The Freescale Cup Racing Car Reference Design Reference Manual

by: **Cuautli Padilla**
Marco Trujillo
Rodrigo Mendoza

Reviewed by: **Francisco Ramírez**

TFCMPC5604BRM
Rev 1.0
8/2011

How to Reach Us:
Home Page:
www.freescale.com
Web Support:
http://www.freescale.com/support
USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or
+1-480-768-2130
www.freescale.com/support
Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support
Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064
Japan
0120 191014 or
+81 3 5437 9125
support.japan@freescale.com
Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com
For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
+1-800 441-2447 or
+1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Contents

Chapter 1
Introduction

1.1 Software development block diagram

The diagram below explains how the software works in a graphical way.

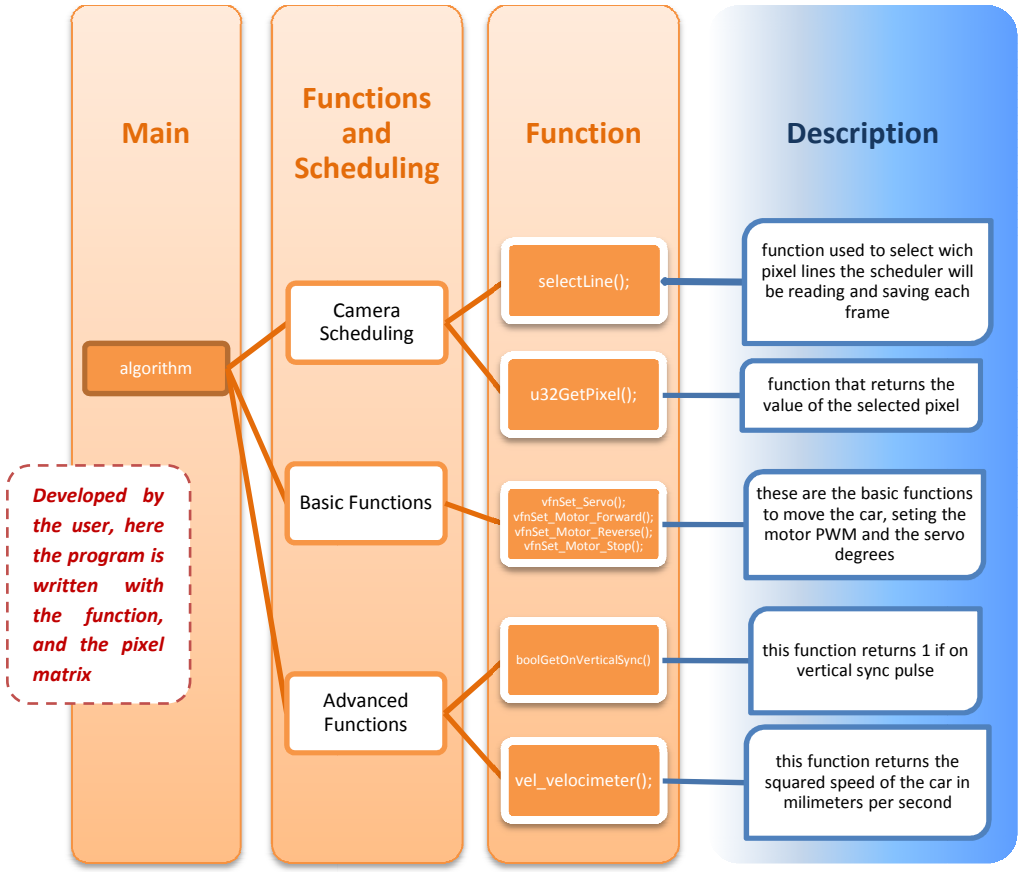


Figure 1. Reference Code Architecture

The software was developed so that the user won't have to deal with the lower level functions. User only has to add the functions to make his algorithm in the main statement. The code was thought in a way that it would be simple and compact.

In the rest of the document each of the functions will be explained as well as all the hardware implementations, so at the end, the reader would be able to understand camera signals, motor control, speed feedback and line following theory. The objective is that the reader will be able to make his own line following car using Freescale MPC5604B.

1.2 Basic Functions

These are the basic functions required to move the car, to set the motor PWM and its direction, and also to set the servo position.

Actually these functions are calibrated for the car, this means that the servo movement is limited so the wheels won't collide with the chassis, but since each car and servo are manually assembled then these calibration values may vary a little.

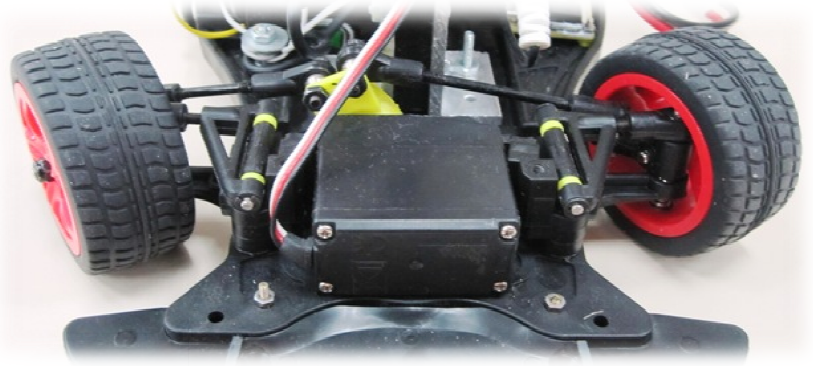


Figure 2. Servo motor disposal (to be moved with the PWM signal)

1.3 Advanced functions

These functions could be used depending on the user needs;

- *boolGetOnVerticalSync()* is a function that returns 1 if the actual state of the scheduler is idle, and it is on a vertical sync pulse. This function is useful when in your algorithm you need to perform a task of high priority that shouldn't be interrupted.

- *vel_velocimeter()*, is a function that will return the actual speed of the car, it is useful and sometimes vital for achieving optimal performance on the race, this function will return the actual speed of the car in millimeters per second. This in conjunction with the set motor function can be used to control the speed of the car; user could use a PID control or switch control to vary speed of the car. This function can only be used while on vertical sync, so the use of *boolGetOnVerticalSync()* function is necessary.

Chapter 2
HW Architecture

2.1 DC Motor Control

2.1.1 How DC motors work?

The DC motor is a machine that transforms electric energy into mechanical energy in form of rotation. Its movement is produced by the physical behavior of electromagnetism. DC motors have inductors inside, which produce the magnetic field used to generate movement. But how does this magnetic field changes if DC current is being used?

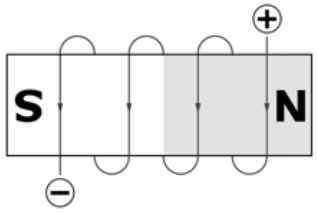


Figure 3. Electromagnet and coil

Figure 3 shows an example of an electromagnet, which is a piece of iron wrapped with a wire coil that has voltage applied in its terminals. If two fixed magnets are added in both sides of this electromagnet, the repulsive and attractive forces will produce a torque (Figure 4).

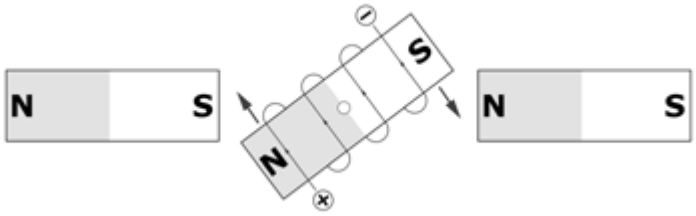


Figure 4. Torque dynamics

Then, there are two problems to solve: feeding the current to the rotating electromagnet without the wires getting twisted, and changing the direction of the current at the appropriate time. Both of these problems are solved using two devices: a split-ring commutator, and a pair of brushes (Figure 5).

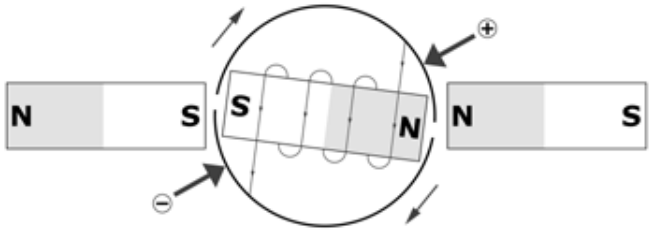


Figure 5. Brushed DC motor concept

As it can be seen, the commutator has two segments which are connected to each terminal of the electromagnet, besides the two arrows are the brushes which apply electric current to the rotary electromagnet. In real DC motors, three slots can be found instead of two slots and two brushes (Figure 6).

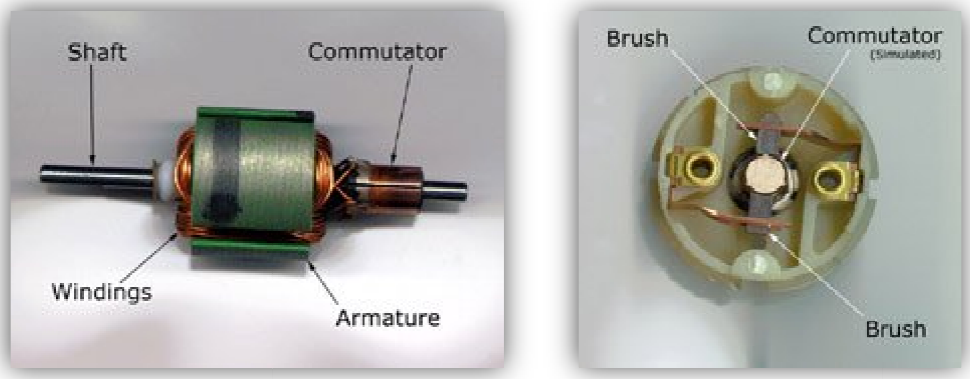


Figure 6. Brushed DC motor

This way, as the electromagnet is moving its polarity is changing and the shaft may keep rotating. Even if it is simple and sounds that it will work great there are some issues which make these motors energy inefficient and mechanically unstable, the principal problem is due to the timing between each polarity inversion.

Since polarity in the electromagnet is changed mechanically, at some velocities polarity is changing too soon, which result in reverse impulses and sometimes in changing too late, generating instantaneous “stops” in rotation. Whatever the case, these issues produce current peaks and mechanical instability.

2.1.2 How a DC motor can be controlled?

DC motors have only two terminals. If you apply a voltage to these terminals the motor will run, if you invert the terminals position the motor will change its direction. If the motor is running and you suddenly disconnect both terminals the motor will keep rotating but slowing down until stopping. Finally if the motor is running and you suddenly short-circuit both terminals the motor will stop.

So there is not a third wire to control a DC motor, but knowing the previous behaviors it can be designed a way to control it, and the solution is an H-bridge (Figure 7).

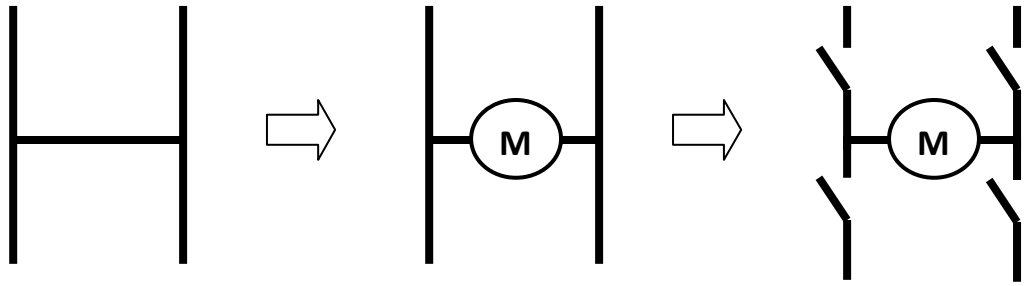


Figure 7. H-bridge concept

At the end of figure 7, it can be seen there are four gates and a motor connected between them. This is the basic concept of an H-bridge, where the four gates represent four transistors. By manipulating these gates and connecting the upper and lower terminals to a voltage supply, you can control the motor in all the behaviors listed before (figure 8).

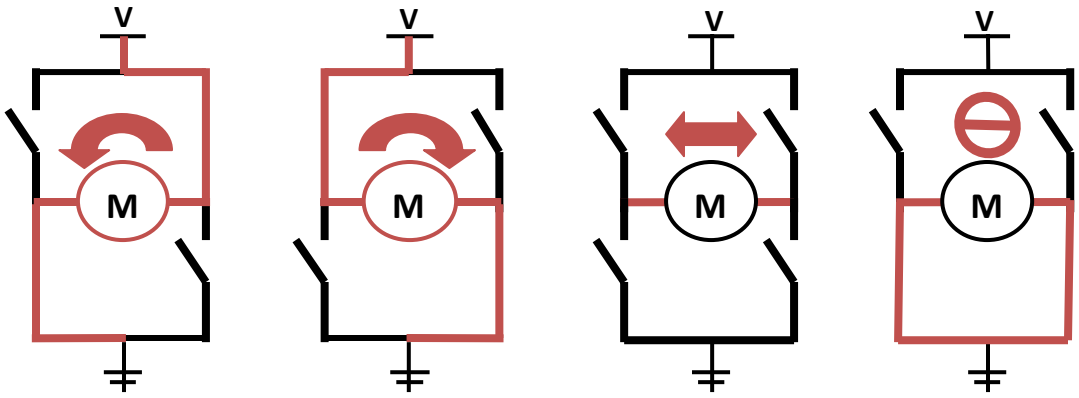


Figure 8. H-bridge position/options

2.1.3 Importance of an H-bridge

H-bridges seem to be very simple but there are important conditions users may consider before choosing one or making their own:

- ***Nominal current:*** the maximum current the H-bridge can resist at a constant supply.
- ***Peak current:*** the maximum current the H-bridge can hold for short instances.
- ***Internal resistance:*** the approximate resistance that the H-bridge adds to the circuit.

There are some other characteristics like operating voltage, operating current, operating frequency and some other ratings and parameters which depending on the application, may be important; for the case of this reference design, it were not considered.

So far, with previous paragraphs, the reader can understand how to change direction and break (stop) the motor, but, what about **speed control**?

A typical option is regulating the current flow in the motor by adding an external resistance, this method is inefficient and hard to regulate by software. A similar approach (external variable resistor) by software for the speed control is a PWM (Pulse Width Modulation) control.

This method of a PWM consist in toggling the H-bridge gates in order to limit the amount of time that the current flows through the motor. To achieve this, a digital signal with constant frequency but variable high state or "duty cycle" need to be created. Figure 9 shows some examples of PWM and duty cycle.

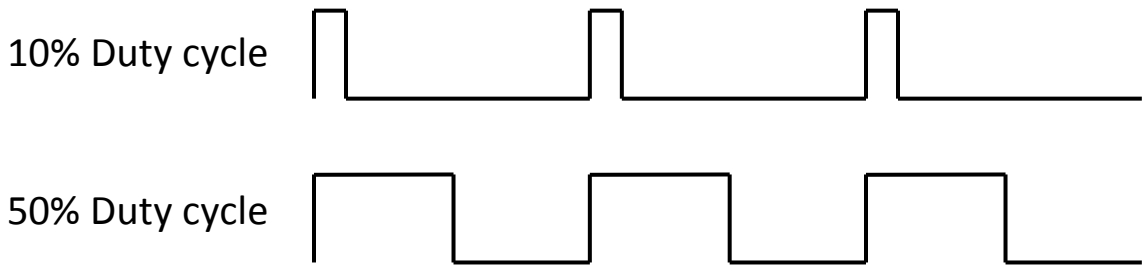


Figure 9. PWM exemplification

For this reference design, the designer should consider that the nominal current of the motor can run up to 3A and peak current can reach up to 13A. Three different H-bridges where benchmarked. Initially the Freescale H-bridge MC33921, then the ST L6203 was tested but at the end the ST

VNH2SP30-E was chosen. The following chart shows some advantages and disadvantages of them; this will allow taking a decision:

H-bridge	Advantages	Disadvantages
Freescale MC33921	Enough nominal current tolerance. Low internal resistance. Easy to use evaluation board.	Insufficient peak current tolerance.
ST L6203	Enough nominal current tolerance. Enough peak current tolerance Cheap.	High internal resistance. Quasifunctional (low $R_{DS\ ON}$ when voltage is under 12V)
ST VNH2SP30-E	Enough nominal current tolerance. Enough peak current tolerance Very low internal resistance.	Expensive.

Table 1. H-bridges benchmark

The VNH2SP30-E have the more appropriate characteristics for the purpose of this reference design. Unless it is a little expensive, it worked pretty well and there is no need of making a PCB because there is a nice board developed by Pololu.

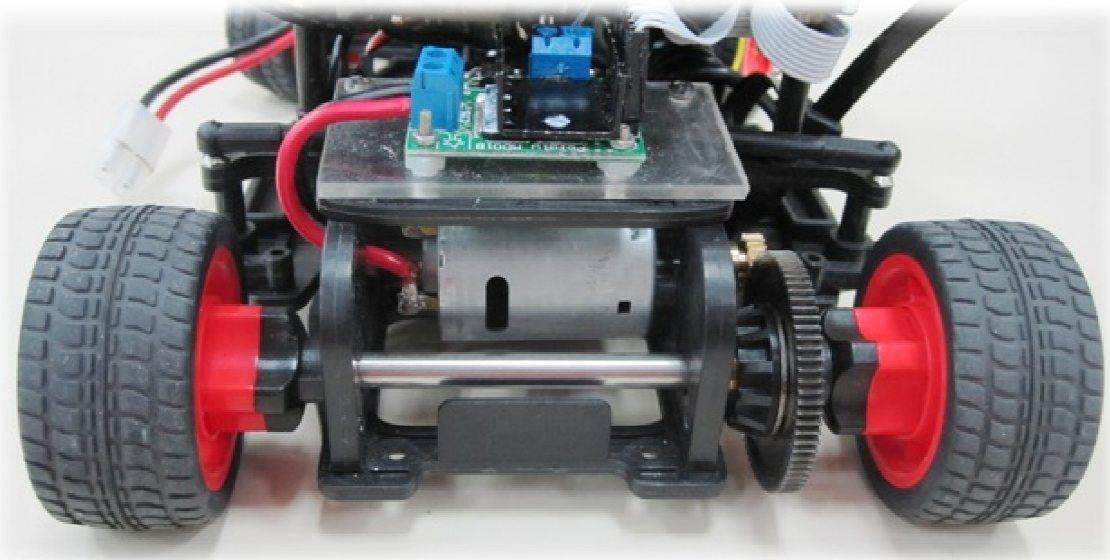


Figure 11. DC Motor and H-bridge disposal

2.1.4 Sample code for the DC motor and the H-bridge

The following code was developed for the ST VNH2SP30-E H-bridge. According to its data sheet, pins $DIAG_A/EN_A$ and $DIAG_B/EN_B$ must be connected to an external pull up resistor in order to enable the whole bridge. For controlling the bridge states, two GPIO's pins of the microcontroller were connected to the logic inputs IN_A and IN_B as shown in Table 2. Finally for the speed control connect a PWM pin of the microcontroller to the PWM input of the H-bridge. For a better reference, check Annex A4.

IN_A	IN_B	$DIAG_A/EN_A$	$DIAG_B/EN_B$	OUT_A	OUT_B	CS	Operating mode
1	1	1	1	H	H	High Imp.	Brake to V_{CC}
1	0	1	1	H	L	$I_{SENSE}=I_{OUT}/K$	Clockwise (CW)
0	1	1	1	L	H	$I_{SENSE}=I_{OUT}/K$	Counterclockwise (CCW)
0	0	1	1	L	L	High Imp.	Brake to GND

Table 2. Basic H-bridge (ST VNH2SP30-E) specifications

User of this sample code shall define which pins are going to be used as GPIO and which ones as PWM. User can do this directly in the Driver_Motor.c at the top of the document in the definition part:

```
#define MOTOR_MCB_CHANNEL       16                  /*Counter for PWM*/
#define MOTOR_IN_2_PIN          GPIO_PIN_E4         /* PE4 connected to IN_A*/
#define MOTOR_IN_1_PIN          GPIO_PIN_E5         /* PE5 connected to IN_B */
#define MOTOR_EN                18                  /*Channel for PWM*/
#define MOTOR_EN_PCR            PCR_EMIOS_0_18      /*PE2 connected to PWM*/
```

Once the user knows which pins will be used is time to initialize the whole Quoriva MCU and try a simple code to move the motor forward and a controlled speed. Let's review main.c file:

```
#include "MPC5604B_M27V.h"             // For register definition
#include "Setup.h"                    // Principal initialization
#include "Driver_SIU.h"               // SIU driver
#include "Driver_EMIOS.h"             // EMIOS driver
#include "Driver_MPC5604B.h"          // Useful defines (most for PCR)
#include "Driver_Motor.h"             // Motor Driver
#include "main.h"                     // Main header

void main (void) {

      vfnInit_All();                  //Necessary initialization of modes,
                                      //clocks, watchdog
      vfnSetup_Emios_0();             //Initialization of the EMIOS module
      vfnInit_Motor();                //Initialize ports for Motor control
      vfnInit_Emios_0();              //Allow EMIOS counter to start working

      while(TRUE)
      {
            vfnSet_Motor_Forward(20); //Run the motor forward at 20% of
                                      //power
      }
}
```

For more information about the Driver_Motor.c look at the Freescale application note AN4251.

2.2 Servo Motor Control

2.2.1 How a servo works?

A servo motor is capable of staying in a fixed position inside its operating range; it is composed by a DC motor, a gear array and a control circuit. They are very often used in robotics and RC systems. As a difference with a DC motor, the servo motor has a third wire to control it. The way of controlling it, is via a PWM signal as shown in figure 11. The AN4251 shows more information on how this motor works.

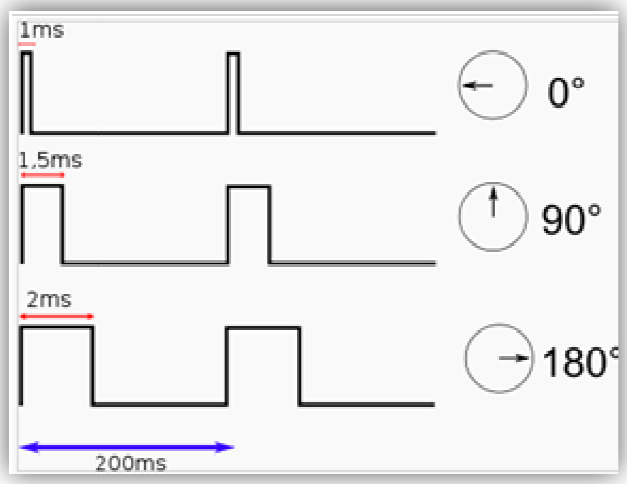


Figure 11. values/specifications to move the stepper motor.

2.2.2 Example code

All the pins to be used on the MCU need to be defined. This definition can be done at the Driver_Servo.c in the “definitions” part located at the top of the document.

```
#define MOTORS_MCB_CHANNEL      16                  /*Counter for PWM*/
#define SERVO_CTRL              22                  //Emios channel 22 for
                                                    //WM
#define SERVO_CTRL_PCR          PCR_EMIOS_0_22      //PE6 connected to servo*/
```

Then, user must initialize these pins and the whole Quoriva MCU and try a simple code to move the servo in a specific position. Change the values of the following definitions in the Driver_Servo.c

```
#define SERVO_MIN_US            1290        /* Min val in microseconds 2030*/
#define SERVO_MAX_US            2090        /* Max val in microseconds */
```

Then run the following code at main.c until you find the limits of the Servo.

```
#include "MPC5604B_M27V.h"              // For register definition
#include "Setup.h"                      // Principal initialization
#include "Driver_SIU.h"                 // SIU driver
#include "Driver_EMIOS.h"               // EMIOS driver
#include "Driver_MPC5604B.h"            // Useful defines (most for PCR)
#include "Driver_Servo.h"               // Motor Driver
#include "main.h"                       // Main header

void main (void) {

      vfnInit_All();                    //Necessary initialization of modes, clocks,
                                        //watchdog
      vfnSetup_Emios_0();               //Initialization of the EMIOS module
      vfnInit_Servo();                  /*Initialize ports for Servo control*/
      vfnInit_Emios_0();                /*Allow EMIOS counter to start working*/

      while(TRUE)
      {
            vfnSet_Servo(50, 0, 100); /*Position the servo motor at exactly half
range*/
      }
}
```

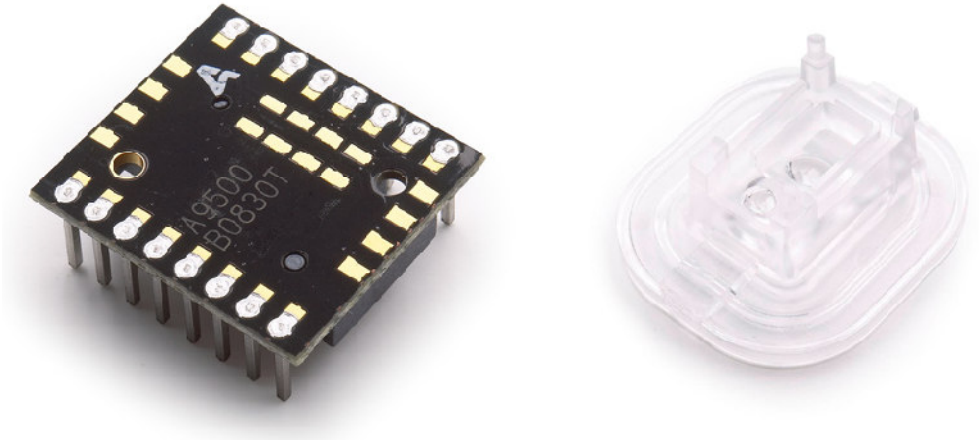
For more information about the Driver_Servo.c look at the Freescale application note AN4251.

2.3 Car speed application

2.3.1 The speedometer dilemma

In a previous application note (AN4251), it was developed a speedometer using a Hall Effect sensor and placing two tiny magnets in the car's back wheel. The problem with this technique consists in waiting for a whole revolution of the wheel to capture the speed; in addition what you get is the wheel speed and not the car speed.

One of the best achievements of this reference design, is the speedometer. To measure the speed, now it was implemented a mouse optical sensor ADNS-9500. This allows capturing the speed of the car and returns instantaneous velocity whenever the code request for it. Figure 12 shows a picture of the sensor and its lens.



a. Optical Mouse Sensor *b. Protective Lens*

Figure 12

The sensor works with a SPI (Serial Peripheral Interface) but communication is not its main complication. Since this sensor is not developed for this kind of applications, some modifications have to be done to adapt it for TFC purpose, starting by developing a board with all its hardware specifications. You can find a picture of the sensor schematic and layout at the Annex 1. If you want to modify or recreate the PCB that was developed, **you can find the layout and schematic files here.**

According to hardware specifications of the optic sensor it was necessary to place the lens at 2.4 mm from the floor for an optimal response, nevertheless different approaches were tried; from less than 1mm and up to 5mm and the sensor still worked.

To make this sensor run, it is necessary to follow a specific power up sequence and load a 3Kbytes ASCII file before asking for any measurement. Its initialization is quite complex, but this is solved in the driver_velocimeter.c file. In case you want to make your own, the best recommendation is to read the ADNS-9500 datasheet.

2.3.2 Sample code

The following example is an algorithm that moves the motor forward in case any speed is detected. Try pushing your car backwards and it will run forward until stopping. This portion shall be written at main.c

```
#include "MPC5604B_M27V.h"              // For register definition
#include "Setup.h"                      // Principal initialization
#include "Driver_SIU.h"                 // SIU driver
#include "Driver_EMIOS.h"               // EMIOS driver
#include "Driver_MPC5604B.h"            // Useful defines (most for PCR)
#include "Driver_Motor.h"               // Motor Driver
#include "driver_velocimeter.h"         // Velocimeter driver
#include "main.h"                       // Main header

void main (void) {

      vfnInit_All();            //Necessary initialization of modes, clocks,
                                //watchdog
      vfnSetup_Emios_0();       /*Initialization of the EMIOS module*/
      vfnInit_Motor();          /*Initialize ports for Motor control*/
      vfnInit_Emios_0();        /*Allow EMIOS counter to start working*/
      init_velocimeter();       /*Initialize the mouse optic sensor*/

      int velocity=0;

      while(TRUE)
      {
            velocity = vel_velocimeter();    //Capture the squared value of
                                             //velocity in mm/s
            if (velocity > 100*100)          //Condition for velocity > 100
                                             //                        mm/s
            {
                  vfnSet_Motor_Forward(80); //Run forward at 80% of power
            }
            else
            {
                  vfnSet_Motor_Stop();      /*Stop motor*/
            }
      }
}
```

2.4 RCA Camera

2.4.1 Camera Interface

Visit and read http://www.ntsc-tv.com/ to know more about a RCA camera signal or read AN4245 to understand the basics of NTSC signals.

To interface the camera to the microcontroller the specs should be read and take these points in consideration:

- NTSC output (can work also with PAL output modifying the scheduler)
- Positive voltage video output
- Video voltage output should be less than your microcontroller output
- For this application, black and white cameras and color cameras fits the application the same way since the only color that we are measuring are black and white.
- If you have a camera with a negative and positive output, please review the recommendations in AN4245 for making its output compatible with your MCU.

The RCA camera used in this reference design is a cheap B&W RCA camera (Figure 13).

Figure 13. RCA camera

The advantages of this camera are:

- Easy to find (STEREN, Part Number: CCTV-104)
- Directly connects to the MPC5604B
- No signal conditioning required
- 5.9V minimum for normal operation even if the package says 12V
- Low cost

An easy way of checking if the output of the camera suits your needs is using an oscilloscope, and checking if the signal is positive and lower than 5V.

2.4.2 Camera signal readings

2.4.2.1 Scheduling

The scheduling was developed specifically for this application. The theory of operation will be explained in the next lines.

First of all the MPC5604B ADC is always busy, searching for the horizontal sync pulse. When it detects sync, it interrupts and saves the value of the actual line. The actual camera used for this application has 242 lines before each vertical sync pulse as shown in figure 14. And normally every RCA NTSC camera has this quantity of lines. So when it finds sync it interrupts and saves the value of the line until its value is 242 or also if it finds a vertical sync pulse that is the end of the frame. When this occurs the scheduler enters an idle state for a short period of time before the next frame starts.

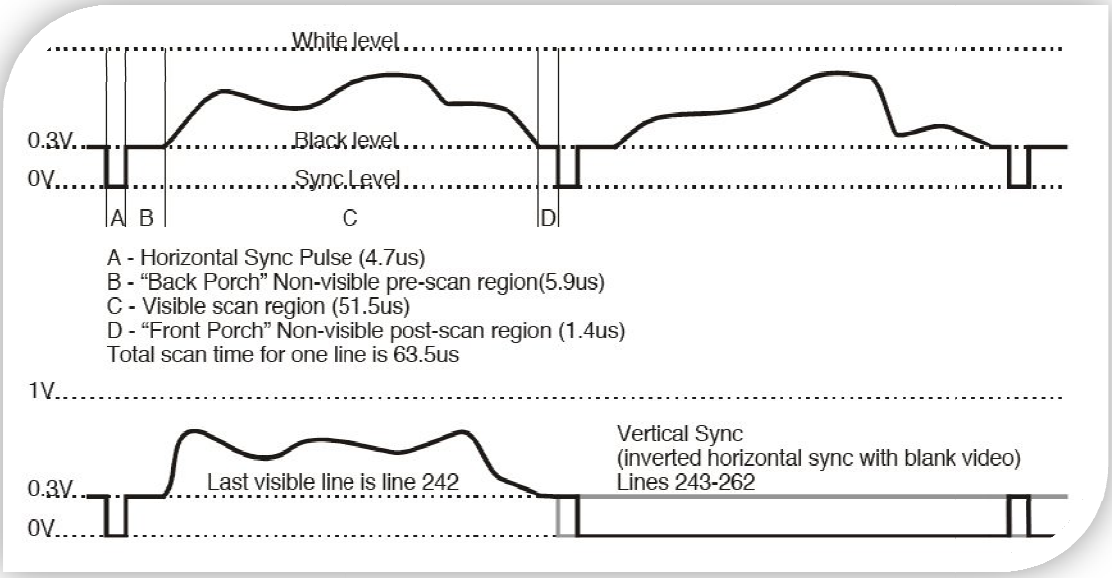


Figure 14. RCA NTSC signal

With the function *selectLine()* the user can select which lines will be saved, every other line will be dropped. When in the interruption, the scheduler finds that the line you asked for with the *selectLine()* function, is the actual line that is being read, then it starts an ADC capture routine, which will save the entire line in a matrix that you can access with the function *u32GetPixel()* – in this case, the MPC5604B is taking 44 ADC samples of a picture line. With this function in main file you will only have to make your algorithm to find the position of the line, and the scheduler will do all the job for acquiring the selected lines of the frame and will be automatically refreshed each frame.

The time that the processor is busy is proportional to the number of lines selected; this means that if the entire 242 lines are selected, the MPC5604B will be busy almost the entire time acquiring the frame data. In other words, user should think of an algorithm that will use fewer lines to have more time to process the data and have a efficient line following algorithm.

2.4.2.2 Line Detection

Once the pixel data matrix of the camera is taken, there are many algorithm implementations that you can apply to the data to detect the line, and be able to locate the position of it, its direction, and curve intensity.

The simplest algorithm that could be implemented is searching for the minimum value of an entire line of pixels; this will return the position of the line. For example, if you have a matrix of 4 lines with 10 pixels, in each line as shown in Table 3:

1	2	3	4	5	6	7	8	9	10	Line Pos
190	186	185	189	191	189	101	189	191	186	X
186	189	188	189	191	110	191	187	186	187	X
188	188	189	196	109	185	187	193	192	191	X
188	190	199	197	123	189	199	193	191	190	X

Table 3. 4 lines matrix array

After implementing the algorithm, user will get table 4:

1	2	3	4	5	6	7	8	9	10	Line Pos
0	0	0	0	0	0	1	0	0	0	7
0	0	0	0	0	1	0	0	0	0	6
0	0	0	0	1	0	0	0	0	0	5
0	0	0	0	1	0	0	0	0	0	5

Table 4. 4 lines de-codified matrix array

After seeing previous tables, we can understand the line the camera is actually seeing is as shown in figure 15.

Figure 15. Random view of the RCA camera

This algorithm is simple and works very well in this kind of application; the only inconvenient is when the camera is blind and it is looking only towards a white portion of the track, the algorithm will still find the lowest value even if the entire image is on blank. This can be fixed adding a threshold, for example if the lowest value is over a established threshold then ignore that value because it is too high and it's not a valid black line.

Other algorithms could be implemented also to correct this mistake; like taking the derivative of the data. Taking the derivative would give a new data matrix that will contain the peaks where the edge of the line was detected, having this peaks will help to determine the width of the line and the position, with this information it's easier to determine if the line is valid. If the width of the line is too high, then it's because the line was invalid. There are many algorithms that could be implemented for the line detection, but as an advice keep the algorithm simple, it's not necessary to apply really difficult mathematical functions to find the line.

2.4.2.3 Line Following

After reviewed the previous concepts; basic line following consist on turning the servo toward the direction of the line, and increasing servo degrees as the curve increases its intensity, also slowing down speed in curves and increasing speed on straight pads will make your car go even faster.

Here is a simple pseudo code example of how would a line following algorithm would look like. This pseudo code takes one line of the entire frame and follows the minimum pixel value. On The Freescale Cup track, the minimum value would be the black that is the line and the maximum would be the white, and then it follows the black line with a 75% motor power.

```
//////////////////////////Code Beginning////////////////////////////////
/////this function finds the position of the minimum pixel value.//////
//////////////////////////////////////////////////////////////////////////

Function_made_by_user_to_find_minimum_value()
{
              loop( until the end of the pixel line)
              {
                        value=u32GetPixel(x,counter)
                       if (value is the min of the entire line)
                       {
                                  //save value position of the matrix
                       }
               }
              return (saved value)
}
selectLine(x);                    //Each frame, the scheduler will save line x
                                  //selected in the pixel matrix
main()
{
       Int servodirection;
       Servodirection = Function_made_by_user_to_find_minimum_value()
       vfnSet_Servo(servodirection,4,49);
                                  //4,49 are default values for the
                                  //RCA NTSC Camera
                                  // this function sets the position
                                  //of the servo
       vfnSet_Motor_Forward(75); //motor forward with 75% dutty cycle
}
//////////////////////////////////End of code////////////////////////////////
```

When doing line following, following need to be considered:

- Keeping a constant PWM won't make your car keep a constant speed the entire track
- Speed control can be really easy to achieve once you get the proposed speedometer working.
- Applications don't have to completely follow the track as it is, turning inside the curve and going straight some small turns or smooth variations on the track can save a lot of time.
- Using the proposed camera scheduler will save days of software design.
- Generally after a straight line if the car is going really fast, the reverse function will allow slowing down the car and it will turn without slipping.
- For test purpose, using a remote turn off device for the car can save you time, as normally every trial has the possibility to go off the track and collide and it could be damaged, or simply it can be stopped for any other intention required.

2.5 Assembly

2.5.1 Power/Battery

The battery that is being used is the one showed in Figure 16.

Figure 16. 7.2V battery pack

This battery has a rated voltage of 7.2V and 2200mAh, following considerations shall be taken care:

- Camera working voltage should be lower than your battery voltage for proper camera operation.
- Remember to respect all components voltage specs.
- Motor drains lots of amps; the more you stress the motor, the more current it will drain then the voltage will drop down.

2.5.2 Capacitive Integration

You should always remember to use capacitors where they are needed. Using pF capacitors where there are high frequencies noises such as pads near clocks will help. Also using a big uF capacitor near the motors will help to reduce voltage drop when high current are needed.

2.5.3 Camera mounting

These are recommendations when placing a camera on the car:

- Experiment with different camera heights, many times the camera is better at a higher place since the black line will have less deformation (Figure 17).
- The further away the camera can see, the more response time the car will have to lower it speed before making a turn.
- Use materials resistant to impact, like aluminum, when building the holding device for the camera, so it will resist if the car collides when doing test runs as shown on Figure 18.

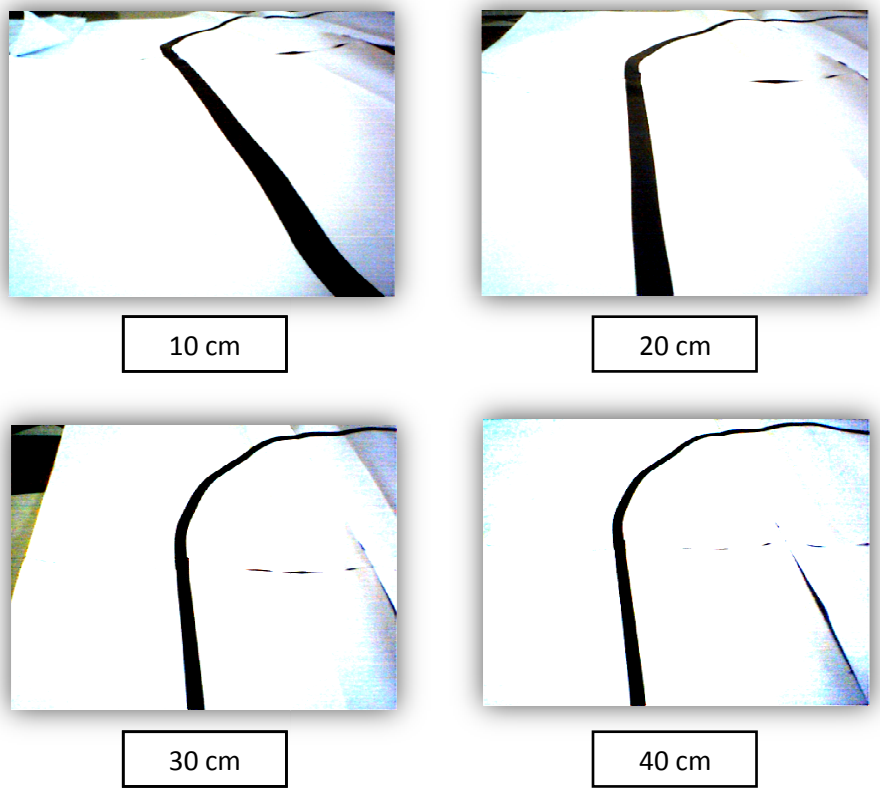


Figure 17. Different views of the camera at different heights measured from the floor to the camera lens.

As you can see it is easier to distinguish the curve pattern when the camera is further away from the ground, but also the car stability is compromised when the camera is too tall. So the user must find equilibrium when positioning the camera.

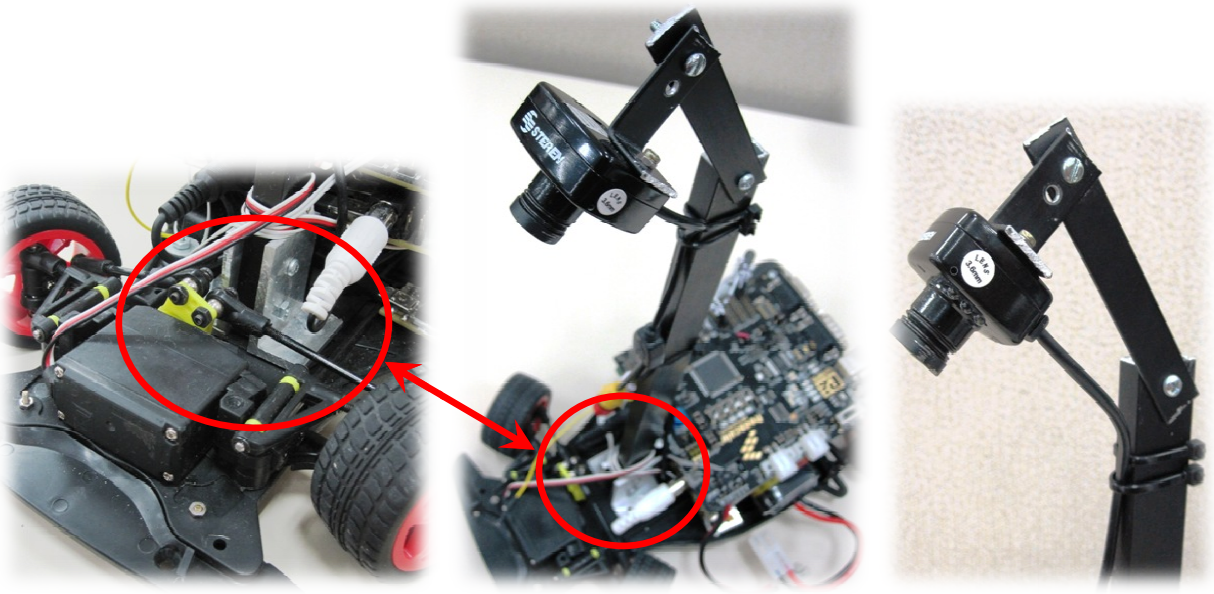


Figure 18. RCA camera mounting

Annex A
A1. Speedometer Schematics

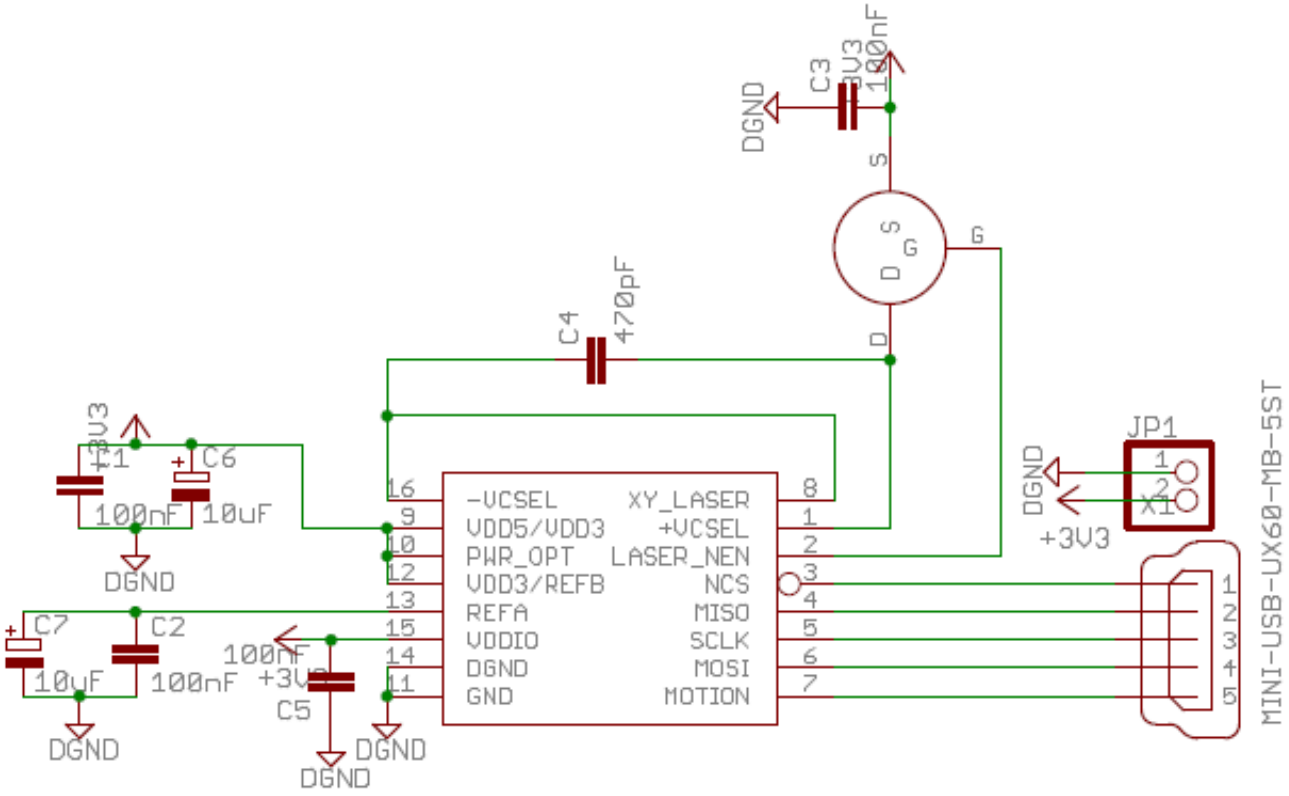


Figure 19. Schematic for the speedometer sensor

A2. Speedometer Layout

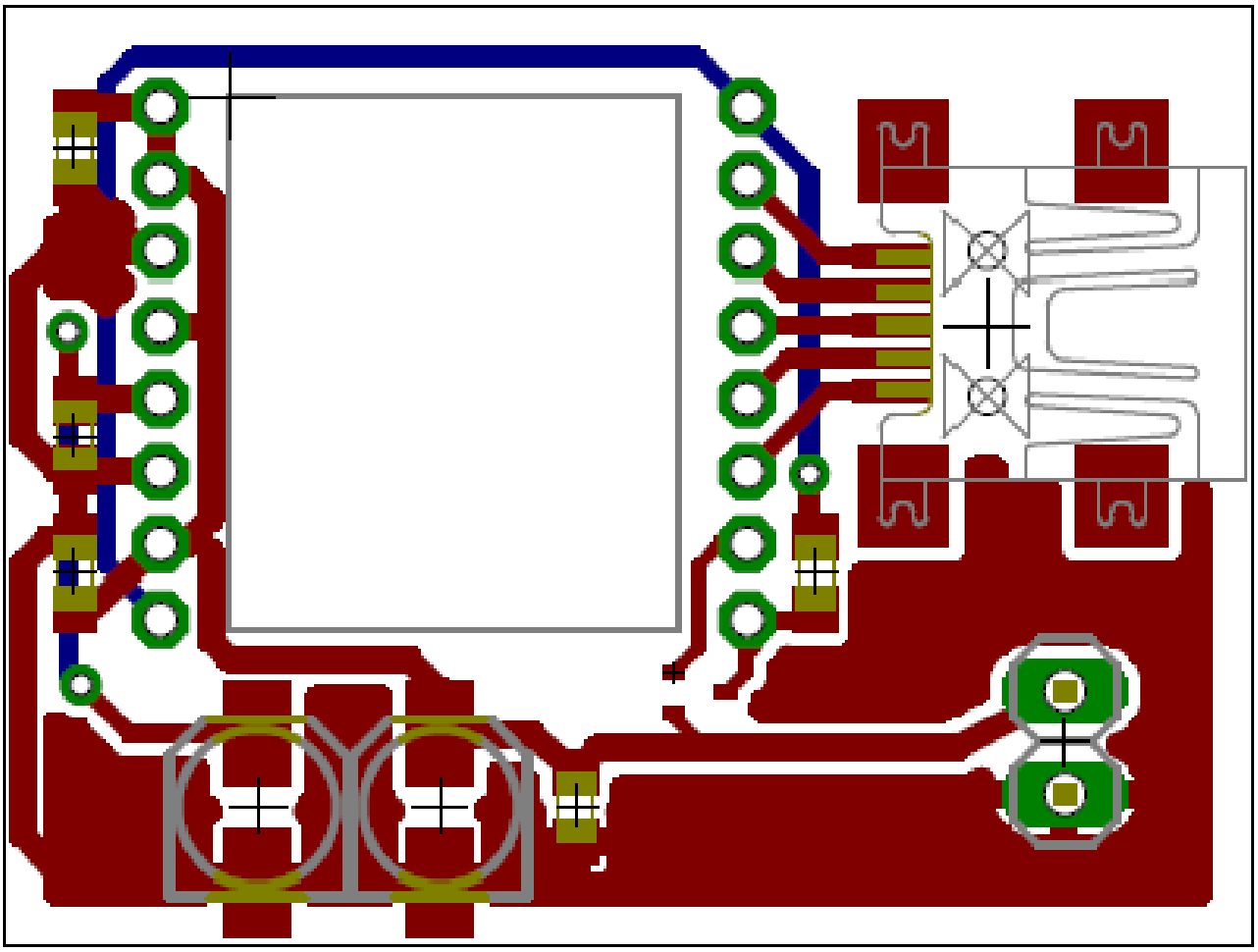


Figure 20. PCB Layout for the speedometer sensor,
in red is the top face and the bottom is in blue.

A3. Code – pin out correlations

This table is intended for a better reference on the code provided with this document. It explains the connections implemented on the code relating the TRK-MPC5604B board with each peripheral used and explained in the document.

		Device:	MPC5604B
DC Motor	**TRK-MPC5604B**	**I/O**	**Functionality**
IN1	PE4	input	GPIO
IN2	PE5	input	GPIO
PWM	PE2	input	PWM

Servo Motor	**TRK-MPC5604B**	**I/O**	**Functionality**
IN	PE6	input	PWM

Velocimeter ADNS-9500	**TRK-MPC5604B**	**I/O**	**Functionality**
SOUT	PA13	input	SPI
SIN	PA12	output	SPI
SCK	PA14	input	SPI
PCS	PA15	input	SPI

RCA Camera	**TRK-MPC5604B**	**I/O**	**Functionality**
NTSC signal	PB5	output	ADC

Table 5. TRK-MPC5604B pinout correlations

A4. ST VNH2SP30-E Building Blocks

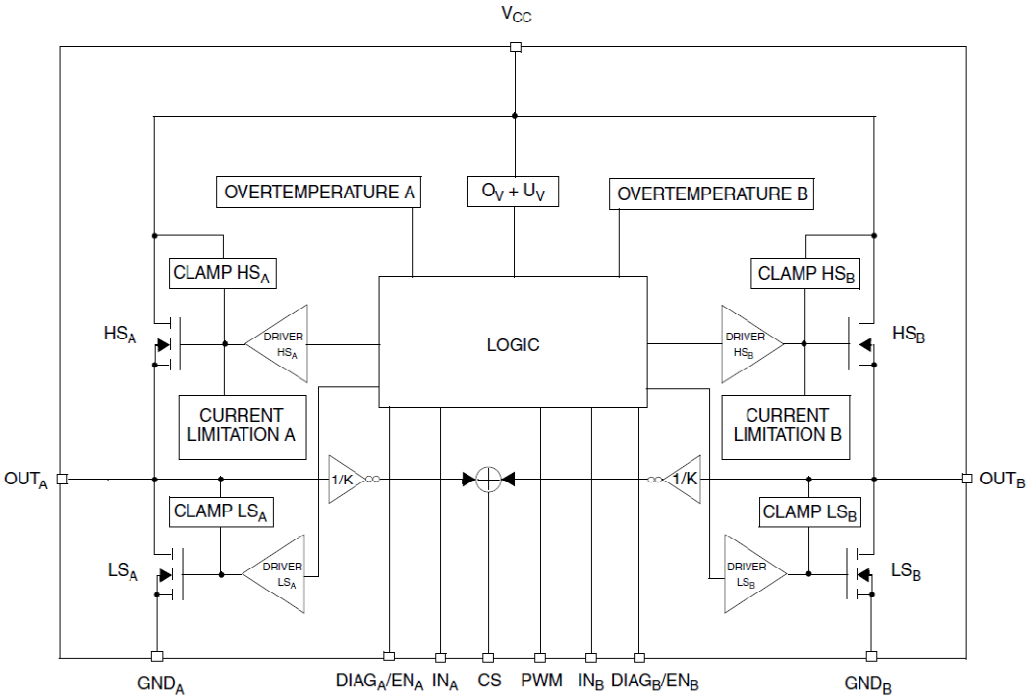


Figure 21. Circuit Diagram of the ST VNH2SP30-E