng , tương ứng với nó, 1 slot sẽ được thực thi.

**QMdiArea:**

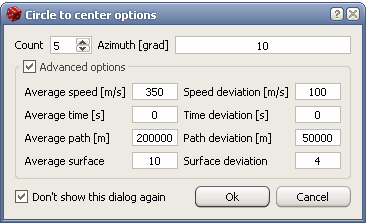
Lớp QMdiArea là tiện ích cung cấp một khu vực trong đó một cửa sổ con được tạo ra trong cưa sổ QMainwindow.

Về cơ bản, giống như cửa sổ quản lý, nó kế thừa hầu hết các chức năng của lớp QMainwindow.



*HÌNH 2.8. Các cửa sổ con được tạo ra trong cửa sổ chính khi gọi lớp QMdiArea*

**QDialog:**

****

*HÌNH 2.9. Hộp thoại QDialog*

Một hộp thoại tương tự như QWindow nhưng không chứa QMenuBar, QStatusbar, QToolBars.

Hộp thoại QDialog thường là nhưng hộp thoại nhỏ, với các tác vụ đơn giản và tương tác với người dùng trong thời gian ngắn. Ví dụ như các hộp thoại chỉnh sửa dữ liệu, hộp thoại tạo mới đối tượng hay đơn thuần là các hộp thoại hiển thị thông báo, cảnh báo người dùng.

**QFile:**

Lớp QFile cung cấp một giao diện hổ trợ việc đọc và ghi vào tệp.

Là một thiết bị nhập xuất I/O cho việc đọc và ghi tệp dạng văn bản, dạng nhị phân hoặc dạng tài nguyên khác như hình ảnh, âm thanh. QFile có thẻ kết hợp với QTextStream hoặc QDataStream để đọc và ghi dữ liệu từ tệp một cách thuận tiện hơn.

**QStackedWidget:**

Lớp QStackedWidget cung cấp một ngăn xếp các widgets nơi mà chỉ có duy nhất một widget hiển thị trong một thời điểm tùy vào chỉ số index trong hàm setCurrentIndex (index) hay setCurrentWidget (QWidget).

Sử dụng các hàm addWidget(QWidget \*widget ), insertWidget(int index, QWidget \*widget) và removeWidget(QWidget \*widget ) để thêm, chèn và xóa các widget trong QStackedWidget.

**QTableWidget và QTableWidgetItem.**

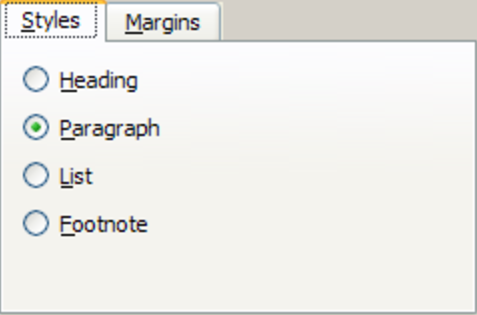
* + - Lớp QTablewidget cung cấp một khung nhìn dựa trên bảng mục tiêu với một mô hình mặc định.
    - Những lớp này cung cấp công cụ xây dựng bảng với số lượng hàng và cột theo yêu cầu.
    - Các tiêu đề trong bảng được tạo ra bằng cách cung cấp một danh sách các chuỗi cho các hàm setHorizontalHeaderLabels () and setVerticalHeaderLabels ().



HÌNH 2.10*. Table được tạo khi gọi lớp QTableWidget, QTableWidgetItem*

**QTabWidget:**

QTabWidget cung cấp một ngăn xếp các tab , trong đó mỗi tab là một widget .



*HÌNH 2.11. Ví dụ về TabWidget*

**QMessageBox:**

QMessageBox cung cấp một hộp thoại nhằm thông báo cho người dùng hoặc yêu cầu người dùng cung cấp một xác nhận, một chọn lựa cho hành động vứa thực hiện. Ví dụ : Khi xóa 1 thông điệp, do sau khi xóa dữ liệu sẽ mất và không thể khôi phục được nên 1 hộp thoại yêu cầu xác nhận có đúng là người dùng muốn xóa hay không sẽ được hiển thị kèm theo cảnh báo. Hoặc khi thực thi một hành động không hợp lê, sẽ có một cảnh báo tới người dùng thông qua QMessageBox.

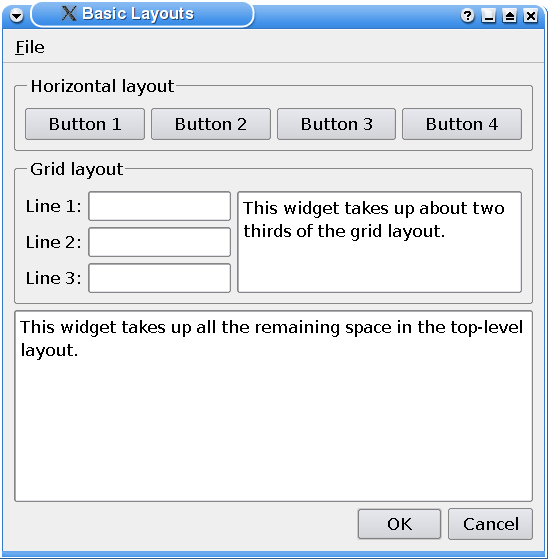
**Layout:**

Qt hổ trợ lớp QLayout cùng các lớp dẫn xuất của nó giúp lập trình viên có thể dễ dang thay đổi định dạng vị trí của các khối chức năng trong các widget, cũng như vị trí của các widget trong các trang.

Layout sẽ khiến giao diện đẹp mắt hơn và thân thiện người dùng hơn. Trong Qt , chúng ta có thể bố trí trang dùng mã hoặc layout bằng giao diện, tức là tự vẽ giao diện layout sau đó chỉnh sửa lại bằng mã C++.

Các lớp dẫn xuất thường dùng của Qt là: **QVBoxlayout, QHBoxLayout, QGridLayout**

* + - Các lớp này xắp xếp các widgets theo một mạng lưới hàng ngang hay hàng dọc, và đặt các widgets mà nó quản lý vào vị trí chính xác.
    - Lớp QVBoxLayout xắp xếp các widgets theo hàng dọc.
    - Lớp QHboxLayout xắp xếp các widgets theo hàng ngang.
    - Lớp QGridLayout tùy chỉnh xắp xếp lại các lớp QVBoxLayout và QHboxLayout bằng cách sử dụng phương thức addlayout ().
    - Khoảng cách giữa các hàng và các cột được tùy chỉnh khi khai báo hàm setColumMinumumWidth ().



*HÌNH 2.12. Các loại Layout trong Qt*

* 1. **Kết luận chương**

Trên đây là toàn bộ cơ sở lí thuyết cơ bản phục vụ cho quá trình thiết kế phần mềm cơ sở dữ liệu CAN. Những kiến thức tham khảo từ các tài liệu tham khảo uy tín là nền tảng vững chắc giúp chúng tôi tư duy đúng hướng và giải quyết đúng vấn đề. Phần mềm được xây dựng dựa trên việc áp dụng chặt chẽ các nguyên tắc cốt lõi trong kỹ thuật lập trình.

# **Chương 3: KẾT QUẢ THỰC HIỆN VÀ ĐÁNH GIÁ**

* 1. **Giới thiệu chương**

Chương này trình bày tiến trình thực hiện các chức năng của phần mềm, nêu rõ từng bước thực hiện các nhiệm vụ, cùng với kết quả đạt được ở mỗi phần .Cuối cùng là phần tự đánh giá kết quả đã đạt được trong thời gian thực hiện đồ án của nhóm.

* 1. **Thiết kế và thi công**
     1. ***Phần giao diện*** 
        1. ***Giao diện chính***

Giao diện CAN database gồm có giao diện chính và các cửa sổ chỉnh sửa các thông điệp hoặc signal.

* **Giao diện của sổ chính:**
* **Tiến trình thực hiện:**

Giao diện màn hình chính là một cửa sổ chính sử dụng QMainWindow, trong đó có Menu các hành động như: tạo mới database, mở database có sẵn, lưu database đã mở, lưu lại database dưới dạng tên khác, thoát khỏi chương trình, chức năng hổ trợ Help,…những hành động này sẽ được mô tả bởi các QAction trong Qt và toàn bộ những QAction sẽ được gói trong QMenu tại thanh MenuBar.

* **Kết quả:**

Giao diện chính khi chạy chương trình

* Các chức năng tại thanh Menu bao gồm:

Tạo database

Mở database

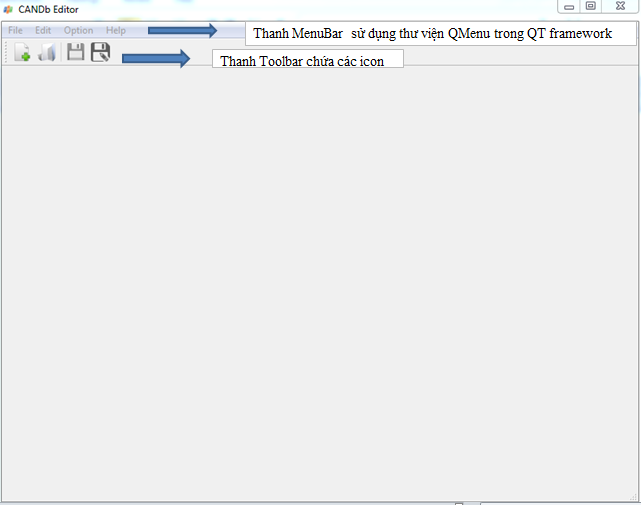
Lưu database dưới dạng file .txt

Thoát khỏi chương chình đang chạy

* F:\FPT\12-5\CANDbEditor\icons\new.pngCác chức năng được đưa ra tại thanh Toolbar dưới dạng icon bao gồm:

Tạo database:

F:\FPT\12-5\CANDbEditor\icons\saveas.pngMở database:

Lưu database 

*HÌNH 3.1. Giao diện chính CAN database*

* **Giao diện chính sẽ chứa thông tin message và signal:**
* **Tiến trình thực hiện:**

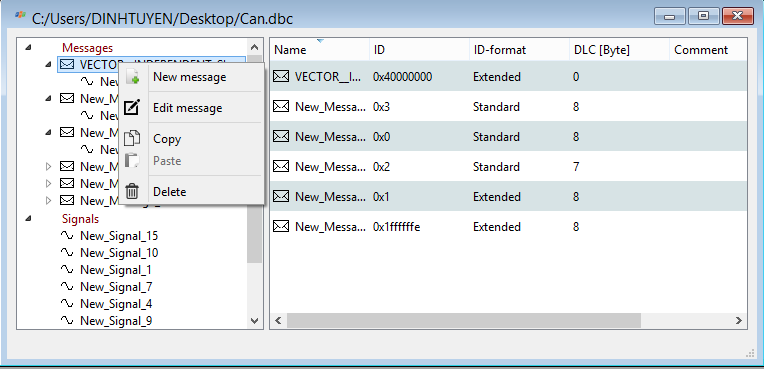
Cửa sổ này được tạo từ QMdiArea và có 2 phần QTreeView và một QStackedWidget chứa các QTableView:

* QTreeView chứa danh sách tên các message và signal cũng như là các signal ở trong message có trong mạng CAN
* QTreeView chứa danh sách tên các message và signal cũng như là các signal ở trong message có trong mạng CAN
* QStackedWidget chứa 3 QTableView:
* m\_MsgsTableWidget
* m\_SignsTableWidget
* m\_ComTableWidget

Tương ứng với 1 bảng chứa danh sách thông tin chi tiết của message/signal và signal trong từng message

Giao diện này được thiết kế mô hình model/view Qt nên tất cả dữ liệu trên view đều được lấy từ model. Chúng ta sẽ tạo ra một model kế thừa từ QAbstractItemModel và tái định nghĩa các phương thức ảo trong lớp cơ sở để làm model cho phần này.

* **Kết quả:**



*HÌNH 3.2. Giao diện CAN database sau khi load dữ liệu lên*

Giao diện chính sau khi thiết kế sẽ có hình dạng như trên. Ngoài phần hiển thị dữ liệu, còn có thêm các context menu hiển thị các chức năng thêm/sửa/xóa khi click chuột phải vào một vị trí thuộc QTreeView hoặc QTableView nào đó.

* + - 1. ***Giao diện của sổ chỉnh sửa:***
* **Tiến trình thực hiện:**

Phần này gồm 3 cửa sổ:

- Cửa sổ chỉnh sửa dùng cho message

- Cửa sổ chỉnh sửa dùng cho signal

- Cửa sổ chỉnh sửa dùng cho signal trong message

Mỗi cửa sổ sẽ là một QDialog, trong từng QDialog sẽ chứa các QTabWidget với nhiều tab chứa thông tin chi tiết về đối tượng được trỏ tới.

Để có được giao diện cân đối trong từng tab, ta sẽ sửa dụng QLayout và các lớp dẫn xuất QVBoxLayout,QHBoxLayout hay QGridLayout.

Tiêu đề của cửa sổ sẽ là tên của message/signal hoặc signal trong message tương ứng.

Mỗi tab được thiết kế giao diện dựa trên các hộ trợ thư viện từ Qt như:

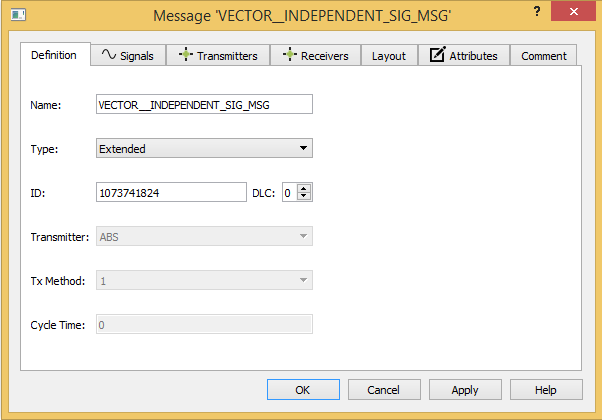
* Các mục tên thông Tin sử dụng QLable: Name, Type v.v
* Các mục dòng chữ giúp điền thông tin sử dụng QLine
* Các mục thông tin lựa chọn sổ xuống sử dụng QComBobox
* Các mục thông tin cho phép tang giảm theo quy định sử dụng QSpinBox
* Các mục thông tin không cho phép chỉnh sửa sẽ sử dụng lệnh

…-> setEnabled(false);

để khóa chức năng sửa đổi.

Mỗi một cửa sổ sẽ có các hành động: OK, Cancel, Apply, Help được tích hợp vào các QPushButton (nhờ vào signal click). Các hành động này sẽ được mô tả chi tiết trong phần thiết kế xử lí dữ liệu.

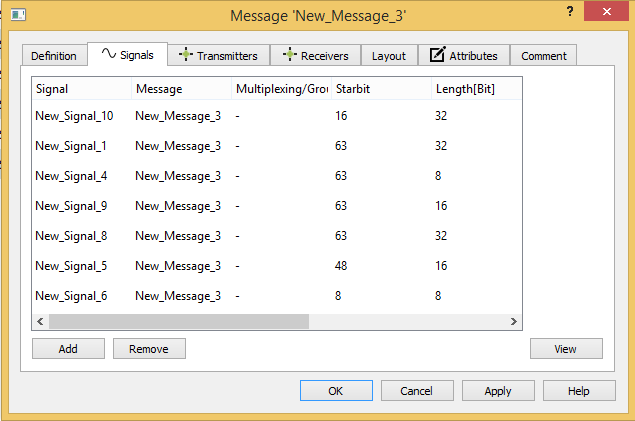
* **Kết quả:**



*HÌNH 3. 3. Giao diện cửa sổ message editor: tab definition*

Để có được hộp thoại như trên và sử dụng cho tất cả các trường hợp cần chỉnh sửa thông tin message (kích phải vào context menu và click đôi vào message trên cả treeview và tableview), ta sẽ định nghĩa một lớp MsgEditor kế thừa từ QDialog.

Tab definition có chứa các thông tin về đối tượng. Tab này là một widget trong đó chứa các tiêu đề là các QLabel và phần nội dung có thể là QLineEdit, QComboBox, QSpinBox được layout theo kiểu QGridLayout.

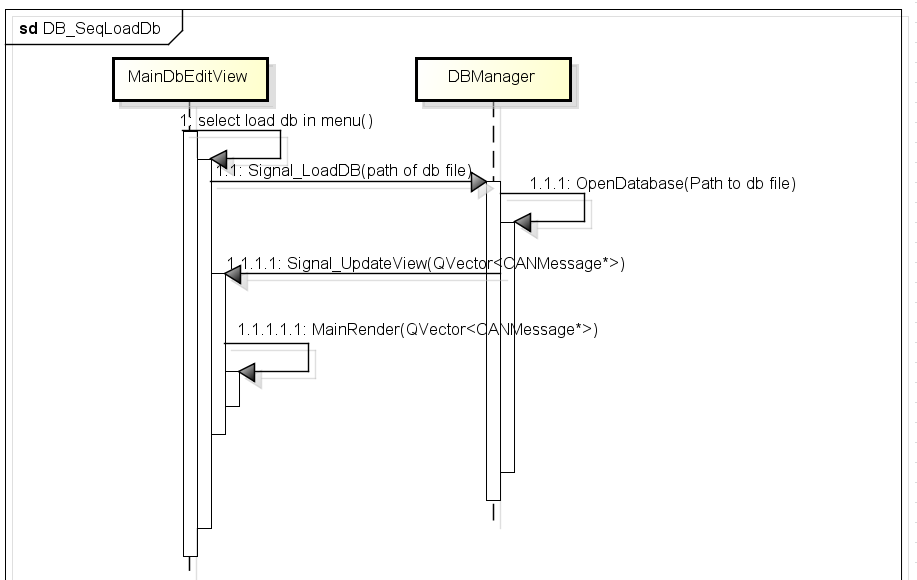


*HÌNH 3.4. Giao diện cửa sổ message editor: tab signal*

Tab signal là một widget dung QVBoxLayout để layout 2 phần: 1 QTableWidget chứa danh sách các signal nằm trong message đó, và các QPushButton Add/Remove/View để thực hiện các hành động thêm/view/xóa các signal trong message.

Ngoài 2 tab chính, còn có các tab phụ khác như Layout, Attribute, Comment…do thời gian hạn chế nên trong đồ án này chúng tôi chỉ tập trung vào các tab chứa nội dung chính.

Ở đây, chúng tôi chỉ đưa vào giao diện cửa sổ chỉnh sửa message , 2 cửa sổ còn lại được thiết kế tương tự.

* + - 1. ***Đánh giá***
* Giao diện đã thể hiện được đầy đủ các thông tin message/signal
* Thông tin hiển thị trong các Table với các mục Header giới thiệu dễ dàng quan sát và chỉnh sửa.
* Giao diện vẫn chưa hoàn thiện và đang chỉnh sửa để có giao diện đầy đủ như các khối Network, ECUs, Environment variables và có Icon dễ nhìn hơn.
  + 1. ***Phần xử lí back-end***
       1. ***Thực hiện các chức năng trên thanh menu màn hình chính:***
* Load database

*HÌNH 3.5. Sequence diagram LoadDB*

* Trình tự đọc database lên từ file.dbc:
* Bước 1: Khi click OpenDatabase 1 hộp thoại yêu cầu chọn database cần chỉnh sửa được hiển thị .
* Bước 2: Sau khi đã chọn database cần chỉnh sửa thì signal Signal\_LoadDB(path) trên lớp MainDbEditView sẽ được phát ra.
* Bước 3: Tại lớp DbManager sẽ có slot OpenDatabase(path) được kết nối với signal Signal\_LoadDB(path) , slot này sẽ gọi các hàm loadDB(path) để lấy thông tin từ database đưa vào các cấu trúc dữ liệu như Qvector chứa danh sách các message/signal
* Bước 4 : Sau khi có được thông tin message/signal thì Signal\_updateview(QVector<CAnMessage\*>) sẽ được gửi đi, view sẽ cập nhật thông qua slot tương ứng
* Phân tích dữ liệu CAN database để đưa dữ liệu từ file lên:
* Ta cần phải phân tích được dữ liệu để có thể load dữ liệu 1 cách chính xác:

Phân tích để nhận diện được Message, Signal và tất cả các thông số của chúng để đưa vào các QVector<CANMessage\*> và QVector<CANSignal\*> dùng làm dữ liệu nguồn cho toàn bộ hệ thống.

* Format của 1 file .dbc
* Format của 1 Message

BO\_ 2 New\_Message\_3: 8 Vector\_\_XXX

Dòng text trên là dạng Message như sau:

BO\_ : Nhận dạng Message

ID của message với 2 loại tương ứng

BO\_ 2 New\_Message\_3: 8 Vector\_\_XXX : CAN Standard

BO\_ 2147483651 New\_Message\_4: 8 Vector\_\_XXX : CAN Extended

New\_Message\_3 : Tên message

8: Chiều dài message, tính theo Byte

Vector\_XXX: Comment

* Format của 1 Signal

SG\_ New\_Signal\_1 : 24|8@1- (1,0) [0|0] "" Vector\_\_XXX

Dòng text trên là dạng Signal như sau:

SG\_ : Nhận dạng Signal

New\_Signal\_1 : Tên của Signal

24 : StarBit

8: Chiều dài của signal, tính theo Bit

Byte Order của Signal với 2 loại tương ứng

SG\_ New\_Signal\_1 : 24|8@1- (1,0) [0|0] "" Vector\_\_XXX : Intel

SG\_ New\_Signal\_10 : 23|8@0- (1,0) [0|0] "" Vector\_\_XXX : Motorola

Value Type của Signal với 2 loại tương ứng

SG\_ New\_Signal\_11 : 32|8@1+ (1,0) [0|0] "" Vector\_\_XXX : UnSigned

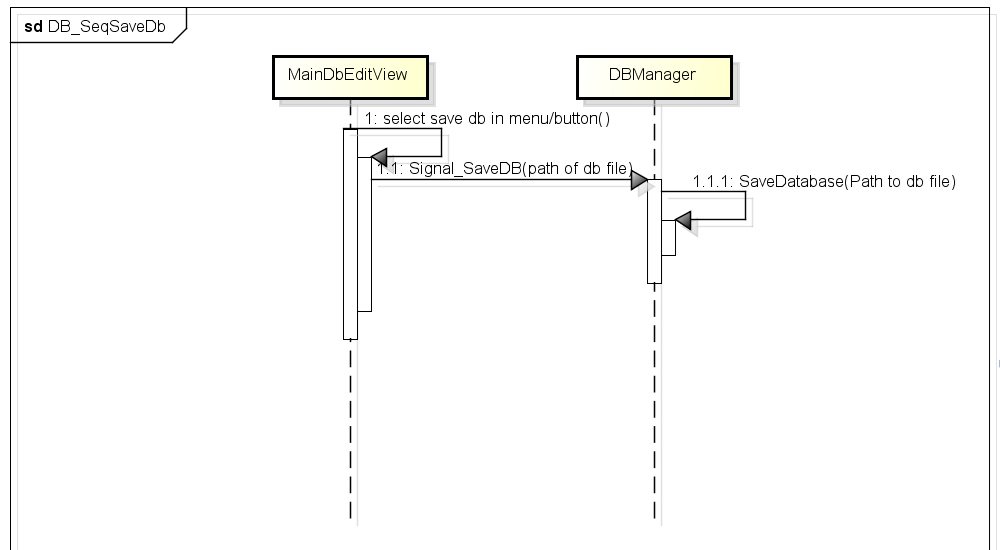
SG\_ New\_Signal\_1 : 24|8@1- (1,0) [0|0] "" Vector\_\_XXX : Signed

(1,0) : (Factor,Offset)

[0|0] : [Min|Max]

Việc đọc dữ liệu từ file được thực hiện nhờ QFile. Dữ liệu được tách thành các chuỗi và chia thành từng phần dùng hàm phương thức split với các biểu thức chính quy trong Qt.

* Kết quả: Đã đọc được database đúng định dạng message/signal và các thông số của chúng vào đúng vị trí.
* Save database
* Tiến trình thực hiện:



*HÌNH 3.6. Sequence diagram SaveDB*

* Bước 1: Tương tác của người dùng sẽ được thực hiện trên lớp view, khi chọn save database thì signal Signal\_SaveDB(path) sẽ được gửi đi
* Bước 2: Tại lớp DBManager, một slot tương ứng với signal Signal\_SaveDB(path) là SaveDatabase(path) , tại đây dữ liệu sẽ được lưu lại tại file có đường dẫn path

Dữ liệu được lưu lại chính là dữu liệu trong vector QVector<CANMessage\*> đã được chỉnh sửa trong quá trình tương tác giữa người dùng với giao diện. Như vậy, database sẽ được thay đổi đúng theo mục đích của người dùng.

* Kết quả:
* Đã save được database gần giống với định dạng của file .bdc, tuy chưa thể thực hiện giống hoàn toàn do còn một số chức năng chưa được hoàn thiện nhưng database vẫn thể hiện được các nội dung chính gồm danh sách các message/signal
* Đánh giá
* Thực hiện được các chức năng như mở file, tạo file và lưu file
* Khi load database, giữ liệu đã đọc được chính xác với file .dbc có sẵn, các thông tin về message, signal đã được lưu trữ đúng vị trí
* Chức năng save as đã hoạt động đúng khi lưu được file dưới dạng .dbc và .txt. So sánh file đã lưu với thông tin của Message của Signal vẫn đúng theo format ban đầu.
  + 1. ***Phần edit cơ sở dữ liệu***
* Chỉnh sửa dùng hộp thoại
* Khi ta click đôi chuột pải vào table view thông tin cụ thể của các signal sẽ được hiển thị tại 1 dialog khác để tiện theo dõi vào chỉnh sửa
* Trong TableView có 3 kiểu click đôi chuột phải như sau:

void slotDoubleClickMsgsTab(QModelIndex index);

* Đối tượng trong lớp MsgEditor sẽ được tạo ra

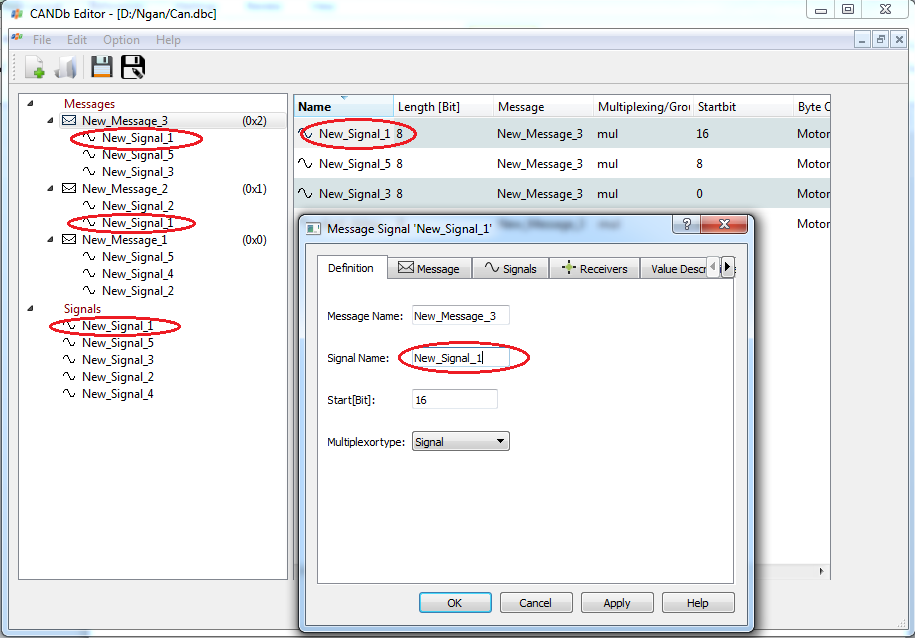
void slotDoubleClickSignsTab(QModelIndex index);

* Đối tượng trong lớp TabDialogMess sẽ được tạo ra

void slotDoubleClickCombTab(QModelIndex index);

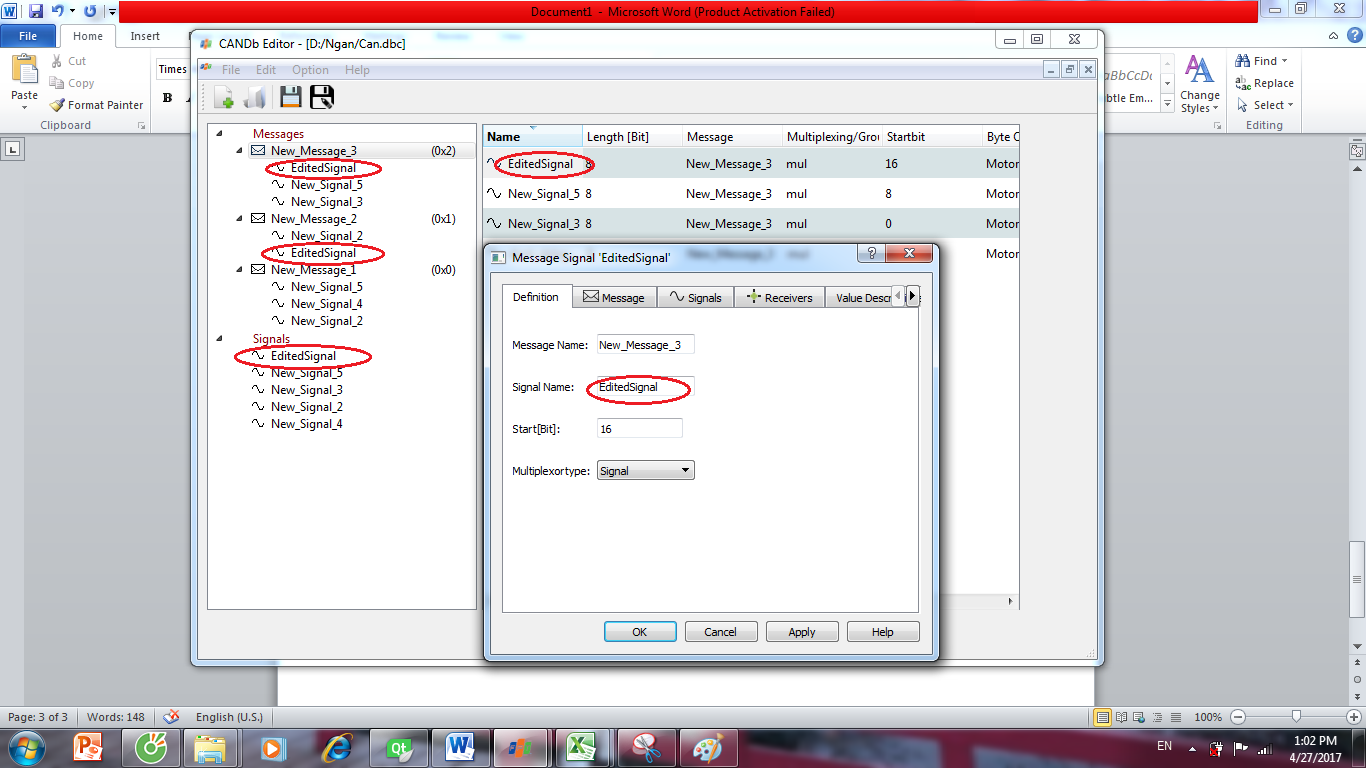
* Đối tượng trong lớp TabDialogSignalMessage sẽ được tạo ra

Tương ứng với mỗi table thì các hàm trên sẽ được gọi và các dialog sẽ show ra.



*HÌNH 3.7. Thông tin Signal trước khi edit*

* Sau khi chỉnh sửa phần Signal Name tại Dialog và nhấn OK,
* Thông tin của Signal đó lập tức thay đổi ở các vị trí:
* Tại Table View
* Tại Tree View
* Tại Source



*HÌNH 3.8. Thông tin Signal sau khi edit*

* Tương tự đối với các thông tin khác của Message hoặc signal ta đều có thể thay eidt được tại phần mềm
* Chỉnh sửa dùng delegate.

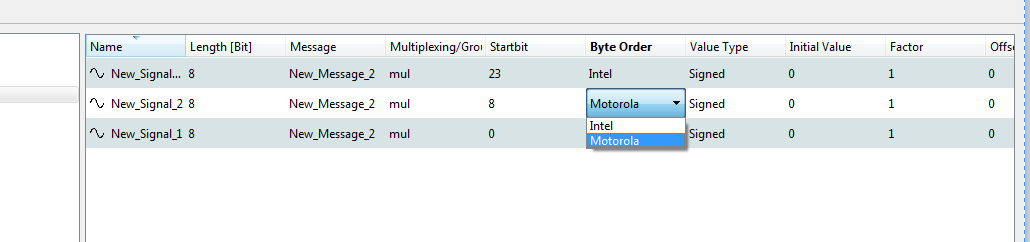
Ngoài việc chỉnh sửa thông qua các Tab, các thông tin có thể được chỉnh sửa bằng cách Delegate trực tiếp tại các table. Đây là một chức năng có được nhờ sử dụng mô hình model/view. Để có thể delegate được dữ liệu tại một vị trí nào đó, ta phải gọi phương thức setItemDelegateForColumn (column,delegateType)

* **Tiến trình thực hiện:**
* Bước 1: Khi click chuột đơn 2 lần (khác với trường hợp doubleClick) một phương thức khởi tạo createEditor() của delegate tương ứng đã đặt tại vị trí đó sẽ được gọi. Một bảng chọn lựa QComboBox, QSpinBox, hay đơn giản 1 QLineEdit sẽ xuất hiện cho phép nguwoif dùng chỉnh sửa dữ liệu.
* Bước 2: Sau khi Edit xong dữ liệu và kích chuột ra khỏi phạm vi cửa sổ chỉnh sửa thì phương thức setModelData () của delegate sẽ được gọi, phương thức này sẽ gọi lại hàm setData () của model tương ứng , làm thay đổi dữ liệu tại model , từ đó view cũng sẽ được cấp nhật theo.
* Bước 3: Thay đổi dữ liệu gốc cũng như trên các model khác.

Delegate chỉ có thể gọi model->setData() để cập nhật dữ liệu tại chính model

chứa dữ liệu đang được delegate mà không thể cập nhật dữ liệu cho các model khác trong chương trình cũng như thay đổi dữ liệu gốc. Vì vậy, khi hàm setData() được gọi trong model, ta phải phát thêm signal updateByDelegate(QModelIndex index) với index chỉ số của vị trí đang được delegate là báo cho lớp quản lí dữ liệu gốc và các model khác biết về sự thay đổi đó, các lớp này sẽ có các slot đáp ứng đối với signal này.

* **Kết quả:**



*HÌNH 3.9. Delegate dữ liệu*

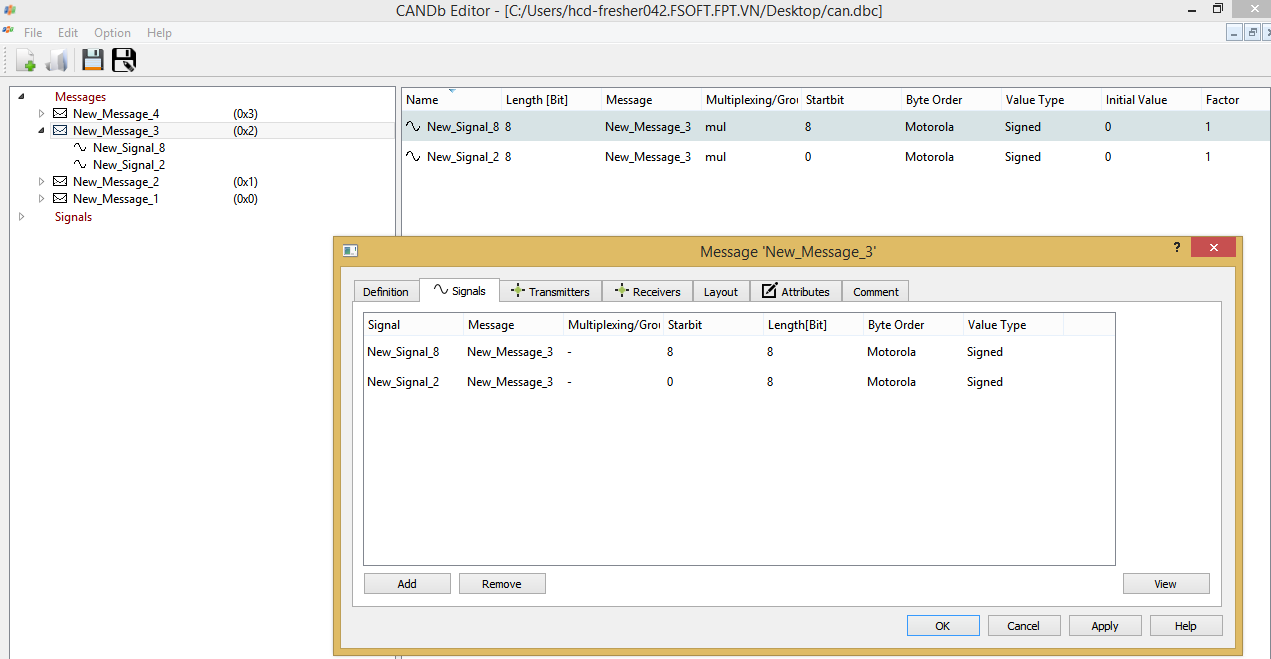
Sau khi delegate tại table, dữ liệu tại tableview, treeview và dữ liệu gốc sau khi lưu lại file đã thay đổi tương ứng. Chức năng chỉnh sửa dùng delegate hoạt động đúng.

* Thêm
* **Tiến trình thực hiện:**
* Add Message
* Bước 1: Khi kích chuột phải và chon add message tại header Message treeview, một signal Signal\_AddMessage () sẽ được phát đi.
* Bước 2: Signal Signal\_AddMessage () sẽ được đáp ứng bởi slot AddnewMessage () trong đó thực hiện các phương thức thuộc lớp DBManager sau:
* Phương thức CreateMessage (): Tạo 1 message mới hoàn toàn với các dữ liệu là các thông số mặc định của message, hiển thị một hộp thoại chỉnh sửa cho các thông số đó.
* Phương thức AddOneMessage (): thêm message vừa tạo vào QVector<CANMessage\*>, phương thức này chỉ được thực hiện khi nhấn Apply hoặc Ok trong hộp thoại chỉnh sửa thêm message.

Step 3: Signal\_AddMessageRespone () được tạo ra sau đó MianRender (QVector<>) sẽ được gọi để cập nhật dữ liệu tại model và cập nhật hiển thị.

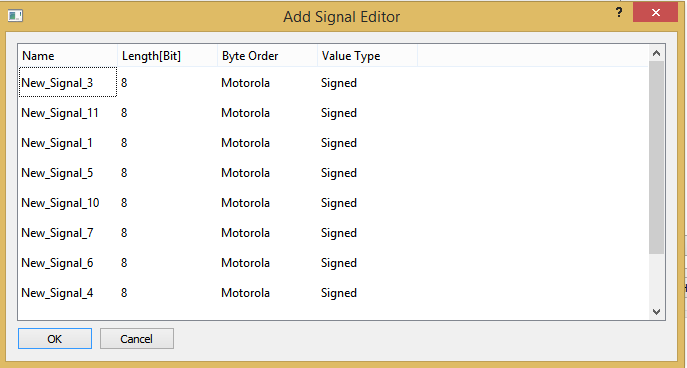
* **Kết quả:**

Tại mỗi dialog message tương ứng, ta chuyển tab sang tab Signal, thông tin của các Signal thuộc Message đó sẽ hiện ra dưới dạng Table



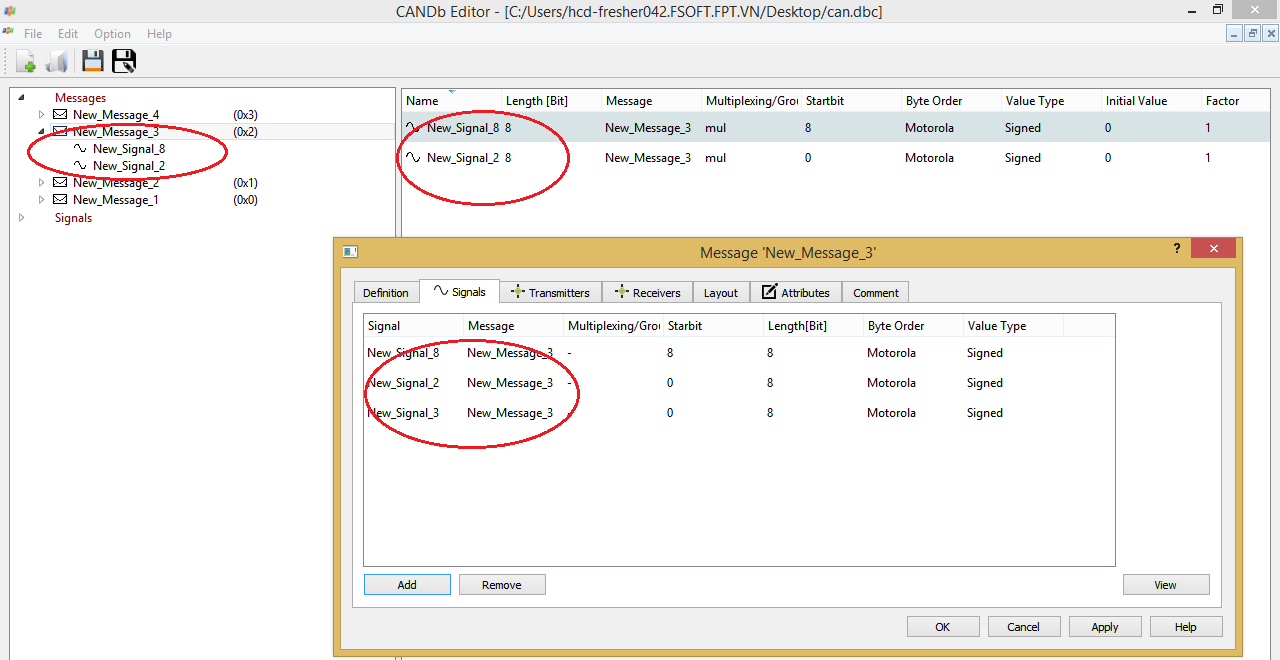
*HÌNH 3.10. List Signal trước khi Add*

Sau khi click vào Nút Add bên góc trái của tab, Bảng thêm Signal sẽ hiện ra với các Signal ngoài Signal đã thuộc Message (Vì 1 message không thể chứa 2 signal giống nhau hoàn toàn)



*HÌNH 3.11. Bảng Thêm Signal*

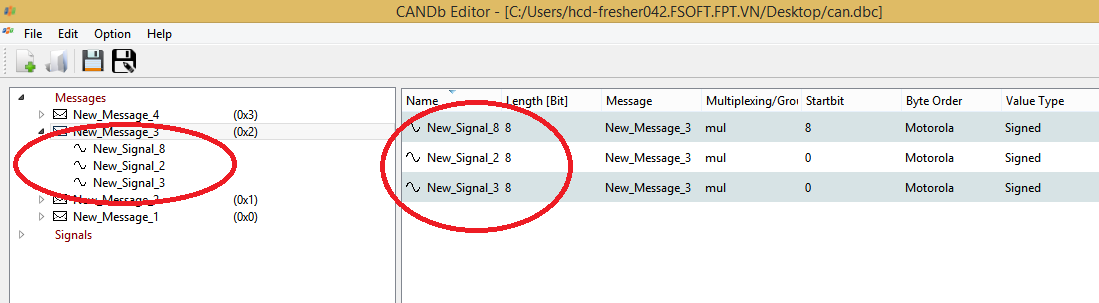
Sau khi chọn 1 hoặc nhiều Signal rồi Nhấn OK thì ngay lập tức nó được thêm vào tab signal, tuy nhiên dữ liệu trên model và dữu liệu nguồn vẫn chưa thay đổi



*HÌNH 3.12. List Signal sau khi được thêm*

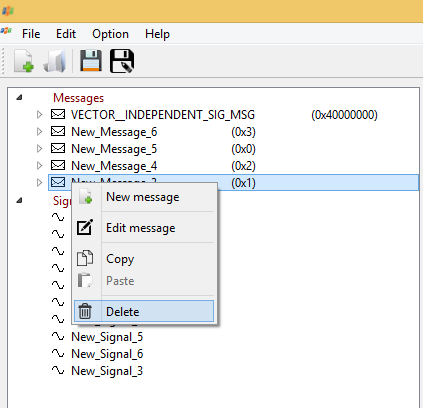
Khi nhấn nút OK tại bảng thêm Signal, Signal được thêm vào List Signal tại Tab Signal nhưng Tại TreeView và TableView chưa xuất hiện Signal mới đồng nghĩa với việc Source chưa được update

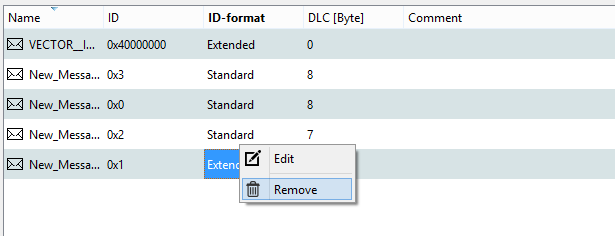
Sau khi nhấn thêm Nút OK tại Tab Signal thì lập tức Tại TreeView, TableView và Source được cập nhật Signal mới



*HÌNH 3.13. List Signal được update tại TreeView và TableView*

* Xóa
* Đối với xóa message (hoặc signal) từ danh sách message (hay signal) , có 2 cách để xóa:
* Xóa bằng cách click chuột phải vào nhấn delete trong ở treeview.



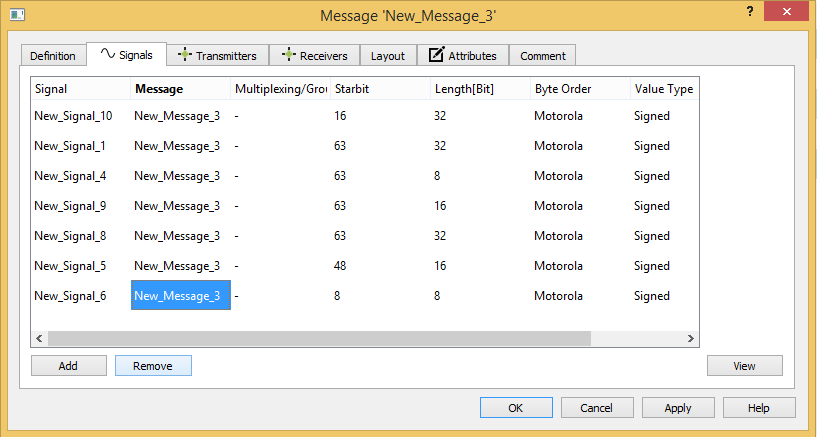


*HÌNH 3.14. Remove message*

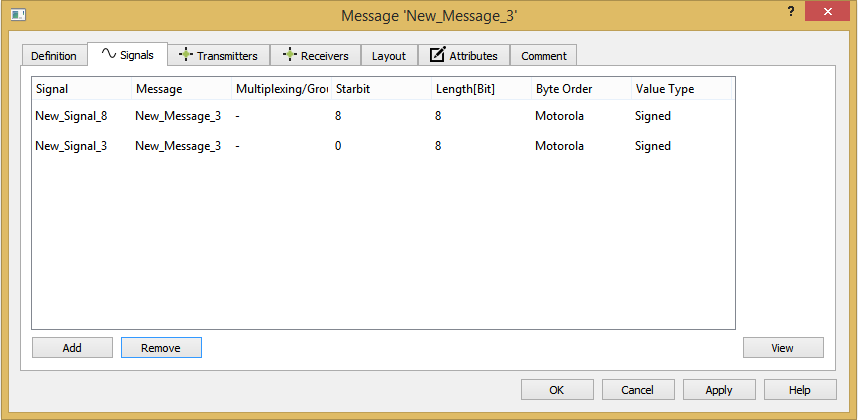
* Xóa bằng cách click chuột phải vào nhấn delete trong ở tableview tương ứng.
* Đối với xóa signal trong message, ngoài 2 cách trên ta có thể xóa dùng hộp thoại quản lí danh sách signal trong message như sau:

Mở hộp thoại chỉnh sửa message, chuyển sang tab signal

Khi chọn 1 Signal và Nhấn OK, Signal đó sẽ được xóa khỏi danh sách Signal trong bảng

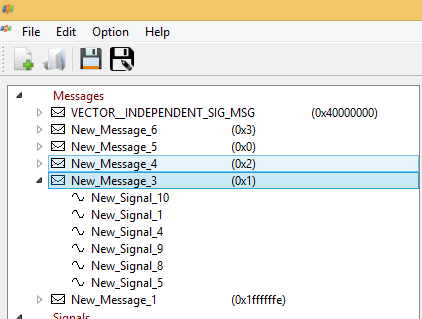


*HÌNH 3.15. List Signal trước khi remove Signal*



*HÌNH 3.16. List Signal sau khi remove Signal*

Dữ liệu chỉ được xóa thực sự khi người dùng nhấn xác nhận Ok hay Apply, lúc đó dữ liệu tại source và treeview sẽ được cập nhật

**

*HÌNH 3.17. List Signal sau khi đã xóa tại TreeView*

* + 1. ***Đánh giá***
* Đã phân tích được đúng format của các Message và Signal để load đúng dữ liệu vào các Tab
* Dữ liệu của các Message và Signal đã được edit và up date
* Đã Delegate được dữ liệu tại các Table
* Hoạt động Thêm Sửa và Xóa đã hoạt động đúng mục đích
  1. **Hướng phát triển của đề tài**
* Đề tài này được thực hiện trong vòng 3 tháng và hiện tại vẫn còn rất nhiều chức năng chưa hoàn thiện, trong tương lai đề tài sẽ được củng cố các chức năng hiện tại, cũng như phát huy thêm nhiều chức năng mới như Setting và Tại TreeView sẽ cung cấp thêm các Header như Network, ECU hoặc Biến môi trường v.v
* Phần mềm sẽ được kết hợp với các phần mềm Simulator và Hardware để nhận các Message trực tiếp từ các ECU và view lên theo mong muốn
  1. **Kết luận chương**

Các kết quả thực hiện đã được trình bày và hệ thống được xem là hoàn thiện các chức năng cơ bản có thể hoạt động tốt. Tuy rằng ban đầu có xuất hiện rất nhiều lỗi nhưng sau quá trình chỉnh sửa chúng tôi đã hoàn thành được đề tài, thực hiện được đúng các chức năng Thêm, Sửa, Xóa

# **KÊT LUẬN**

Sau khi kết thúc đồ án này trong 4 tháng thực hiện, chúng tôi đã hoàn thành được đề tài với đúng các chức năng và yêu cầu đề ra từ đầu.

* Phần mềm đã hiển thị các đúng thông tin Message và Signal trong cơ sở dữ liệu trên giao diện người dùng, đồng thời người dùng có thể thao tác với cơ sở dữ liệu thông qua việc thêm, chỉnh sửa và xóa với các Message, Signal cần thiết.
* Giao diện tuy chưa được xem là mỹ thuật nhưng dễ sử dụng và dễ quan sát đối với người dùng

Phần mềm “Cơ sở dữ liệu ” đóng vai trò quan trọng trong hệ thống “Mô phỏng CAN BUS” nhờ việc hiển thị được các thông tin mà ta có thể biết được việc thay đổi hoặc sai sót của các Message nhận từ ECU để kịp thời phát hiện để chỉnh sửa.

Tuy vẫn còn một vài chức năng chưa được thực hiện nhưng đề tài đã hoàn thành được các chức năng cơ bản có thể sử dụng độc lập. Trong tương lai gần hệ thống sẽ được chỉnh sửa để có giao diện đẹp đẽ hơn và đem lại sự bắt mắt cho người sử dụng.

# **TÀI LIỆU THAM KHẢO**

* doc.qt.io
* Ngôn ngữ lập trinh C++ PGS TS Trần Đình Quế , Học viện Bưu chính Viễn thông

**PHỤ LỤC**