Hướng dẫn tìm hiểu chương trình BKIT MCR

07/2010
Bkit Hardware Club – Bkit4u

MỤC LỤC

Chương 1. Tổng quan 7

Chương 2. Đặt tả cách các module 8

2.1. Giao diện kết nối 8

2.2. Mạch nguồn 9

2.2.1. Mạch ổn áp đầu ra cố định 5V 9

2.2.2. Mạch ổn áp đầu ra 6V 10

2.2.3. Mạch Nguồn trên xe BKIT MCR 10

2.3. Xử lý tín hiệu Mạch Dò Đường 11

2.3.1. Nguyên lý hoạt động 11

2.3.2. Lập trình đọc ADC trên ATmega64 12

2.4. Điều khiển các module trên Mạch VĐK 14

2.4.1. Mô tả chung 14

2.4.2. Khối Led 15

2.4.3. Khối Loa Beep 20

2.4.4. Khối DipSwitch 21

2.4.5. Khối Nút nhấn 22

2.5. Điều khiển RC Servo 24

2.6. Mạch Công Suất và nguyên lý điều khiển động cơ điện một chiều 26

2.6.1. Nguyên lý điều khiển động cơ một chiều 26

2.6.2. Mạch Công Suất 27

Chương 3. Xây dựng giải thuật điều khiển xe 30

3.1. Các hàm cơ bản của chương trình 30

3.2. Cấu trúc chương trình: 31

3.3. Giải thuật xử lý khi qua các đoạn đường thẳng và cong 34

3.4. Sơ đồ trạng thái khi qua đoạn đường cua vuông 38

3.5. Sơ đồ trạng thái khi qua chuyển làn phải.....40

3.6. Sơ đồ trạng thái khi qua chuyển làn trái 41

3.7. Hàm *test ( )* dùng để test các bộ phận xe 41

MỤC LỤC HÌNH

Hình 2-1. Vị trí các mạch điện trên xe BKIT MCR....8
Hình 2-2. Sơ đồ khối các kết nối các mạch trên xe BKIT MCR 8
Hình 2-3. Hình dạng IC LM2576 trong thực tế....10
Hình 2-4. Sơ đồ nguyên lý mạch ổn áp 5V....10
Hình 2-5. Sơ đồ nguyên lý mạch ổn áp 6V....10
Hình 2-6. Mạch nguồn....11
Hình 2-7. Sơ đồ nguyên lý Sensor hồng ngoại....11
Hình 2-8. Mô hình hoạt động Sensor dò đường....11
Hình 2-9. Mạch dò đường....12
Hình 2-10. Kết nối ADC trên ATmega64....13
Hình 2-11. Ví dụ về giá trị biến sensor....14
Hình 2-12. Sơ đồ khối mạch VĐK....14
Hình 2-13. Cách mắt led đơn....15
Hình 2-14. Cách mắc và điều khiển led đơn với VĐK.... 15
Hình 2-15. Led 7 đoạn và sơ đồ bố trí....16
Hình 2-16. Sơ đồ nguyên lý led 7 đoạn (cực dương chung và cực âm chung).... 16
Hình 2-17. Hình dạng một số transistor trong thực tế....16
Hình 2-18. Sơ đồ nguyên lý mạch khóa điện tử nối nguồn. 17
Hình 2-19. Sơ đồ nguyên lý mạch khóa điện tử nối đất. 17
Hình 2-20. Sơ đồ khối khối led....18
Hình 2-21. Sơ đồ nguyên lý khối led....19
Hình 2-22. Sơ đồ nguyên lý khối Loa beep....21
Hình 2-23. Hình dạng DipSW-4 thực tế trên mạch....21
Hình 2-24. Sơ đồ nguyên lý khối DipSW....22
Hình 2-25. Nguyên lý hoạt động của DipSW....22

Hình 2-26. So đồ nguyên lý khối nút nhấn 23
Hình 2-27. Tín hiệu rung nhiễu khi nhấn nút 23
Hình 2-28. RC Servo 25
Hình 2-29. Tín hiệu PWM điều khiển RC Servo 25
Hình 2-30. Động cơ điện một chiều 26
Hình 2-31. Điều khiển chiều quay động cơ điện một chiều 26
Hình 2-32. PWM điều khiển tốc độ động cơ 27
Hình 2-33. Mạch công suất 28
Hình 3-1. Dùng hàm handle (int) điều khiển góc bẻ lái của servo 30
Hình 3-2. Giá trị trả về của hàm sensor_inp 31
Hình 3-3. Sơ đồ giải thuật 31
Hình 3-4. Sensor bắt được nửa line bên trái 32
Hình 3-5. Sensor bắt được nửa line bên phải 32
Hình 3-6. Sensor bắt được nguyên một line 33
Hình 3-7. Mạch dò không vuông góc với vạch trắng ngang 33
Hình 3-8. Mạch dò không nằm giữa đường đua 34
Hình 3-9. Một số trạng thái sensor gặp trên đường đua 34
Hình 3-10. Xe qua đoạn cong 30o. 35
Hình 3-11. Sơ đồ giải thuật trạng thái chính I 36
Hình 3-12. Hình mạch dò bắt được hai line 38
Hình 3-13. Các trạng thái trong khi cua vuông 39
Hình 3-14. Sơ đồ các trạng thái trong khi cua vuông 40
Hình 3-15. Các trạng thái khi chuyển làn phải 40
Hình 3-16. Sơ đồ trạng thái khi qua chuyển lan phải 41

MỤC LỤC BẢNG

Bảng 2-1. Các trạng thái quét led.....20

Bảng 2-2. Mô tả các đường tín hiệu từ Mạch Công Suất đến Mạch VĐK..... 28

Bảng 3-1. Ti lệ vận tốc hai bánh.....36

Bảng 3-2. Các trạng thái led gặp trên đường đua và góc cua tương ứng..... 37

Chương 1. Tổng quan

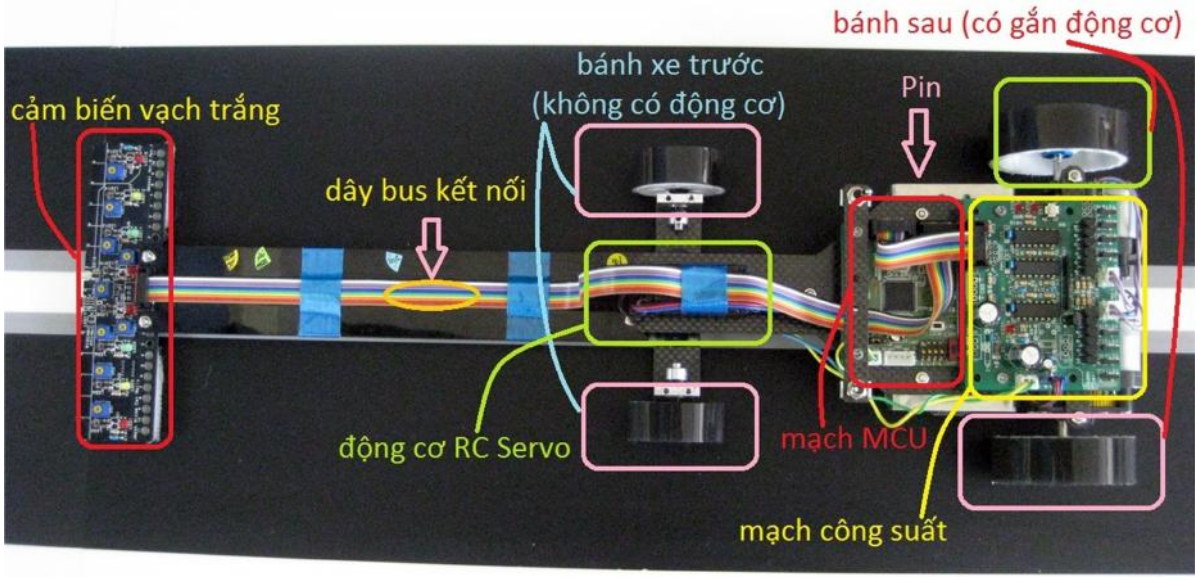
Tài liệu này sẽ hướng dẫn:

Mô tả nguyên lý, hướng dẫn cách điều khiển các module trên mạch VĐK.

Giải thích về chương trình điều khiển chiếc xe này.

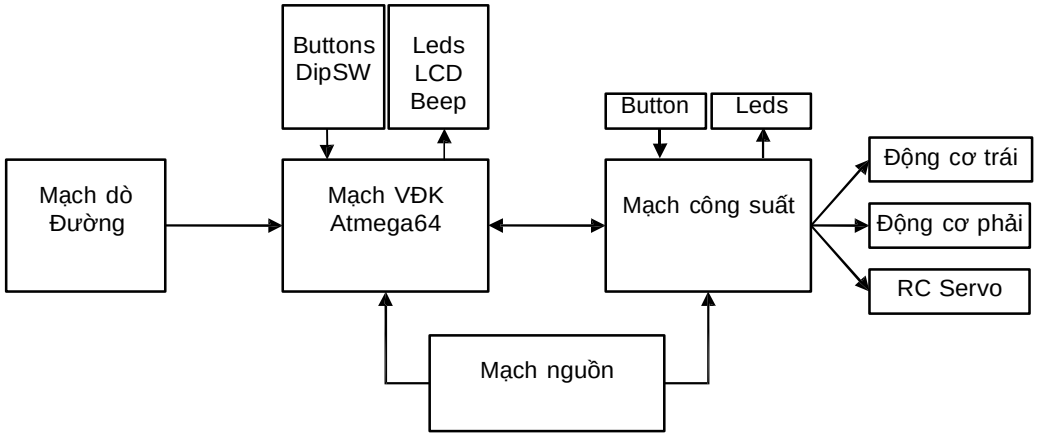
Chương 2. Đặt tả cách các module

2.1. Giao diện kết nối



Hình 2-1. Vị trí các mạch điện trên xe BKIT MCR

Xe BKIT MCR bao gồm Mạch Vi Điều Khiển (VĐK), Mạch Dò Đường, Mạch Công Suất, Mạch Nguồn kết hợp với nhau để điều khiển sự di chuyển của xe, tức là điều khiển hệ thống động cơ trái, phải và động cơ lái rc servo.



Hình 2-2. Sơ đồ khối các kết nối các mạch trên xe BKIT MCR

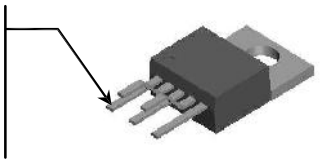
Mạch Dò Đường	Gồm có 8 bộ sensor hồng ngoại nhạy màu, làm nhiệm vụ nhận biết màu Trắng và Đen của đường đua.
Mạch vi điều khiển (VĐK)	Mạch chứa vi điều khiển ATmega64. Vi điều khiển được lập trình làm nhiệm vụ thu thập dữ liệu từ các thiết bị input (sensor, dipsw, nút nhấn), tính toán xử lý những dữ liệu đó và xuất dữ liệu điều khiển các thiết bị output (led, loa beep, LCD, rc servo, motor). Ngoài vi điều khiển chính, mạch còn chứa một số thiết bị I/O (led, nút nhấn, dipsw,…) và các bus để kết nối với các mạch khác trong hệ thống xe. Trên mạch còn tích hợp một module để giao tiếp với máy tính qua cổng USB, giúp cho việc nạp chương trình từ máy tính xuống xe dễ dàng hơn.
Mạch Công Suất	Mạch làm nhiệm vụ nhận tín hiệu điều khiển từ Mạch VĐK để điều khiển các động cơ và RC Servo. Vì động cơ và RC Servo hoạt động ở điện thế cao (12V và 6V) hơn so với điện thế điều khiển từ Mạch VĐK (5V) nên mạch được gọi là mạch công suất, làm nhiệm vụ khuếch đại tín hiệu đầu vào điện với thế thấp thành tín hiệu điều khiển đầu ra với điện thế cao. Ngoài ra trên mạch công suất còn có thêm các led tín hiệu, và một nút nhấn.
Mạch Nguồn **(Đã tích hợp lên mạch công suất)**	Mạch Nguồn làm nhiệm vụ biến điện thế 12V của pin thành điện thế 5V ổn định cung cấp cho Mạch VĐK, và thành điện thế 6V cung cấp cho Mạch Công Suất phục vụ việc hoạt động của RC Servo

2.2. Mạch nguồn

2.2.1. Mạch ổn áp đầu ra cố định 5V

Có rất nhiều IC có thể tạo điện áp ngõ ra là 5V với điện áp vào là 12V như 7805, LM317, LM2672, LM2674, LM2576. Do nguồn 5V tạo ra dùng để cung cấp cho mạch VĐK, trong đó mạch VĐK còn cung cấp nguồn cho Mạch Dò Đường và các IC trên Mạch Công Suất vì thế phải dùng IC ổn áp có dòng cung cấp lớn để tránh mạch Vi xử lý bị reset do dòng không đủ lớn.Và IC LM2576 là một trong những IC đáp ứng được các yêu cầu kỹ thuật trên (dòng cấp tối đa là 3A).

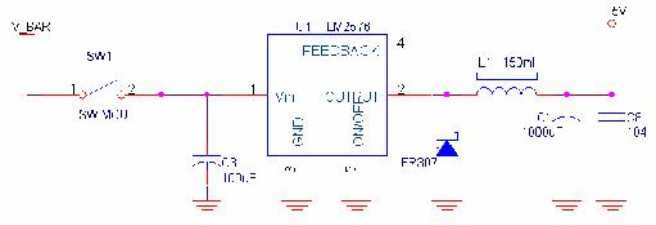
Pin 1	V input
Pin 2	V ouput
Pin 3	GND (-)
Pin 4	Feedback
Pin 5	ON/OFF



Hình 2-3. Hình dạng IC LM2576 trong thực tế

LM2576 gồm một họ IC có thể tạo điện áp ngõ ra cố định với 3.3V, 5V, 12V, 15V và điện áp điều chỉnh được. LM2576 có những đặc điểm tối ưu như: điện áp vào mở rộng đến 40V, dòng ra tải tương đối lớn (có thể tối đa là 3A).

Dưới đây là sơ đồ mạch ứng dụng IC LM2576 tạo điện áp đầu ra 5V.



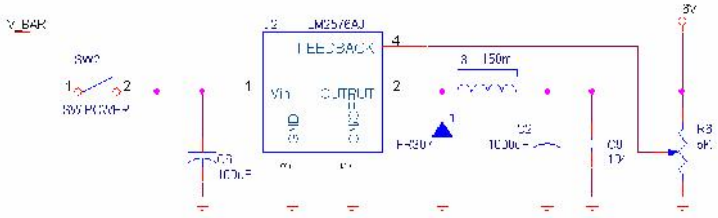
Hình 2-4. Sơ đồ nguyên lý mạch ổn áp 5V

2.2.2. Mạch ổn áp đầu ra 6V

Việc tạo ra điện thế 6V là để cung cấp cho hoạt động của RC Servo.

Dùng LM2576-ADJ là một IC nằm trong họ LM2576, có thể tạo điên áp ngõ ra từ 1.23V đến 37V.

Việc điều chỉnh điện áp ra bằng 6V bằng cách điều chỉnh biến trở R6.



Hình 2-5. Sơ đồ nguyên lý mạch ổn áp 6V

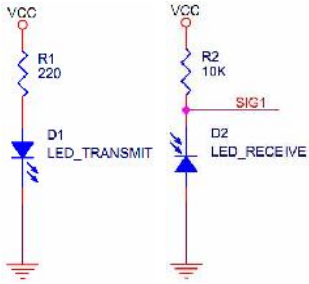
2.2.3. Mạch Nguồn trên xe BKIT MCR

Hình 2-6. Mạch nguồn

2.3. Xử lý tín hiệu Mạch Dò Đường

2.3.1. Nguyên lý hoạt động

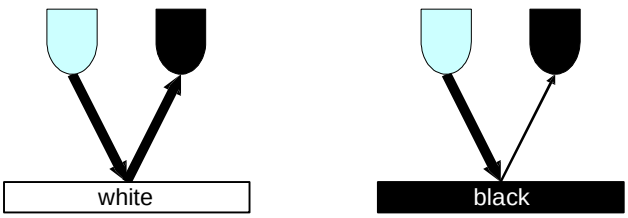
Mạch dò đường có 8 bộ sensor nhạy màu. Mỗi bộ sensor gồm có một led phát tia hồng ngoại và một led thu tia hồng ngoại, kết nối theo sơ đồ nguyên lý sau:



LED_TRANSMIT : Led phát hồng ngoại
LED_RECEIVE : Led thu hồng ngoại

Hình 2-7. Sơ đồ nguyên lý Sensor hồng ngoại

Led phát sẽ phát tia hồng ngoại hướng về phía mặt phẳng đường đua, đường đua sẽ phản xạ tia này lại led nhận. Tùy vào màu sắc đường đua tối hay sáng mà cường độ tia hồng ngoại phản xạ lại ít hay nhiều.



Hình 2-8. Mô hình hoạt động Sensor dò đường

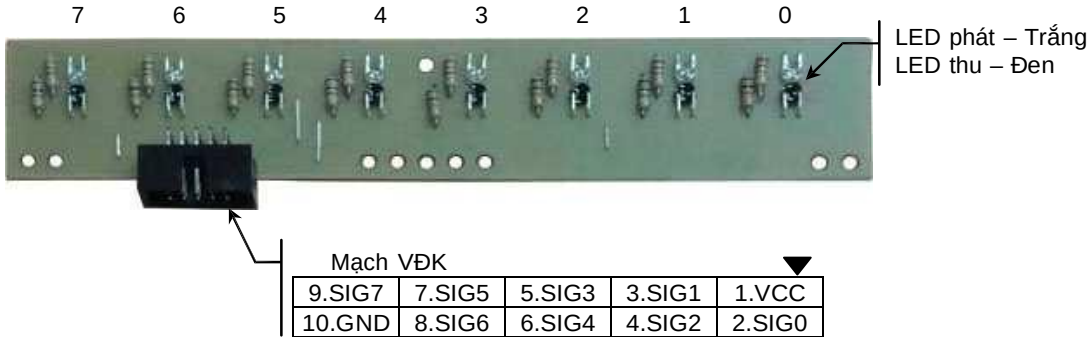
Led thu hồng ngoại hoạt động như một biến trở (điện trở có giá trị thay đổi). Giá trị điện trở của led thu phụ thuộc vào cường độ tia hồng ngoại nó nhận được.

Có thể tóm tắt bằng bảng sau:

Màu	Cường độ tia hồng ngoại phản xạ	Điện trở led thu hồng ngoại	Điện thế tín hiệu đầu ra (SIG0:7)	Giá trị ADC mà VĐK đọc được
Trắng	Lớn	Nhỏ	Nhỏ (<1V)	Nhỏ (<50)
Đen	Nhỏ	Lớn	Lớn(>1,6V)	Lớn (>80)

Vi điều khiển sẽ sử dụng chức năng ADC để đọc giá trị điện thế từ 8 đường tín hiệu (SIG0 SIG7) do Mạch Dò Đường cung cấp, và từ các giá trị đọc được này ta sẽ lập trình tính toán để biết được trong 8 bộ sensor, bộ nào đang trong line trắng, bộ nào ở vùng đen của đường đua.

Dùng bus (dài) kết nối với cổng PORT SENSOR trên Mạch VĐK.

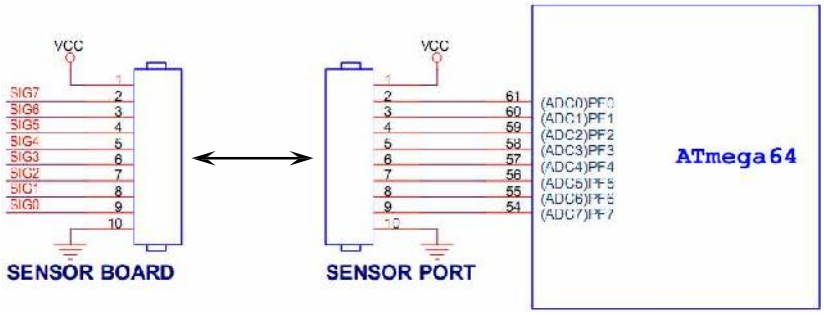


Hình 2-9. Mạch dò đường

2.3.2. Lập trình đọc ADC trên ATmega64

Như đã nói ở trên, để xử lý tín hiệu nhận được từ Mạch Dò Đường ta sẽ dùng chức năng ADC của VĐK ATmega64. ADC (Analog-to-digital converter), tức là chuyển tín hiệu Tương tự Analog thành tín hiệu Số Digital. Cụ thể hơn trong lập trình cho Mạch Dò Đường, ta sử dụng chức năng ADC của ATmega64 để chuyển điện thế nhận được từ các đường tín hiệu SIG0 SIG7 thành giá trị số phục vụ cho việc lập trình tính toán. Giá trị điện thế từ 0V

5V sẽ được chuyển đổi tương ứng với giá trị từ 0 255.



Hình 2-10. Kết nối ADC trên ATmega64

Trong chương trình BKIT MCR 2010, các hàm về ADC được viết trong module adc (thư mục adc gồm file adc.c và adc.h). Một số hàm xử lý chính:

`void` **`adc_init()`**`;` hàm cài đặt các thông số hoạt động cho chức năng ADC của ATmega64. Hàm này chỉ gọi một lần ở đầu chương trình.

`unsigned char` **`read_adc(`**`unsigned char adc_input`**`);` hàm này sẽ thực hiện việc đọc giá trị ở một kênh ADC. Các tham số:

`adc_input`: giá trị từ 0 7, là số thứ tự kênh ADC muốn đọc, tương ứng với SIG0 SIG7 của Mạch Dò Đường.

kết quả trả về của hàm này có giá trị từ 0 255, chính là giá trị của một kênh ADC sau khi chuyển đổi.

`void` **`update_vcompare();`** hàm này sẽ thực hiện việc tính toán và cập nhật giá trị cho mảng giá trị v_compare.

```
unsigned char v_compare[8] ;
```

Mảng v_compare chứa 8 giá trị tương ứng với 8 bộ sensor dò đường, đó là các giá trị được tính toán sao cho khi một sensor ở vùng đen sẽ có giá trị ADC lớn hơn v_compare của nó và ngược lại, khi ở line trắng sẽ có giá trị ADC nhỏ hơn. Nói cách khác giá trị v_compare là giá trị ADC trung gian giữa giá trị ADC lúc sensor nhận màu đen và trắng.

`void read_sensor();` hàm này thực hiện việc định kỳ mỗi 1ms một lần, cập nhật giá trị từ Mạch Dò Đường và đưa kết quả vào biến sensor.

```
unsigned char sensor;
```

Biến sensor có 8 bit, mỗi bit sẽ lưu lại trạng thái của một bộ sensor của Mạch Dò Đường. Bit bằng 1 tức sensor nằm trong line trắng, bit bằng 0 tức sensor nằm trong vùng đen.

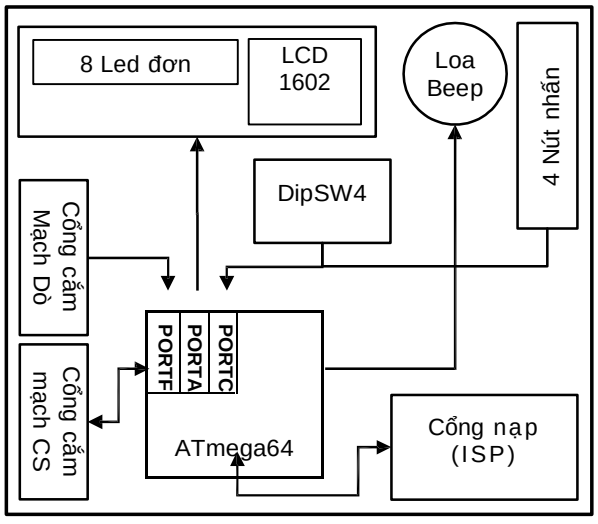
Hình 2-11. Ví dụ về giá trị biến sensor

2.4. Điều khiển các module trên Mạch VĐK

2.4.1. Mô tả chung

Mạch điều khiển xe BKIT MCR sử dụng vi điều khiển AVR ATmega64 của hãng Atmel. Đây là một dòng vi điều khiển khá phổ biến hiện nay cả trong học tập nghiên cứu cũng như trong ứng dụng thực tế.

Để dễ dàng hơn trong việc sử dụng lập trình điều khiển xe BKIT MCR, mạch được thiết kế tích hợp (onboard) một số khối (module) I/O như: 8 led đơn, 2 led 7 đoạn, loa beep, 4 nút nhấn, dipsw4, nạp onboard, … (Hình 2-10).



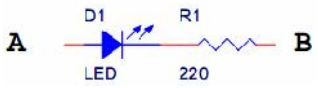
Hình 2-12. Sơ đồ khối mạch VĐK

Chi tiết về thiết kế và cách điều khiển các khối trên mạch VĐK được trình bày dưới đây.

2.4.2. Khối Led

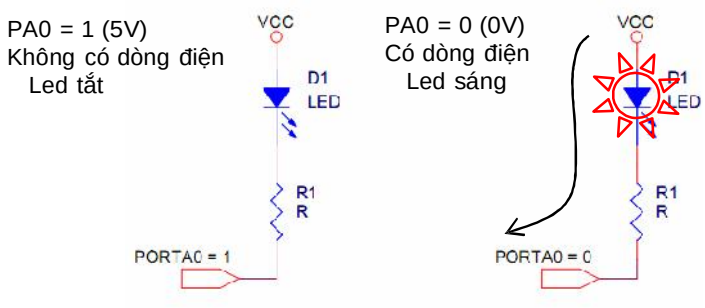
Điều khiển một led đơn

Led (điốt phát quang) thường dùng trên các mạch điện tử để hiển thị thông tin, với 2 trạng thái tắt/sáng. Led thường được mắc nối tiếp với một điện trở (có giá trị khoảng từ 100 đến 2k) để hạn dòng (tránh làm hỏng led), thành một đoạn mạch AB nối tiếp. Khi đó để làm led sáng, ta đặt điện thế +5V vào đầu A và điện thế 0V vào đầu B còn lại. Xem Hình 2-11.



Hình 2-13. Cách mắt led đơn

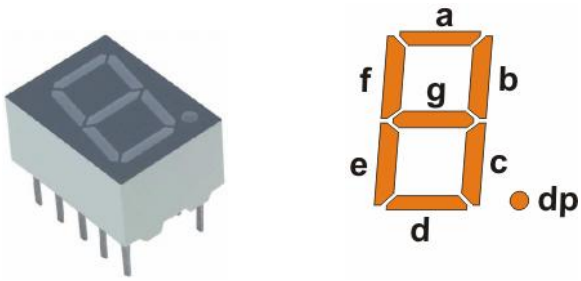
Để điều khiển led bằng VĐK ta cũng áp dụng tương tự. Đầu A ta nối vào VCC (+5V), đầu B nối vào chân VĐK, ví dụ trong hình là nối vào chân PA0 (chân 0 của PORTA) của vi điều khiển ATmega64. Khi PA0 = 1 (5V), led tắt. Và khi PA0 = 0 (0V) led sáng.



Hình 2-14. Cách mắc và điều khiển led đơn với VĐK

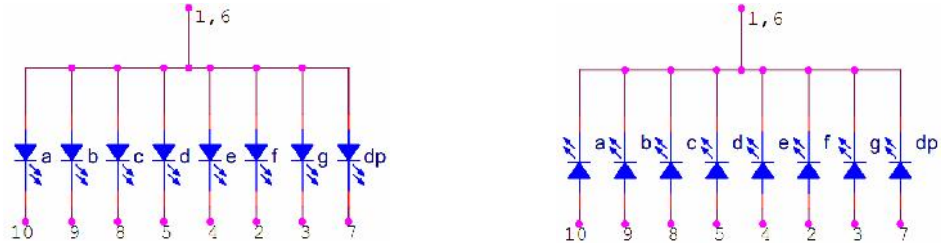
Điều khiển led 7 đoạn

Mỗi led 7 đoạn gồm 8 led đơn ký hiệu là a,b,c,d,e,f,g,dp được sắp xếp ở các vị trí như hình sau:



Hình 2-15. Led 7 đoạn và sơ đồ bố trí

Led 7 đoạn có nhiều hình dạng, kích thước, màu sắc, sơ đồ vị trí chân khác nhau, nhưng xét về nguyên lý thì có thể chia thành 2 loại: cực dương chung và cực âm chung.



a. Cực dương chung **b. Cực âm chung**

Hình 2-16. Sơ đồ nguyên lý led 7 đoạn (cực dương chung và cực âm chung)

Việc điều khiển led 7 đoạn là điều khiển 8 led đơn, trong 8 led đó ta quyết định cho sáng led nào tắt led nào để được sự hiển thị như ta mong muốn. Ví dụ để hiển thị số 3, ta cho sáng led a,b,c,d,g và tắt các led còn lại.

Led 7 đoạn được dùng trên mạch BKIT MCR là loại cực dương chung.

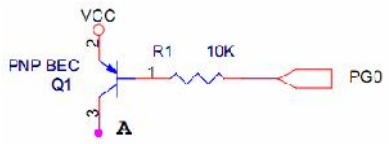
Ứng dụng transistor làm khóa điện tử

Để dễ dàng hơn trong quá trình điều khiển khối led trên Mach VĐK, ta sẽ tìm hiểu thêm về khóa điện tử. Khóa điện tử (còn gọi là công-tắc số) là một công-tắc được điều khiển bằng điện, dùng để đóng/ngắt cho một kết nối nào đó trong một hệ thống mạch.

Hình 2-17. Hình dạng một số transistor trong thực tế

Transistor là một linh kiện điện tử phổ biến, thường được sử dụng như một thiết bị khuếch đại hoặc một khóa điện tử. Xét về cấu tạo, transistor có hai loại là NPN và PNP. Ở đây, ta sẽ không đề cập đến cấu tạo, nguyên lý hoạt động của transistor mà chỉ xét mạch ứng dụng transistor làm một khóa điện tử. Có hai dạng sau:

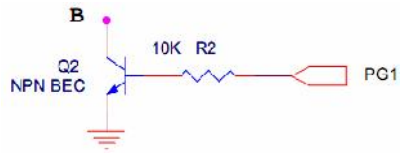
+ Sử dụng transistor PNP làm khóa nối nguồn VCC.



Hình 2-18. Sơ đồ nguyên lý mạch khóa điện tử nối nguồn.

Khóa được điều khiển bởi một chân của VĐK, ví dụ ở đây là chân PG0. Khi PG0=0, khóa đóng (ON), lúc đó đầu A xem như được nối với VCC. Ngược lại khi PG0=1, khóa ngắt (OFF), lúc đó A được cách li với VCC.

+ Sử dụng transistor NPN làm khóa nối đất GND.

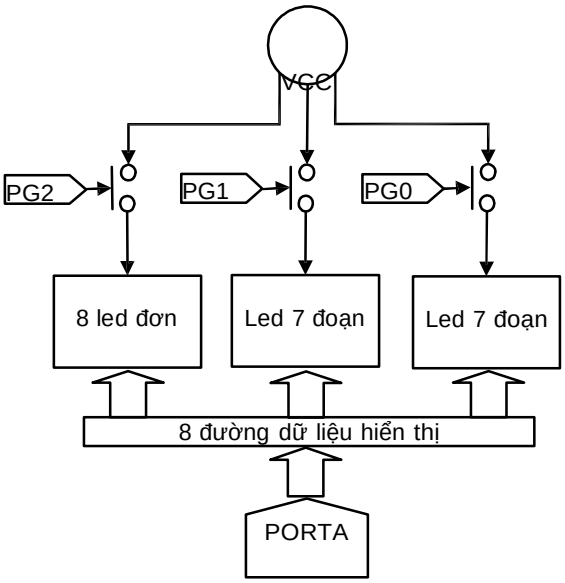


Hình 2-19. Sơ đồ nguyên lý mạch khóa điện tử nối đất.

Tương tự, ví dụ ở đây khóa được điều khiển bở chân PG1. Ngược lại với khóa nối nguồn, khi PG1=1, khóa đóng (ON), lúc đó đầu B xem như được nối với GND. Ngược lại khi PG1=0, khóa ngắt (OFF), lúc đó B được cách li với GND.

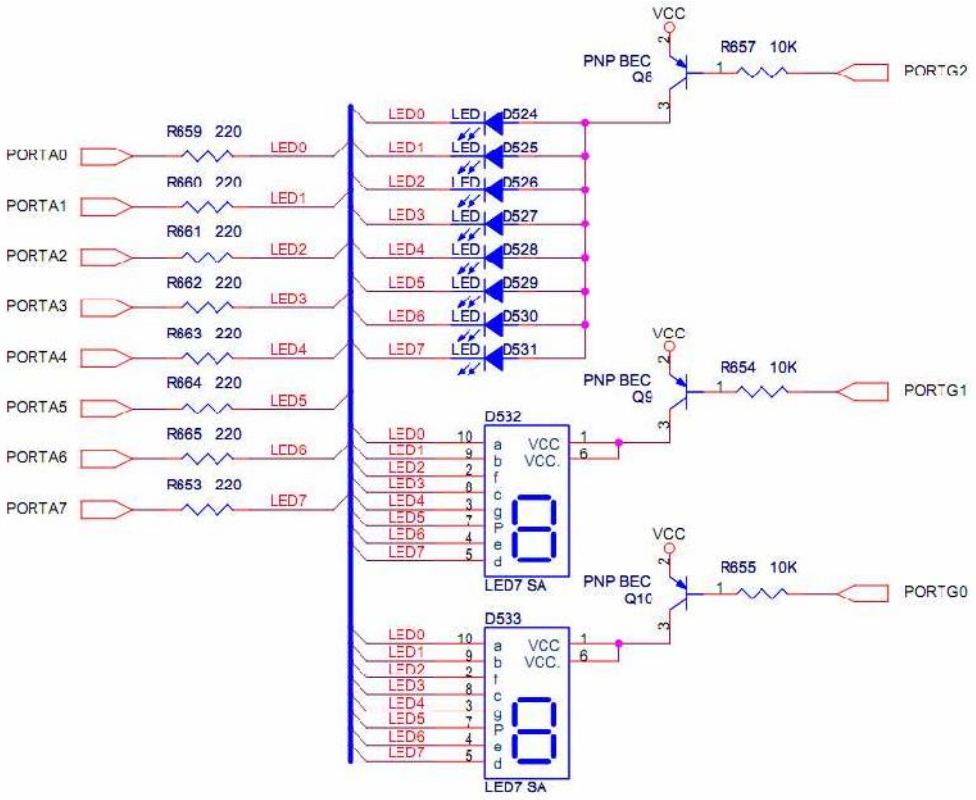
Điều khiển khối led trên mạch VĐK.

Khối led gồm có 2 led 7 đoạn và 8 led đơn được kết nối như hình bên dưới.



Hình 2-20. Sơ đồ khối khối led

Có thể hình dung việc điều khiển led này là điều khiển 3 bộ led, mỗi bộ có 8 led đơn (led 7 đoạn thực chất là 8 led đơn ghép lại). Các đường dữ liệu hiển thị của các bộ led được nối chung với nhau và được điều khiển bởi PORTA. Các chân PG0, PG1, PG2 làm nhiệm vụ đóng ngắt công-tắc, quyết định cho hay không cho bộ led nào hiển thị.



Hình 2-21. Sơ đồ nguyên lý khối led

Ví dụ: để hiện số 3 ở led 7 đoạn thứ nhất ta cài đặt các chân VĐK như sau:

PG2 = 1: tắt bộ led đơn

PG1 = 0: mở led 7 đoạn thứ nhất

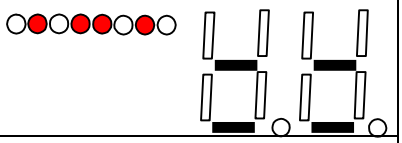
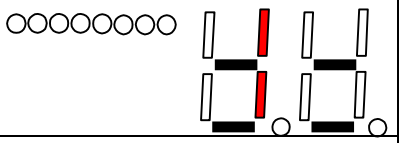
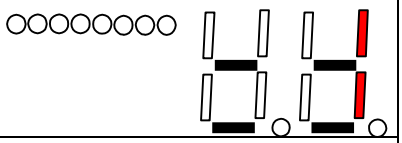
PG0 = 1: tắt led 7 đoạn thứ hai

PORTA = 0x64 (số hex): tức là 0b01100100 (số nhị phân) tương ứng với việc tắt mở các led d,e,dp,g,c,f,b,a

Trong thiết kế, việc nối chung 8 đường dữ liệu hiển thị của 3 bộ led vào PORTA như thế nhằm mục đích tiết kiệm chân I/O của VĐK.

Trong việc điều khiển, nếu dùng cách điều khiển tĩnh như trên thì chúng ta chỉ hiển thị dữ liệu trên một bộ led, hoặc chỉ hiển thị được dữ liệu giống nhau trên các bộ led. Vậy giả sử muốn hiện số 13 trên hai led 7 đoạn (mỗi led hiện một chữ số) thì phải làm sao? Một cách hay dùng để giải quyết vấn đề này gọi là Phương pháp quét led. Đó là, chia việc hiển thị led thành nhiều thời đoạn, mỗi thời đoạn hiển thị một dữ liệu trên một bộ led, và khi thời đoạn

được chia đủ nhỏ, hiệu ứng 24hình/giây được tạo ra và mắt ta sẽ nhìn thấy như thể là dữ liệu khác nhau hiển thị đồng thời trên các bộ led. Có mô tả bằng bảng sau:

Trạng thái	PG2:0	PORTA	
1	011	0xA5	
2	101	0xF5	
3	110	0x64	
(lặp lại trạng thái 1)			

Bảng 2-1. Các trạng thái quét led

Các hàm chính xử lý xuất led trong chương trình

Tất cả các hàm xử lý việc hiển thị led được viết trong module *led* (tham khảo thư mục led gồm file *led.c* và *led.h*). Một số hàm chính:

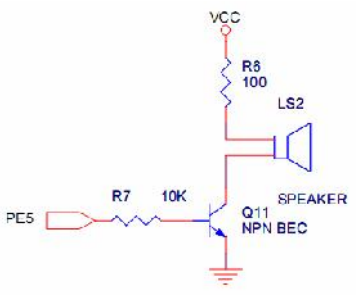
`void led_init();` hàm cài đặt chế độ hoạt động các chân I/O của VĐK trong việc xuất led. Hàm này chỉ gọi một lần ở đầu chương trình.

`void led_mod();` hàm quét led, hàm được gọi bởi ngắt timer định kỳ mỗi một ms một lần. Ở mỗi lần được gọi, hàm sẽ chuyển trạng thái hiển thị led 1 2, 2 3, hoặc 3 1 (Bảng 2-1).

void **led_put(**unsigned char _val**);** hàm xuất giá trị ra 8 led đơn.

void **led7_putHex(**unsigned char _val); hàm xuất giá trị ra 2 led 7 đoạn.

2.4.3. Khối Loa Beep



Hình 2-22. Sơ đồ nguyên lý khối Loa beep

Khối Loa beep tắt/mở bằng một khóa điện tử và được điều khiển bởi chân PE5 của ATmega64.

Khi PE5=1: ON, loa phát ra tiếng kêu.

Khi PE5=0: OFF, loa không phát ra tiếng kêu.

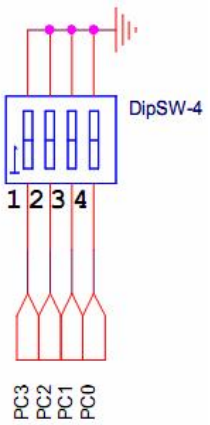
2.4.4. Khối DipSwitch

Hình 2-23. Hình dạng DipSW-4 thực tế trên mạch

DipSW gồm nhiều công-tắc hoạt động độc lập nhưng được gắn chung với nhau thành một thanh. Có nhiều loại DipSW, khác biệt lớn nhất giữa chúng là số lượng công-tắc. Trong Mạch VĐK dùng DipSW-4, tức là DipSW có 4 công tắc.

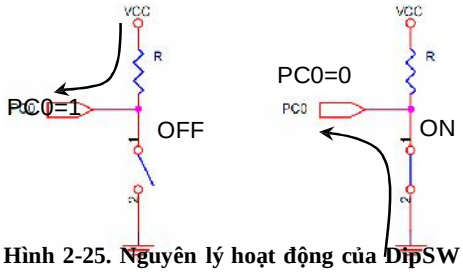
Về nguyên lý thì rất đơn giản, khi gạt một công-tắc lên ON thì hai chân tương ứng ở hai phía công-tắc đó sẽ nối nhau.

Trên Mạch VĐK, DipSW được ứng dụng trên mạch BKIT MCR để cài đặt mode cho xe chạy. Với DipSW-4 ta chọn được 16 mode tứ mode 0 đến mode 15 (0b0000 0b1111).



Hình 2-24. Sơ đồ nguyên lý khối DipSW

Khi lập trình các chân PC3:0 ta sẽ cài đặt ở chế độ input và được kéo lên. Khi công-tắc hở (OFF) ta sẽ nhận được mức 1, và khi công-tắc đóng (ON) ta nhận được mức 0. Khi một chân VĐK cài đặt ở chế độ kéo lên (pull-up), có thể hình dung là bên trong VĐK có một điện trở nối chân đó của VĐK lên VCC.

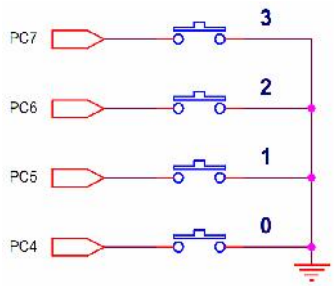


Hình 2-25. Nguyên lý hoạt động của DipSW

2.4.5. Khối Nút nhấn

Mỗi nút nhấn có hai chân, nguyên lý hoạt động rất đơn giản, khi không nhấn nút (OFF) thì hai chân của nút không nối nhau, và ngược lại khi nhấn nút (ON), hai chân của nút sẽ nối nhau.

Trên Mạch VĐK có 4 nút nhấn được kết nối theo sơ đồ sau:

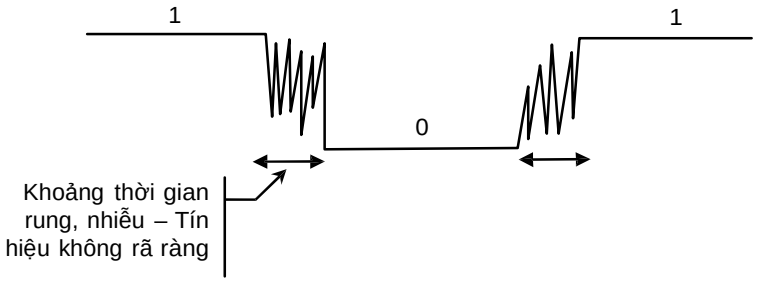


Hình 2-26. So đồ nguyên lý khối nút nhấn

Tương tự như việc lập trình DipSW, các chân PC7:4 ta sẽ cài đặt ở chế độ input và được kéo lên. Khi nút không nhấn (OFF) ta sẽ nhận được mức 1, và khi nút được nhấn (ON) ta nhận được mức 0.

Chống rung phím nhấn

+ Vấn đề rung, nhiễu khi nhấn nút.



Hình 2-27. Tín hiệu rung nhiễu khi nhấn nút

Khi không nhấn nút tín hiệu ở mức cao, khi nhấn nút tín hiệu xuống mức thấp. Trong khoảng thời gian tín hiệu chuyển từ mức cao xuống mức thấp sẽ xảy ra tình trạng rung, nhiễu làm cho tín hiệu không rõ ràng. Mặc dù khoảng thời gian rung, nhiễu là rất nhỏ, chưa tới 1ms (phụ thuộc vào cách nhấn nút và chất lượng nút nhấn), nhưng với tốc độ xử lý rất cao của VĐK thì đây là một vấn đề cần phải giải quyết.

+ Giải pháp chống rung

Có hai giải pháp thường được đưa ra để lựa chọn là giải pháp phần cứng (thực hiện trong giai đoạn thiết kế mạch), và giải pháp phần mềm (lúc lập trình). Ở đây hướng dẫn các bạn một giải thuật chống rung nút bằng phần mềm.

Nội dung: định kỳ cứ sau một khoảng thời gian cố định (1ms) bạn đọc giá trị nút nhấn một lần, so sánh giá trị 3 lần đọc liên tiếp, nếu chúng bằng nhau thì nhận giá trị đó coi như nút nhấn không ở trong trạng thái rung.

Ví dụ: hàm sau được thực hiện mỗi 1ms một lần để lấy giá trị nút nhấn từ PORTC lưu vào biến *key_input*, các biến key0, key1, key2 để lưu 3 giá trị nút nhấn ở 3 trạng thái liên tiếp nhau:

```
void update_key(){
      key2 = key1;
      key1 = key0;
      key0 = PINC;
      if ((key0 == key1) && (key1 == key2)){
          key_input = key0;
      }
}
```

Các hàm điều khiển DipSW và nút nhấn:

Các hàm được viết trong module *input* (tham khảo thư mục *input* gồm file *input.h* và *input.c*).

`void input_init();` hàm khởi tạo các I/O cho việc nhận input. Hàm này được gọi một lần ở đầu chương trình.

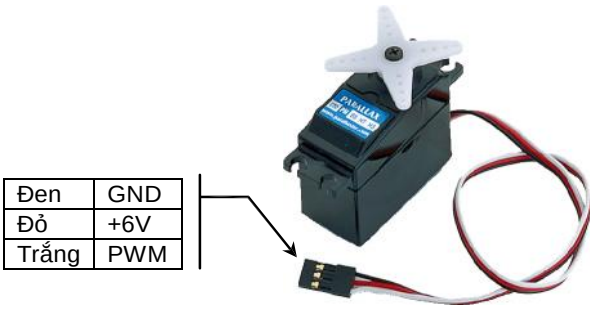
`void update_input();` hàm đọc giá trị các input, xử lý rung, nhiễu và đưa giá trị vào các biến key_input, dipsw_input,… Hàm này được gọi định thời bởi ngắt timer.

`unsigned char get_key(unsigned char _key_id);` hàm kiểm tra xem nút nhấn có thứ tự `_key_id` có được nhấn không. Nếu nút được nhấn hàm trả về 1, ngược lại trả về 0. Tham số `_key_id` nhận một trong các giá trị KEY0, KEY1, KEY2, KEY3 tương ứng với các nút 0,1,2,3 trên Mạch VĐK.

`unsigned char get_dipsw();` hàm lấy giá trị DipSW, kết quả trả về từ 0 đến 15 tương ứng với giá trị cài đặt trên DipSW.

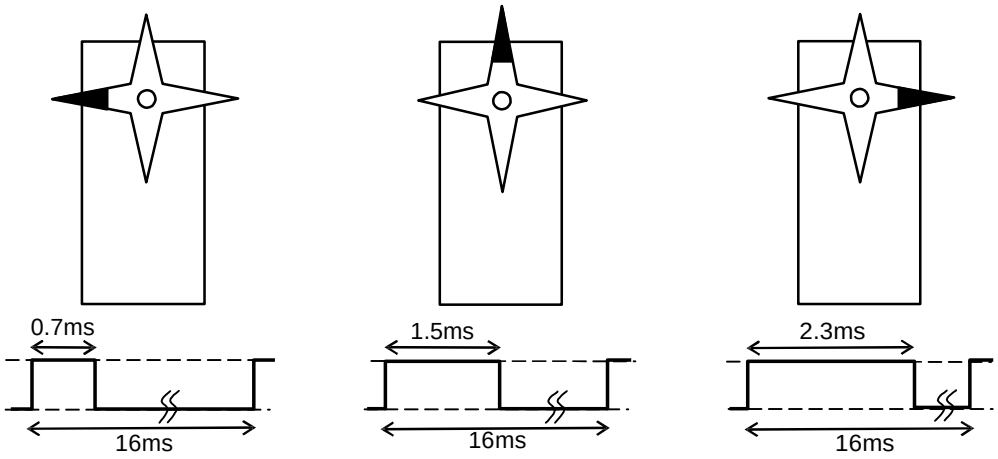
2.5. Điều khiển RC Servo

RC Servo là hệ thống gồm động cơ DC, hộp số, và vi mạch điều khiển. Tùy theo tín hiệu điều khiển mà nó nhận được, RC Servo sẽ quay trục đến một góc xác định trong khoảng từ 0^0 đến 180^0.



Hình 2-28. RC Servo

Điều khiển RC Servo chính là đưa tín hiện vào dây PWM để RC Servo quay theo góc mà mình mong muốn. Tín hiệu PWM cho RC Servo có chu kỳ 16ms, độ rộng xung từ 0.7ms đến 2.3ms, mô tả theo hình bên dưới:



Hình 2-29. Tín hiệu PWM điều khiển RC Servo

Các hàm điều khiển RC Servo:

Các hàm viết trong module *handle* (tham khảo thư mục *handle* gồm file *handle.h* và file *handle.c*).

`void handle_init();` hàm khởi tạo chế độ điều khiển RC Servo, được gọi một lần ở đầu chương trình.

`void handle(int _angle);` hàm cài đặt góc quay cho RC Servo.Tham số `_angle` là góc quay cần cài đặt cho RC Servo, tham số này có giá trị từ -90 đến 90, tương ứng với góc quay từ -90^0 đến 90^0 của đầu xe. Khi gọi handle(0) thì đầu xe ở vị trí giữa.

2.6. Mạch Công Suất và nguyên lý điều khiển động cơ điện một chiều

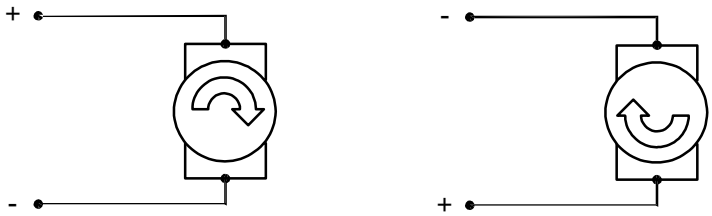
2.6.1. Nguyên lý điều khiển động cơ một chiều

Động cơ điện một chiều (hay động cơ DC) là động cơ hoạt động với dòng điện một chiều.

Hình 2-30. Động cơ điện một chiều

Điều khiển chiều của động cơ điện một chiều

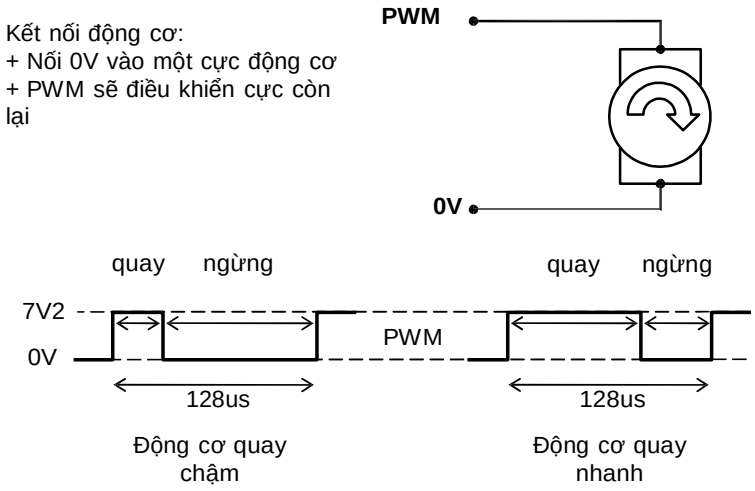
Khi ta cấp điện thế dương (+) vào một cực của động cơ và cấp điện thế âm (-) vào cực còn lại thì động cơ sẽ quay theo một chiều cố định. Và khi ta cấp điện thế ngược lại, đảo chiều dương âm thì động cơ sẽ quay theo chiều ngược lai.



Hình 2-31. Điều khiển chiều quay động cơ điện một chiều

Điều khiển tốc độ động cơ điện một chiều

Để điều khiển tốc độ động cơ điện một chiều ta dùng phương pháp điều xung (PWM). Như trên ta đã biết, khi được cấp điện thì động cơ quay, và khi không cấp điện nữa thì động cơ ngừng hoạt động. Trong một khoảng thời gian rất ngắn 128us (gọi là chu kỳ điều xung T), và chia khoảng thời gian này thành 2 phần, ta sẽ cấp điện cho động cơ một phần thời gian đầu, và ngừng cấp điện ở phần thời gian sau. Lặp lại chu kỳ đó liên tục, như thế động cơ sẽ liên tục ở trạng thái quay-ngừng-quay-ngừng-quay… Vì chu kỳ điều xung là rất nhỏ nên ta sẽ thấy động cơ quay liên tục. Tốc độ của động cơ phụ thuộc và tỷ lệ thời gian động cơ được cấp điện trong một chu kỳ.



Hình 2-32. PWM điều khiển tốc độ động cơ

Trong một chu kỳ, tỷ lệ thời gian động cơ được cấp điện càng nhiều thì động cơ quay càng nhanh.

Các hàm điều khiển động cơ

Các hàm cài đặt và điều khiển động cơ viết trong module *speed* (tham khảo thư mục *speed* gồm file *speed.h* và *speed.c*).

`void speed_init();` hàm khởi tạo chức năng PWM của ATmega64 để điều khiển động cơ, hàm được gọi một lần ở đầu chương trình.

`void speed(int _left_speed, int _right_speed);` hàm cài đặt tốc độ và chiều cho hai động cơ bánh xe. Các tham số:

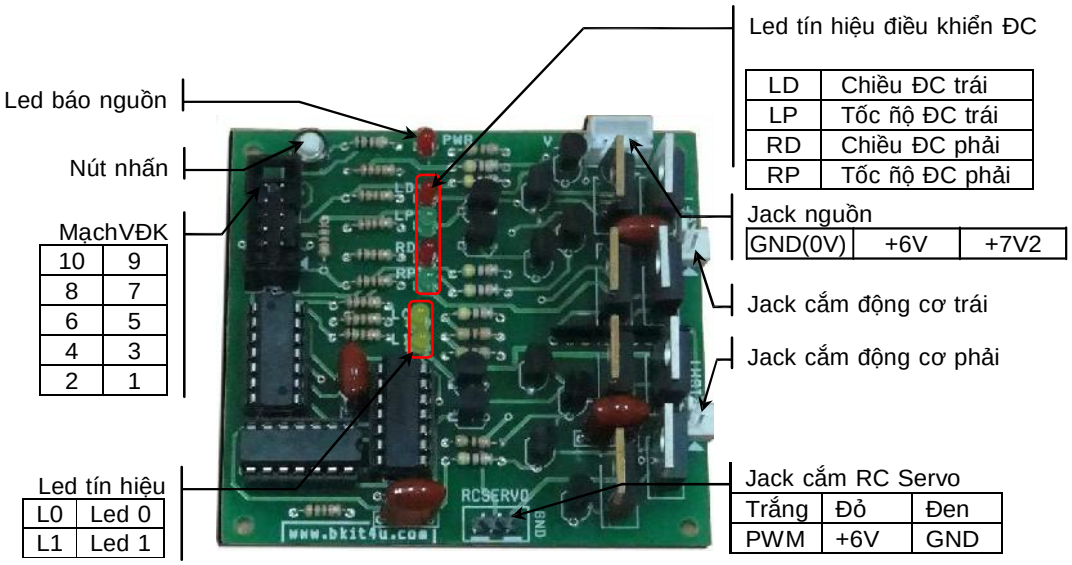
`_left_speed`: có độ lớn từ 0 đến 255 tương ứng với tốc tộ từ 0% đến 100% của động cơ trái. Có dấu dương (+) nếu chạy TIẾN và âm (-) nếu chạy LÙI.

`_right_speed`: tương tự, đây là tham số điều khiển tốc độ và chiều cho động cơ phải.

2.6.2. Mạch Công Suất

Như đã nói ở trên, Mạch Công Suất có nhiệm vụ khuếch đại tín hiệu đầu vào điện với thế thấp từ VĐK thành tín hiệu điều khiển đầu ra với điện thế cao để điều khiển động cơ.

Các thành phần trên mạch được mô tả trong hình sau:



Hình 2-33. Mạch công suất

Jack kết nối Mạch Công Suất với Mạch VĐK gồm 10 chân, mô tả chi tiết trong bảng sau:

Tên chân	Kết nối với ATmega64	Chi tiết	"0"	"1"
1	(+)	+5V		
2	Mạch ⇦ PG3	LED1	Tắt	Sáng
3	Mạch ⇦ PG4	LED0	Tắt	Sáng
4	Mạch ⇦ PE3	Tín hiệu PWM RC Servo	Tín hiệu PWM	
5	Mạch ⇦ PB4	PWM động cơ bên phải	Tín hiệu PWM	
6	Mạch ⇦ PD0	Chiều quay của động cơ phải	Bình thường	Đảo chiều
7	Mạch ⇦ PD1	Chiều quay của động cơ trái	Bình thường	Đảo chiều
8	Mạch ⇦ PB7	PWM động cơ bên trái	Tín hiệu PWM	
9	Mạch PE2	Nút nhấn	Được nhấn	Không nhấn
10	(–)	GND		

Bảng 2-2. Mô tả các đường tín hiệu từ Mạch Công Suất đến Mạch VĐK

*** Ghi chú:

+ Kí hiệu "Mạch PE2" nghĩa là đọc tín hiệu từ Mạch Công Suất đến VĐK.

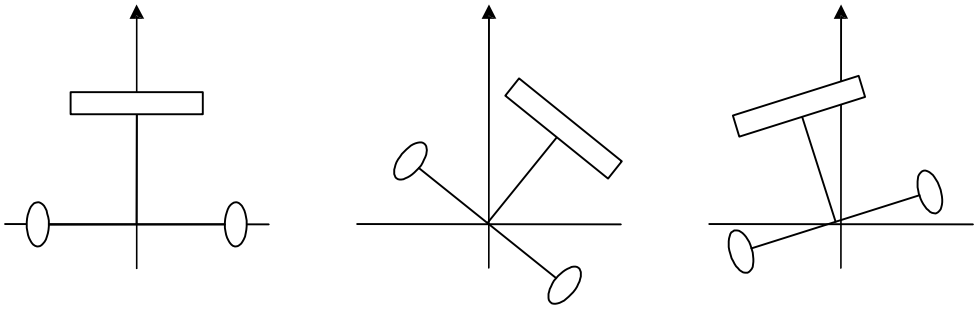
+ Kí hiệu “Mạch ⇦ PB7” nghĩa là xuất tín hiệu từ VĐK đến Mạch Công Suất.

Chương 3. Xây dựng giải thuật điều khiển xe

3.1. Các hàm cơ bản của chương trình

`void speed(int left, int right);` điều khiển duty cycle cho hai bánh phát động bên trài và phải. Giá trị truyền vào left và right là từ -255 đến 255.

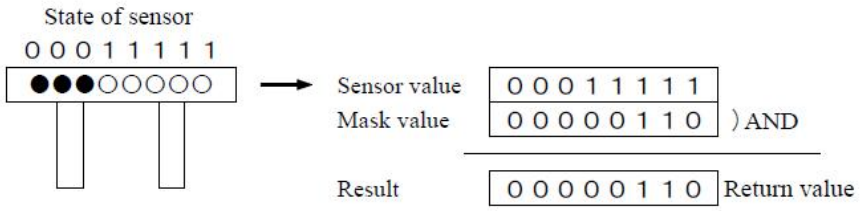
`void handle(int angle);` điều chỉnh góc của động cơ servo so với phương vuông góc với thân xe. Đơn vị của đối số angle là độ (0). Xem hình 5.1

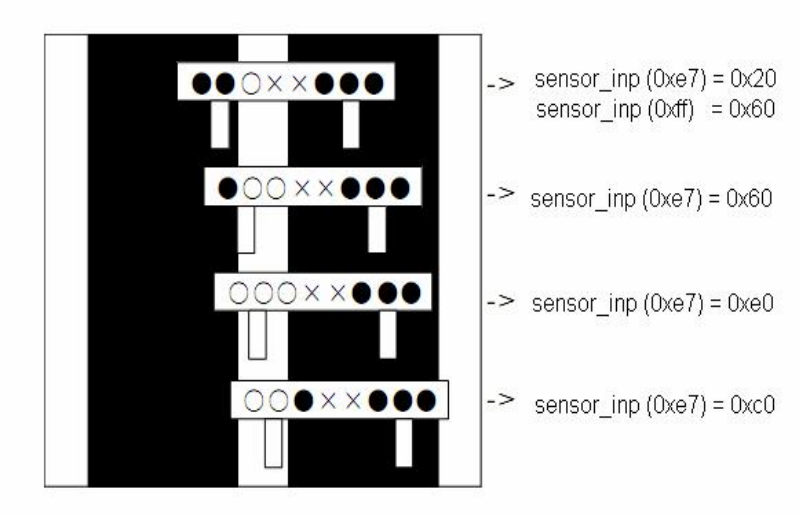


handle (0) **handle (30)** **handle (-15)**

Hình 3-1. Dùng hàm handle (int) điều khiển góc bẻ lái của servo

`unsigned char sensor_inp(unsigned char MASK);` hàm trả về giá trị của 8 sensor dò trước đầu xe sau khi AND với MASK. Ví dụ:

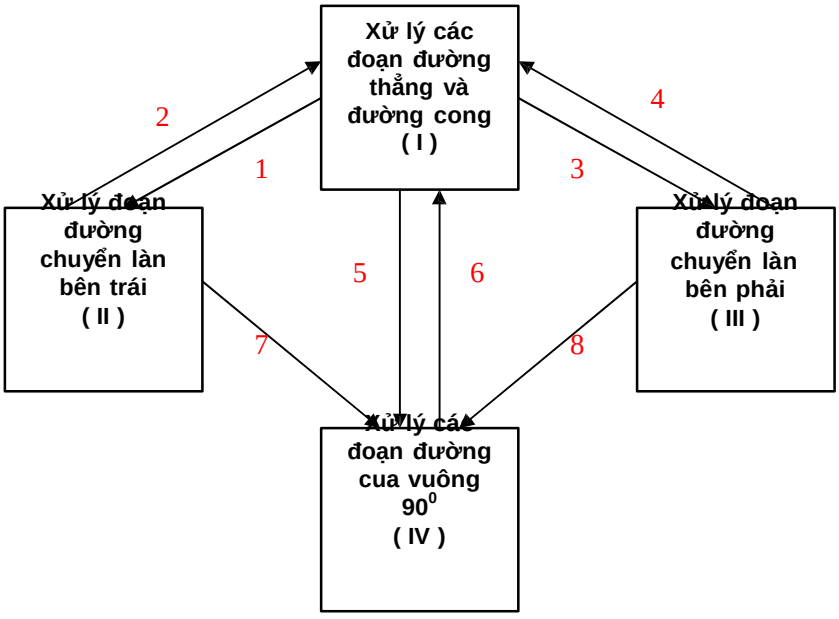




Hình 3-2. Giá trị trả về của hàm sensor_inp

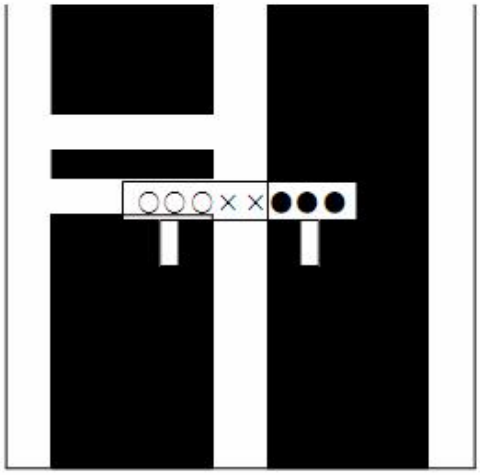
3.2. Cấu trúc chương trình

- Chương trình chia làm 4 trạng thái chính:



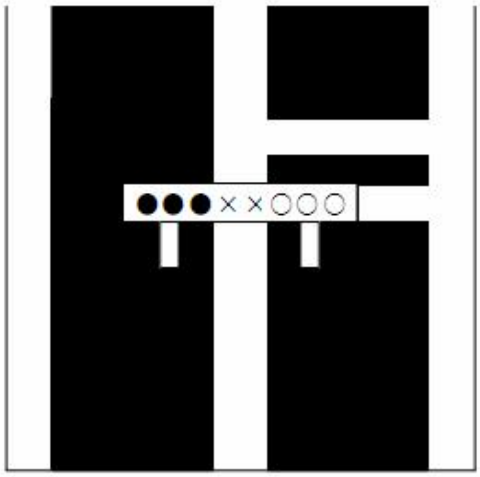
Hình 3-3. Sơ đồ giải thuật

- Bước chuyển (1: I -> II): khi thấy nửa line bên trái.



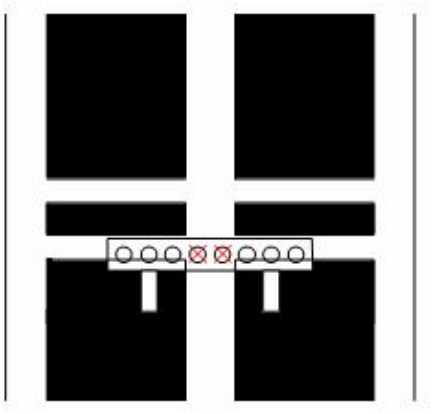
Hình 3-4. Sensor bắt được nửa line bên trái

- Bước chuyển (2: II -> I): khi đã đi qua được đoạn đường chuyển làn trái.



Hình 3-5. Sensor bắt được nửa line bên phải

- Bước chuyển (3: I -> III): khi thấy được nửa line bên phải.
- Bước chuyển (4: III -> I): khi chạy qua được đoạn chuyển làn phải.
- Bước chuyển (5: I -> IV), (7: II -> IV), (8: III -> IV): ngay khi thấy được nguyên một line.



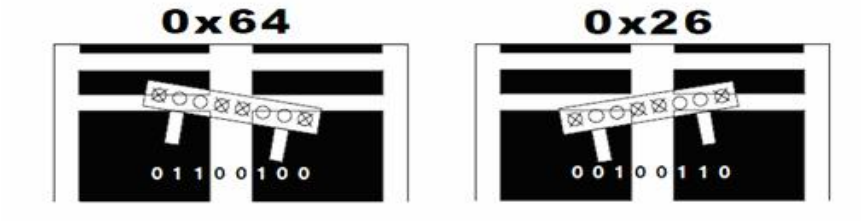
Hình 3-6. Sensor bắt được nguyên một line

- Bước chuyển (6: IV -> I): khi chạy xong được đoạn đường cua 90o.

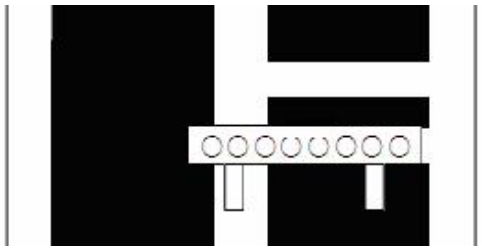
* Chú ý:

Ưu điểm của sơ đồ giải thuật là giải quyết được lỗi nhầm lẫn giữa đoạn đường cua vuông thành đoạn đường chuyển làn. Thường thì không phải lúc nào mạch dò cũng song song với vạch trắng ngang, có khi nữa bên mạch dò bắt được vạch trắng trước (hình 5.7) sinh ra lỗi nhằm lẫn đoạn đường cua vuông thành chuyển làn, việc này thường xuyên xảy ra.

Khi mạch dò đi ngang qua nữa vạch trắng mà mạch dò nằm lệch qua phía nữa vạch trắng thì có thể bị nhằm lẫn đoạn đường chuyển làn thành đoạn cua vuông (hình 5.8), nhưng thường thì ở đường đua thật đúng chuẩn qui định thì lỗi này ít xảy ra. Lỗi này gặp khi sau đường cong là hai nữa vạch trắng chuyển làn xe chưa kịp chỉnh mạch dò vào giữa đường đua. Khi xảy ra lỗi này cũng không đáng ngại vì với kỹ thuật qua đoạn cua vuông thì xe cũng dễ dàng qua được đoạn đường chuyển làn chỉ là khi này vận tốc xe chậm hơn so với khi không nhầm lẫn thôi.



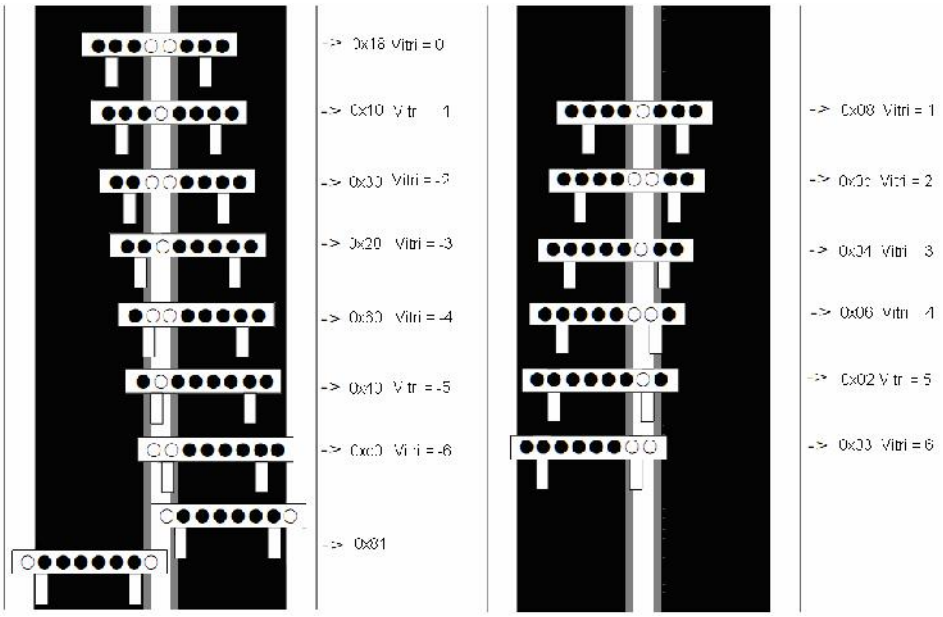
Hình 3-7. Mạch dò không vuông góc với vạch trắng ngang



Hình 3-8. Mạch dò không nằm giữa đường đua

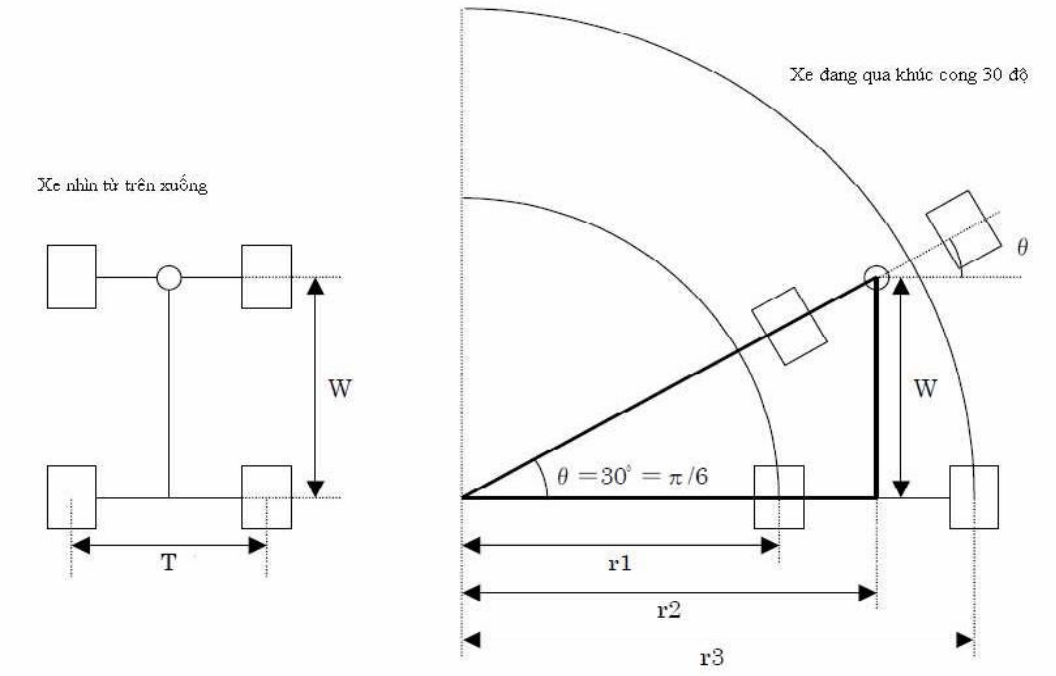
3.3. Giải thuật xử lý khi qua các đoạn đường thẳng và cong

- Các trạng thái sensor có thể gặp thể hiện tương đối độ lệch hướng của xe so với phương đường đua:



Hình 3-9. Một số trạng thái sensor gặp trên đường đua

- Cách tính tỉ lệ vận tốc hai bánh sau theo góc bẻ lái của hai bánh trước:



Hình 3-10. Xe qua đoạn cong 30°.

- Từ hình vẽ ta có:

$$\tan \quad \frac{W}{r2}$$

$$\Rightarrow r2 \quad \frac{W}{\tan} \qquad (1)$$

$$r1 \quad r2 \quad \frac{T}{2} \qquad (2)$$

$$r3 \quad r2 \quad \frac{T}{2} \qquad (3)$$

Từ (1), (2), (3) suy ra tỉ lệ vận tốc hai bánh theo góc bẻ lái ( ):

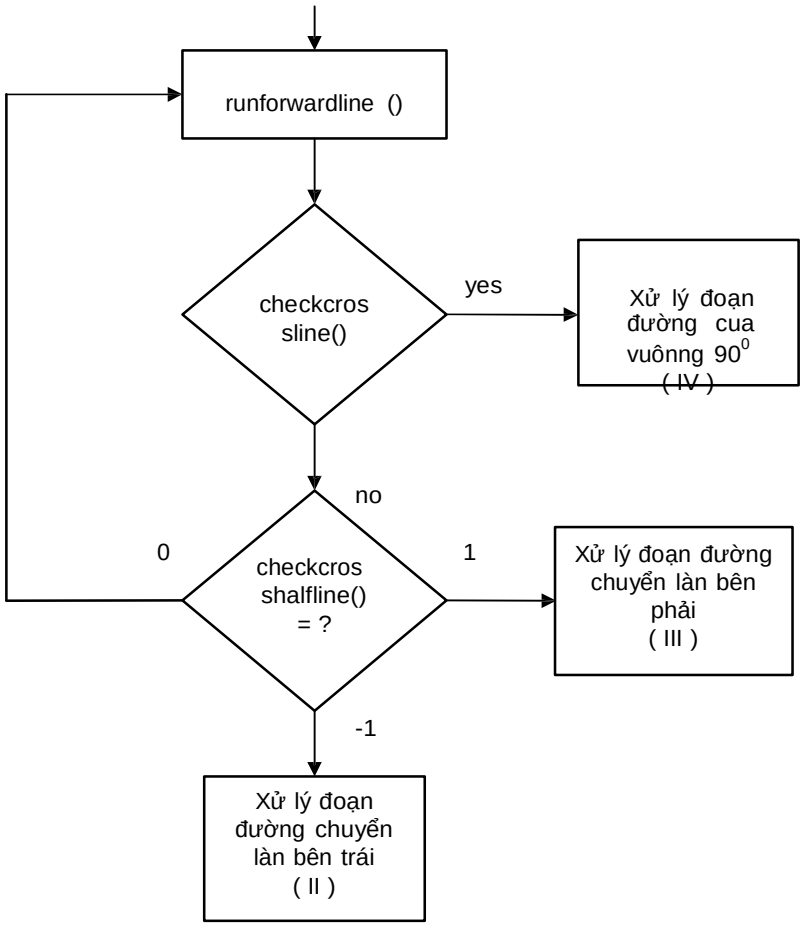
$$\frac{r1}{r2} \quad 100 \quad \frac{\frac{W}{\tan} \quad \frac{T}{2}}{\frac{W}{\tan} \quad \frac{T}{2}} \quad 100 \quad (\%)$$

Ví dụ:
T = 0,14 m
W = 0,175 m

Degree (θ)	r2	r1	r3	r1/r3 * 100
1	10.02574	9.95574	10.09574	99
2	5.01134	4.94134	5.08134	97
3	3.33920	3.26920	3.40920	96
4	2.50262	2.43262	2.57262	95
5	2.00026	1.93026	2.07026	93
6	1.66501	1.59501	1.73501	92
7	1.42526	1.35526	1.49526	91
8	1.24519	1.17519	1.31519	89
9	1.10491	1.03491	1.17491	88
10	0.99247	0.92247	1.06247	87

Bảng 3-1. Ti lệ vận tốc hai bánh

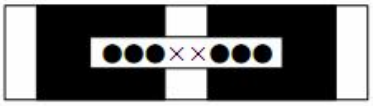
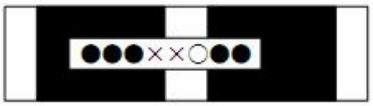
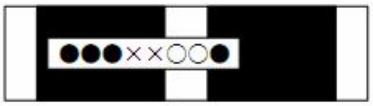
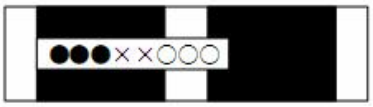
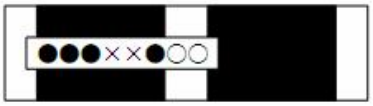
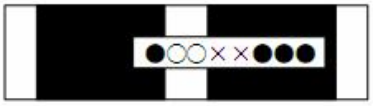
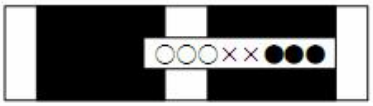
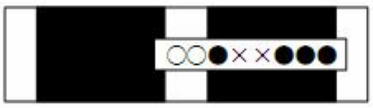
- Sơ đồ giải thuật của trạng thái chính I (xem hình 5.3):



Hình 3-11. Sơ đồ giải thuật trạng thái chính I

- Ý tưởng khi qua các đoạn đường thẳng và cong:

Ta dựa vào trạng thái sensor để biết được độ lệch hướng của chạy của xe so với phương của đường đua, ứng với một trạng thái sensor ta chọn một góc bẻ lái hợp lý (bằng cách dùng hàm *handle (int)*) sao cho xe có xu hướng di chuyển về phía chính giữa đường đua, với độ lệch hướng càng lớn thì ta chọn góc bẻ lái càng lớn.

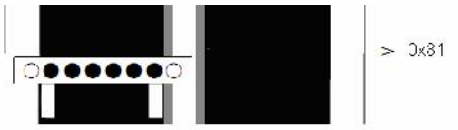
	State of course and sensor	Value when sensor is read	Hexadecimal number	Steering angle	Left motor PWM	Right motor PWM
1		00000000	0x00	0	100	100
2		00000100	0x04	5	100	100
3		00000110	0x06	10	80	69
4		00000111	0x07	15	50	40
5		00000011	0x03	25	30	21
6		00100000	0x20	-5	100	100
7		01100000	0x60	-10	69	80
8		11100000	0xe0	-15	40	50
9		11000000	0xc0	-25	21	30

Bảng 3-2. Các trạng thái led gặp trên đường đua và góc cua tương ứng

Ưu điểm của cách chạy này là: xe chạy qua các đường cong với bán kính cố định rất nhanh giống như đang chạy trên đường thẳng khi xe đã vào được đường cong. Thực tế thì trong các cuộc thi MCR chỉ có hai loại đường cong với hai bán kính khác nhau. Nên ta chỉ cần chọn hai

góc bẻ lái hợp lý ứng với hai độ lệch hướng khác nhau để qua 2 loại đường cong sao cho hai độ lêch hướng được chọn là hai độ lệch hướng trung bình – cụ thể là độ lệch hướng ứng với biến *Vitri* = +- 3 và *Vitri* = +- 4 (xem hình 5.9 ở trên).

Tại mỗi khúc chuyển giao giữa đường thẳng sang đường cong và giữa đường cong sang đường cong hướng khác ta tắt 2 động cơ phát động để toàn bộ **nguồn cung cấp** cho rc servo bẻ lái (thực tế chỉ trong 20 – 40 ms) để xe có thể chạy ở vận tốc ổn định cao hơn bình thường. Và khi phát hiện xe lệch ra khỏi phương chính giữa đường đua quá xa cũng là khi mạch dò bắt được cả line giữa và line biên ta cho xe hãm tốc mạnh để tránh trường hợp xe văng ra khỏi đường đua.



Hình 3-12. Hình mạch dò bắt được hai line

- Ý tưởng trên được hiện thực trong hàm *runforwardline (int tocdo)*

- Biến *vitri* dùng để lưu lại trạng thái led trước đó, dựa vào đó ta tránh trường hợp xe bắt nhằm một trong hai line biên.

- Biến *brake_flag*, hàm *brake(int time)*, hàm *brake_timer (int time, int speed)* dùng để hiện thực ý tưởng cho xe hãm tốc mỗi khi qua đoạn chuyển giao đường thẳng thành đường cong hay đường cong thành đường cong khác hướng.

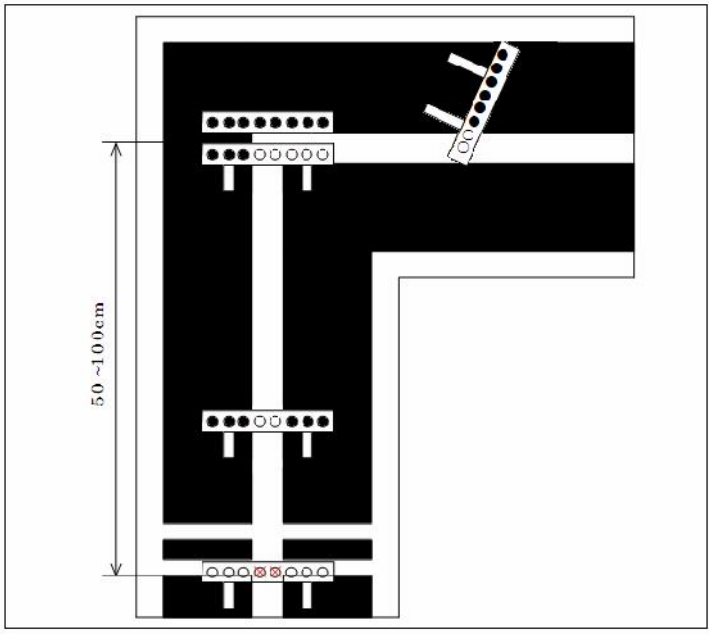
- Hàm *brake (int time)* là hàm hãm tốc bằng cách cho vận tốc hai bánh bằng 0 (*speed (0,0)*) mà vẫn điều khiển bẻ lái servo cho xe hướng theo line tương tự như hàm *runforwardline ( )*.

- Hàm *brake_timer (int time, int speed)* tương tự như hàm *brake (int time)* nhưng ta có thể chỉnh vận tốc hai bánh bằng thông số *speed* truyền vào hàm. Hàm dùng để hãm tốc mạnh.

- Xem cách hiện thực các hàm nêu trên trong sourecode đính kèm.

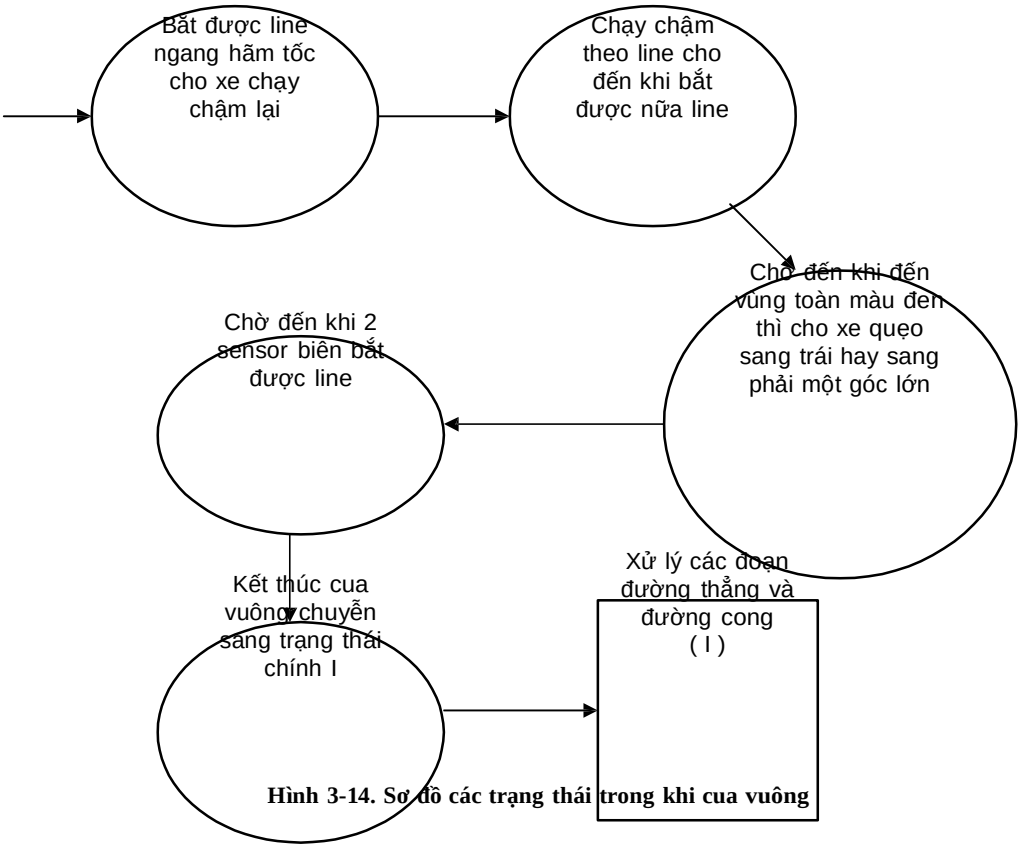
3.4. Sơ đồ trạng thái khi qua đoạn đường cua vuông

- Thứ tự các sự kiện khi qua đoạn cua vuông:



Hình 3-13. Các trạng thái trong khi cua vuông

- Sơ đồ trạng thái:



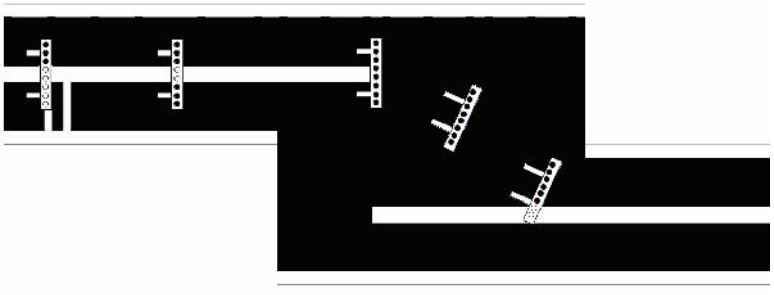
Hình 3-14. Sơ đồ các trạng thái trong khi cua vuông

- Cho xe chạy đến khi toàn bộ sensor đều nhận được màu đen rồi mới bẻ cua giúp cho xe tránh được lỗi quẹo sớm ngay khi gặp hai vạch trắng báo hiệu.

- Tham khảo giải thuật trong hàm *int turn90(int tocdo)*

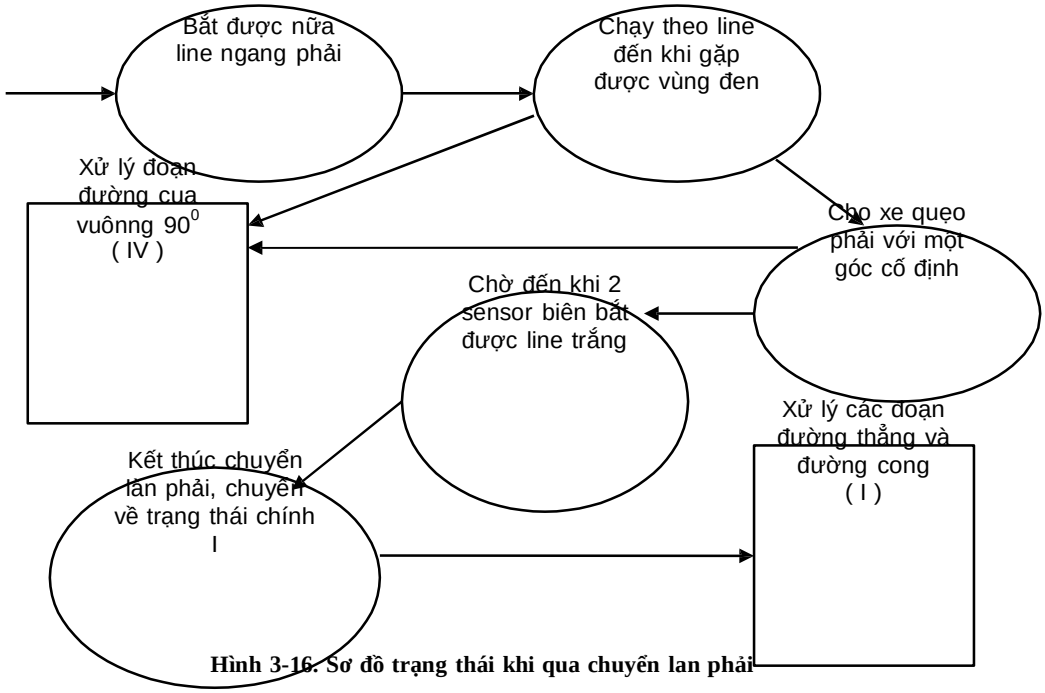
3.5. Sơ đồ trạng thái khi qua chuyển làn phải

- Thứ tự các sự kiện khi qua chuyển làn phải:



Hình 3-15. Các trạng thái khi chuyển làn phải

- Sơ đồ trạng thái:



Hình 3-16. Sơ đồ trạng thái khi qua chuyển lan phải

- Tham khảo giải thuật trong hàm int changewayright(int tocdo)

3.6. Sơ đồ trạng thái khi qua chuyển làn trái

- Tương tự như chuyển làn phải.

3.7. Hàm *test ( )* dùng để test các bộ phận xe

- Hiện thực trong hàm *void test()*

- Case 0: (chọn swicth trên board vi điều khiển) ấn bất kỳ nút nhấn nào trong 4 nút trên board vi điều khiển thì xe tự động cập nhật adc của màu trắng. Sau đó ta quét mạch dò xem các led hiển thị xem xe có nhận được màu đen không nếu không ta có thể chỉnh lại mức tính adc compare trong hàm *update_vcompare()*.

- Case 1: dùng để test 2 động cơ bánh, nhấn K0 động cơ trái quay tiến, nhấn K1 động cơ trái quay lùi, nhấn K2 động cơ phải quay tiến, nhấn K3 động cơ phải quay lùi. Nếu các động cơ không chạy theo đúng như vậy thì tốt nhất nên đảo các zack cắm nếu đảo lộn động cơ trái và phải, hay đảo dây đỏ đen của zack cắm nếu động cơ không quay đúng chiều nêu trên; không nên chỉnh sửa chương trình.

- Case 2: dùng để test động cơ servo, nếu nhấn K0 hoặc K1 servo bẻ sang phải, nếu nhấn K2 hoặc K3 servo bẻ sang trái. Nếu servo không bẻ đúng chiều như trên thì nên chỉnh sửa lại các tính góc trong hàm *handle ()*. Cụ thể là sửa dấu cộng thành trừ hoặc ngược lại. Chú ý có khi hai servo cùng loại cùng nhãn hiệu nhưng lại ngược chiều quay là chuyện bình thường.