**Contents**

Introduction. 1

Building Stage. 2

Testing. 3

Coding. 7

Team Input 7

Conclusion. 8

# **Introduction**

The Renesas MCU Car Rally is an event held in Germany in which different Universities go head to head in order to make a small hand-built car navigate through a track by itself. Our Group consists of 5 members, Daniel McCluskey, Thomas Hopkins, Zack Raeburn, Connor Liggins and Morgan James. During the course of the project, every team member will contribute towards the completion of the car, including both the building of the car and the coding of the car. This project will help us in the future since the learning experience opens up many opportunities for the future and expands our knowledge of Software and Hardware integration.

# **Building Stage overview**

The first step of building the car was to assign different tasks to each of the group members, to do this we used a piece of software called Trello, we inputted the different tasks that would need to be completed such as “Soldering the Motor Board” or “Constructing the Wheels”. Trello allowed us to assign specific tasks to different group members and then easily tick off what we had completed so the other group members were always up to date on how close we were to the completion of the car.

The construction of the car was the hardest part of the project since each step introduced new faults and problems to the mix. One of the problems that we encountered was the “Bridging” of components during the soldering of the Motor Board, this damaged one of the components meaning we had to replace the affected one, this hindered progress since we were unable to finish the construction of the car until the new component arrived. Another problem that we faced was that we did not always have access to the University workshop meaning we could not progress further with the project since we were unable to use any of the tools that we required, we were able to solve this by changing our schedule to more suit the availability of the University Workshop. Another problem that we faced was the wires that stretch across the car would sometimes get caught under the wheels and damage the car, we had a temporary solution for this problem which was to use some more wire and just quickly tie the loose wire down. We eventually got around to permanently solving this problem by using cable ties to more securely hold the loose wires in place.

The last stage of building the car was to upload the default program on to the microcontroller so that we could start programming the car. We came across a problem where we were unable to get the correct wire to upload the program to the microcontroller, this hindered progress for a short while but was solved by purchasing the wire from a local vendor. We were then able to see the car in action and start the vigorous testing process. We all took a copy of the program so that we were all able to start understanding the program so we could all offer input to the coding process.

The building of the car was a very interesting process and has definitely expanded my knowledge of electronic components and Software/Hardware integration. If we were to do the project again then we would most likely take the same approach except have multiple people working on the same tasks to provide assistance to each other if one person were struggling. The testing leading up to the event will take a long time since the default program is only the bare essentials of what we need.

# 

# **BUILD LOG**

## **The Section Below shows each step of the process of building the car.**

## 

## **Universal Plate (Fashioning of Plate Parts)+ Drilling and Preparing Holes for Flathead Screws+Assembling the Sensor Arms and Front Wheel Support Plate**

|  |  |
| --- | --- |
|  | To complete this part of the build the sensor arms had to be assembled by cutting the long plastic strips into 2 pieces of the correct size and then bolting the different pieces together.  The universal plate then had to be drawn upon as to where it would be later cut out, a hacksaw was then used to cut the board along with a vice and a rule to have all of the correct pieces for future parts of the car. After filing the plates so they are smooth a hand drill with the correct size head for the toggle switches was used to drill through the board where the toggle switches went, the drill bit was replaced with one adjusted for the flat head screws and was used to bevel out the holes in the position for the screws to go. |

Written by Morgan James

## **Assembling the Tires, Wheels, and Shafts**

|  |  |
| --- | --- |
|  | The assembly of the wheels initially required the cutting and deburring of round shaft wheel hubs before a mixture of screws spring washers and nuts was used to secure the hubs in place on the inner side of the wheel. The nuts were then tightened using the nut tighteners provided in the tire sets that were used in this process.  The preparation and implementation of the shafts was a slightly trickier process as the shafts that were used to mount the front wheels had to be modified.  This modification involved sawing the front wheel shafts one millimetre from the notch and then deburring the rough edge. The shafts were then fitted with an E-ring on one end before being inserted into the bearings from the outer side and secured with another E-ring.  The bearings were then mounted onto the front wheel support plate making sure that the that the shafts did not make contact in the middle marking the completion of the shaft assembly process. |

Written by Connor Liggins

## **Mounting the Servo Motor**

|  |  |
| --- | --- |
|  | For this part of the build the red servo horns had to be removed using nippers. The white servo horn also had to be removed by way of unscrewing. A 3mm hand drill was then used to widen the holes for the screws to be able to fit in. following that the red servo horn was to be replaced on the servo and confirmed it could rate 90 degrees to either side.  The proceeding adjustments to the servo centre entailed making sure the horn was in the correct position by way of turning the horn counter clockwise as far as it would go and replacing the servo horn so it is straight.  After confirming the above servo horn was screwed in place to allow or the mounting of the servo horn on the front wheel support plate by inserting the screws and washers through the board and servo then tightening a nut to the other side so that when the servo rotates the front wheel support should also rotate. |

Written by Morgan James

## **Assembling the Gearbox**

|  |  |
| --- | --- |
| https://lh4.googleusercontent.com/MKBEAncbKSfCH9I_p66Tn2YLovc-fv7sG_V5Ubr-Qd1LUOQdBTBdNFMULCcw_zf85u7OYyN_i5sVtvGQdRp3BvG4PVYWguND65nvWSMUgKisoLPbM5GneDAQfyO1J-B5pw  https://lh5.googleusercontent.com/VQhr-gPGs8zIaOS3Ta8QTa3q2BAb981sfgup6_1WLI6MLhNE6CQvFDPXmQ_I9fUkVTYrqt8urWORiaDQZnxtj9-xXtdf9MCRjTAXm-okQyKeho-gWcpSfbIJtVi4eGVfnA | Assembly of the gearboxes required motors, gearbox housings, small cogs, gears, drive shafts, screws and spring pins.  First the small gears were attached to the motors and both fitted into the gearbox housing. Next the drive shaft was fitted which holds the gears in place and thus completing the gearbox housing. A spring pin is placed in the drive shaft to create a seat for the wheels which can be attached later.  To create a connection from the motors to the motor board 3 resistors had to be soldered onto the terminals and to the side once the side has been slightly filed as to remove the oxidizing outer layer of metal; filing allows the solder to grip the motor. This set up of resistors helps to reduce noise generated by the motors.  Wires were then attached to the resistors that connect to the motor terminals and slotted into a connector head for easy installation on the motor board. |

Written by Zack Raeburn

## **Power Supply Wiring**

|  |  |
| --- | --- |
|  | The Power supply wiring had to be made from scratch, using just wires, battery clips and heat shrink wrap. We soldered the battery clips together in series, soldered the wires to the switches and then covered the wires in Heat shrink wrap. We then mounted the both the switches and the battery clips onto the backboard. These wires provide power for both the Main control board and the Motor control board. |

Written by Daniel McCluskey

## **Body Assembly**

|  |  |
| --- | --- |
|  | This task required four flathead screws to be inserted from the underside of the main board and to put spring washers on the opposing side along with four 30mm studs. After the tightening of the screws, two screw and washer assemblies had to be prepared along with two more studs, the cable from the MCU board power was then aligned in the centre of the two previously screwed in studs and the assemblies were fastened to the tops of the studs holding together the base board and the servo mount board.  The battery boxes then had to be mounted, this was done by enlarging the mounting holes on the battery boxes to fit the 3.5mm screw bits using a hand drill with a 3.5mm drill bit. From the underside of the main board flathead screws were inserted to the previously enlarged holes and the battery packs were mounted on top of the opposing side of the screws so that the terminals faced inwards. The Battery containers were finally fastened into place using washers and nuts. The power cables then got snapped to the packs to complete the task.  The servo then had to be mounted and so the motor was placed into the opening in the servo support plate, the studs that were partially tightened earlier were removed and the servo back reinforcing plates were positioned correctly and screwed into place. The servo was then secured with screwand washer assemblies on all four holes whilst attaching the front support plate to the front two holes.  The flat cable then had to be prepared so a 120mm length of cable was cut and got a 10-pin female connector to one side of the cable and the other side of the cable had to be separated before being placed into a 10-pin connector. The remaining cable then had 10-pin connectors attached to both ends.  The next part was to mount the MCU board and so flathead screws were inserted into the bottom of the main board and spring washers on the opposing side along with four stud bolts. The sensor board was then attached with the long flat cable and the cable was bound up the side of the front wheel assembly then passed through the opening between the toggle switches and plugged into the MCU board in port 0. The MCU board was then positioned to match the stud positions and then the nuts were screwed on to secure the board. The white connector of the MCU board power cable was then connected.  The next task to complete was the mounting of the motor drive board which was done by placing the board over the stud positions and attaching a flat washer then a spring washer to the end of each stud followed by a nut that was tightened to finish securing the board. The motor cables were then connected followed by the power cable of the motor drive and the servo motor cable. The flat cable with reconfigured wire connections was then inserted into the Cn3 socket on the RMC-RX62T board and the opposite side went into the motor drive board. |

## **Mounting the Wheels**

|  |  |
| --- | --- |
|  | In order to mount the wheels, the shafts on both ends of the car had to be fitted into the notch of the wheel hub making especially sure for the rear wheels that the spring pin was in place before securing both sets of wheels with a spring washer and nut combination. |

Written by Connor Liggins

## 

## **Construction of the MCU**

|  |  |
| --- | --- |
|  | Building the main control unit (MCU) was fairly simple. Most of the components were already attached to the board. All that had to be done was soldering on the power and data connectors.  First the power connector, a white two pin connector block, was placed on the board at connection point CN5; ensuring the orientation matches the outline on the board. Then while holding the connector in place the board was turned over and the protruding pins were soldered to the pads on the solder side of the board.  The same process was followed for the data connectors the only difference being that they are both 10 pin black connectors. These were placed at connection points CN2 and CN3. |

Written by Thomas Hopkins

## **Constructing the Sensor Board**

|  |  |
| --- | --- |
|  | The sensor board was more complicated to construct. Each component had to be individually held in the correct place and soldered on.  First the IR photo sensors were placed on the solder side of the board in their respective locations and soldered in place from the component side. Then the pins of the start bar sensor were bent so that the sensor points forward when it is in place. It was then soldered in place on the component side of the board.  Next the IR LEDs were soldered in place on the component side of the board and bent into the adjacent holes. This was done in order to direct the light produced at the track for the photo sensors to read the reflection.  Then the red LEDs were put in place on the component side of the board and soldered on from the solder side. Using the same method, the resistors, capacitors, resistor array, variable resistors and the 10 pin data connector are soldered to the board. For each component care was taken to ensure the positioning and orientation was correct. The resistors and capacitors must be place in specific locations based on their resistance or capacitance values. The resistor array must be placed so that the spot lined up with the square solder pad on the board. The variable resistors will only fit on the board in one orientation.  Finally, a strip of polyester pile tape was stuck to the front edge of the solder side of the board. This completed the sensor board. Once bolted on and plugged in the sensor board worked first time with no additional changes. |

Written by Thomas Hopkins

**Constructing the Motor control board**

|  |  |
| --- | --- |
|  | The construction of the motor board required lots of precision soldering. To construct the Motor board, we needed to solder each individual component onto the blank Motor Board. This meant making sure that the components were in the right orientation and that the right components were in the correct place.  A problem we came across was the "bridging" of the gaps between some of the components, but we were able to overcome the problem by re-soldering the affected components.  When the board was complete we tested it with a continuity tester in order to check that there were no "Ghost" joints where the solder had not correctly connected the Component to the board.  After that we mounted the Motor board to the Body of the car and plugged in the appropriate wires such as the Gearbox wires and the connection to the logic board. |

Written by Daniel McCluskey

## 

## **Future additions**

|  |  |
| --- | --- |
|  | In the future the car will need a Light blocker placed on the front of the sensor, this is so the lighting of the room that the track is in at the Renesas MCU car rally does not affect the sensors, since the lighting will always by the same under the sensor. |

**Management**

|  |  |
| --- | --- |
|  | During the construction of the car, it was essential to have a schedule so that the whole team knew when they were required to attend meetings and have their parts of the car finished so that they could meet the deadline of the assignment. The Scheduling system that Morgan James came up with was very successful, it let the group know that there was 2 meetings every week. |
|  | During the building of the car we had to assign different tasks to each of the group members, to do this we used a piece of software called Trello, we inputted the different tasks that would need to be completed such as “Soldering the Motor Board” or “Constructingthe Wheels”. Trello allowed us to assign specific tasks to different group members and then easily tick off what we had completed so the other group members were always up to date on how close we were to the completion of the  car. |

# **Testing**

# **TEST LOG – Recommended Hardware Testing**

Before the car can be taken to the Renesas MCU car rally, every aspect of the car needs to be vigorously tested. These tests will ensure that the car does not break or fault when it is used during the competition. The code was installed on to the car and we edited the code to test it at different stages e.g. separate code would have to be installed in order to test the speed of the motors.

## **Testing Table**

|  |  |  |
| --- | --- | --- |
| **Test Description** | **Expectation** | **Result** |
| Test the LEDs on the sensor board to see if they light up when the sensor linked to it is activated | Each LED should light up when the connecting sensor is activated | Pass  The LEDs flashed when it was turn on and then started detecting the track below it. |
| Test to see if the button on the motor board responds to being pushed. | When the Button on top of the motor board is pressed it should make the car start. | Pass  When the button was pressed, the motor board started and the car started to move. |
| Test the servo motor for general functionality` | The servo motor should function normally | Pass  The servo motor functioned correctly. The servo motor was able to turn in both directions and was easily calibrated |
| Test the right motor for general functionality, forwards and backwards | When the Right motor is turned on it should run at the expected speed in the appropriate direction. | Pass  When the test was executed, the right motor was able to turn forwards and backwards |
| Test the left motor for general functionality, forwards and backwards | When the Left motor is turned on it should run at the expected speed in the appropriate direction. | Pass  When the test was executed, the left motor was able to turn forwards and backwards |
| Test the motor board switch to see if it turns on correctly | When the switch is toggled, the Motor board should turn on and it's LED’s should flash. | Pass  When the switch was toggled, the motor boards LEDs flashed and the program waited to be executed. |
| Test the logic board switch to see if it turns on correctly | When the switch is toggled, the Logic board should turn on and it's LED’s should flash. | Pass  When the switch was toggled, the Logic boards LEDs turned on and the program waited to be executed. |
| Test if the car can drive forwards for 2 seconds at 50% speed | When the car starts, it should accelerate for 2 seconds at 50% speed. | Pass  The cars motors were able to run at 50% speed for 2 seconds. |
| Test if the car can drive forwards for 5 seconds at 50% speed | When the car starts, it should accelerate for 5 seconds at 50% speed. | Pass  The cars motors were able to run at 50% speed for 5 seconds. |
| Test if the car can drive forwards for 10 seconds at 50% speed | When the car starts, it should accelerate for 10 seconds at 50% speed. | Pass  The cars motors were able to run at 50% speed for 10 seconds. |
| Test if the car can drive forwards for 2 seconds at 100% speed | When the car starts, it should accelerate for 2 seconds at 100% speed. | Pass  The cars motors were able to run at 100% speed for 2 seconds. |
| Test if the car can drive forwards for 5 seconds at 100% speed | When the car starts, it should accelerate for 5 seconds at 100% speed. | Pass  The cars motors were able to run at 100% speed for 5 seconds. |
| Test if the car can drive forwards for 10 seconds at 100% speed | When the car starts, it should accelerate for 10 seconds at 100% speed. | Pass  The cars motors were able to run at 100% speed for 10 seconds. |
| Test the car can follow a straight piece of track for at least 3 seconds | When placed on the track, the car should accelerate along the straight track for at least 3 seconds. | Fail  When placed on a straight track the car went straight to 100% speed and quickly derailed on the other side. |
| Test the car can follow a left curved track | The car should follow the curved track and not derail. | Pass  The car was able to follow a left bend track and did not derail. |
| Test the car can follow a right curved track | The car should follow the curved track and not derail. | Pass  The car was able to follow a right bend track and did not derail. |
| Test the car can react appropriately to a right angle turn going left | The car should make a left 90 degree turn after going over 2 white strips on the track. | Fail  The car was unable to do a right angled turn, instead it continued going straight and derailed. |
| Test the car can react appropriately to a right angle turn going right | The car should make a right 90 degree turn turn after going over 2 white strips on the track. | Fail  The car was unable to do a left 90 degree turn, instead it continued going straight and derailed. |
| Test the car can react appropriately to a lane change going left | The car should make the lane change after going over 2 half white stripes and should shift to the appropriate lane. | Fail  The car started turning to make the lane change but never turned back and continued going in a circle. |
| Test the car can react appropriately to a lane change going right | The car should make the lane change after going over 2 half white stripes and should shift to the appropriate lane. | Fail  The car started turning to make the lane change but never turned back and continued going in a circle. |

## **Test Conclusions**

The hardware testing shows there are some minor issues with the vehicle behaviour which have been highlighted in the testing table above. The first set of tests to fail were the lane change tests, this was due to the program on the micro controller not knowing how to handle the signal for the lane change on the track, the code will be edited so that the car knows what to do in the event of a lane change. Another problem that occurred was the car not being able to do 90 degree turns. However, the code was edited and the car was able to do 90 degree turns before the end of the assignment. Another problem that occurred was that the car was going to fast on straight tracks, meaning that if it came to a corner it would shoot off the track. This will be fixed by changing the speed of the motors on straight tracks within the program for the car.

## **Overview**

The main components consist of: the RMC-RX62T MCU board, a “Sensor board Ver 5” and a “Motor drive board Ver 5” (Renesas,2014). Without modifications the car seems to perform well being powered by 8 batteries. Following the guide of the Renesas Manual (Renesas,2014) we created two battery packs containing 4 batteries each which allowed for less drain on the controlling signals that were in binary which controlled the operation of the build in the RMC RX62T MCU board.

## **Code analysis**

There is an infinite “while loop” proceeding through most of the code that allows the car to move at all times whilst also monitoring. Within this loop there is a “switch” statement(learn-c.org ‘While loops’ (n.d).)

All code described in the following code analysis refers to the default program "kit12\_rx62t.c" provided with the car by Renesas.

### Line 19 "#include "iodefine.h"":

This line includes the framework so that the program can correctly interface with the Logic control board for the car, the framework makes it easier to write the code to interface with the motor board, the servo motor and the sensor board from the logic controller.

### Line 31-40 Bit Masking:

This is the part of the code which defines the bit masks to make it easier to interpret what the sensor board is returning to the logic board. For example, it defines MASK4\_4 as 0xff which tells the logic board that there is no white track under the sensor board, this will make the program return to its standby state.

### Line 44-58 Function Declarations:

These lines declare the functions for the program, it tells the program to expect to see the functions below in the program. This is so that you can call functions before writing the function body without causing an error. For example, line 44 declares the init function, and line 72 calls the init function and lines 484-558 contains the function body. If line 44 did not declare the function, then the program would crash on line 72 since it does not know that the function actually exists yet.

### Line 62-64 Global Variables:

These lines contain the global variables that are required across the whole program. They are declared outside of any function like the main body so that they are in scope for all of the functions. Declaring them inside of a function would cause them to be out of scope to any other functions. For example, if the pattern variable was declared in the main function, then the init function would not be able to access it since it would be out of scope.

### Line 72 Initialising the car:

This is the line the calls the init function on lines 484-558 which contain the function body. This initialises all of the variables required so that the addresses for the components in the code can be interpreted by the micro controller. Without the init function, the user would have to manually type out each of the addresses every time they needed to reference it.

### Line 75 Handle Function:

The handle function controls the rotation of the servo motor, depending on what number it is given, it will move rotate the servo motor a certain amount of degrees, in turn, rotating the sensor board and the front of the car making it turn. For example if the code "handle(0);" is called, the servo motor will reset to its centre position. A positive value will make the servo motor turn the car left, and a negative value will cause the car to turn right.

### Line 76 Motor Function:

The Motor function controls the speed of both of the motors on the motor board. The syntax for the function is "motor("Left Motor Speed", "Right Motor Speed")". The value is given as a percentage e.g. feeding the value '100' into the function will cause the appropriate motor to run at 100% speed. The first value fed to the function will control the speed of the left motor, the value fed into the second input will change the speed of the right motor. A Lower value will make the motor spin slower and a higher value will make the motor spin faster. A value below 0 will cause the motors to reverse direction. If both the values are the same the car will go straight, if the values are different, the car will turn slightly in the direction of the slower motor.

### Line 79 Pattern switch statement:

The switch statement on line 79 is the main function of the program, it is contained within a while loop so it will run until the Micro controller loses power. The switch statement is like an if statement, but makes it easier to check a lot of values at once. The switch statement checks what the variable "pattern" is every time it gets called and executes a certain case depending on its value. For example, it will execute case 0 on the very first run through since that what the variable initialises as. Each case has a short description in the multi-line comment at the top of the switch brackets to tell you what each pattern is for. For example, Case 51 is for processing the 1st half line of the lane change signal. The pattern variable is changed within each switch statement so the program does not get stuck within 1 case.

### Line 105-120 Waiting for Switch input:

This is the case that runs when the Microcontroller is first initialised (Or switched on), it checks for the button on top of the motor board being pressed and toggles the LED on a timer whilst the button is not being pressed.

### Line 121-137 Checking if car blocker has moved:

This is the case that runs when the button is pressed from case 0, this case checks for an object in front of the sensor board, the object is the start bar in the final race at the Renesas MCU rally, but during testing we used a piece of a track to simulate it. Whilst the start bar is present, the program loops through the same case and toggles an LED so that it flashes. When the start bar has been removed, the pattern variable gets set to 11 and the program leaves the case using "break;". The presence of the start bar is checked by the startbar\_get() function. Which is on line 601.

### Line 139 -213 Track detection:

This is the most used case in the switch statement, this is the case that handles the input received from the sensor board for detecting what track the car is on. The case first checks if it has run over any strips which are used to signal that either a 90-degree turn is approaching or if a lane change is approaching. If it does detect any white strip, it will change the pattern and will call "break;" so it will change to the appropriate pattern.

#### Case 0x00:

This is the case where the input means the car is going on a straight and is in the centre and so tells the motor to go 100% throttle.

#### Case 0x04:

This is the case where the input tells the car it’s just left of the centre and so should perform a slight right hand turn of the servo to realign itself and so the motors will both still go at 100% throttle.

#### Case 0x06:

This is the case where the input tells the car it is quite a bit left of the centre and so should perform a slight right hand turn and so the left motor will go at 80% throttle and the right motor will go at 67% throttle.

#### Case 0x07:

This is the case where the input tells the car it is a good portion left of the centre and so should perform a right hand turn and so the left motor will go at 50% throttle and the right motor will go at 38% throttle.

#### Case 0x03:

This is the case where the input tells the car it is a large distance left of the centre and so should perform a large right hand turn and so the left motor will go at 30% throttle and the right motor will go at 19% throttle.

#### Case 0x20:

This is the case where the input tells the car it’s just right of the centre and so should perform a slight left hand turn of the servo to realign itself and so the motors will both still go at 100% throttle.

#### Case 0x60:

This is the case where the input tells the car it is quite a bit right of the centre and so should perform a slight left hand turn and so the right motor will go at 80% throttle and the left motor will go at 67% throttle.

#### Case 0xe0:

This is the case where the input tells the car it is a good portion right of the centre and so should perform a left hand turn and so the right motor will go at 50% throttle and the left motor will go at 38% throttle.

#### Case 0xc0:

This is the case where the input tells the car it is a large distance right of the centre and so should perform a large left hand turn and so the right motor will go at 30% throttle and the left motor will go at 19% throttle.

### Line 215-251 Checking the second strips for turn detection

These are the cases that check if the car is still on the track after performing a large right turn from Case 11. It makes the necessary adjustments depending on what the sensor board returns to the Logic board. For example, if it detects the white line of the side of the track and the white line of the centre of the track, the program returns so that it can continue turning, but if the track is back in the centre, the program returns to the case 11 and continues track detection again.

### Line 253-348 90 Degree turns:

The program can only get to these cases is if Case 11 detects the first 90 degree turn signal on the track, these cases first check for the second white strip that signals the 90 degree turn, after a timer counts down, if the second strip is not found, the program returns to case 11 (Track detection). However, if the second strip is found, the program will change to case 23, this waits for the sensor board to detect whether it is a left or right 90 degree turn. Once it detects track on one side of the sensor board it will change to the appropriate case (Case 41 for right turn or Case 31 for Left turn). The program will then turn the car in the appropriate direction and switch to another case which waits for the sensor board to be centred and then straightens out the car so it will drive straight.

### Line 349-469 Lane changing:

The program starts by detecting that there is a half line on either side of the track and moving on to the relevant case depending on the position of the half line (left lane change moves to case 61 and right lane change moves to case 51). It then waits until the sensor board detects the second line before moving onto the relevant case (Case 63 for left lane change and case 53 for a right lane change). That case will then wait until it cannot detect any white lines across the track before changing the direction of the car and moving onto the relevant direction case (case 64 for a right lane change and case 61 for a left lane change) where it will wait until the sensor detects that the car is back in the middle of the track. The program then switches over to case 11 which will redirect the car so that it can finish turning and continue straight down the track.

### Line 473-476 Default case:

The default case changes the pattern back to the initial state where the program waits for the user to press the button on the motor board to start the car.

# **Conclusion**

In conclusion I think that the group worked very well as a team and the building the car went as well as it could have given the circumstances. Every group member contributed as much as they could and we all put in our utmost effort in order to ensure that the car was completed before the deadline.

One part of the building process that we would like to improve on is the soldering of the motor board, if we had taken more time to solder the board, we would not have encountered any issues with “Bridging” and would have been able to complete the board in a timelier matter. Another improvement we would make would to schedule times with the University workshop so we knew when we could work on the project and not get turned down when the workshop was in use by other students. These two improvements would have vastly decreased the amount of time that it took to build the car.

We feel confident that the car will be in perfect condition and we will easily be able to fine tune the program ready for the Renesas MCU Car rally in March and that we will be strong competition for our competitors.

# **Team Input**

|  |  |
| --- | --- |
| **Task** | **Group member with percentage of input** |
| **Building Stages** | |
| **Gearbox assembly and motor** | Zack R - 100% |
| **Board cut** | Morgan J - 100% |
| **Wheel assembly** | Connor L - 100% |
| **Servo motor assembly** | Morgan J - 100% |
| **Drilling and preparing screw holes** | Morgan J - 100% |
| **wiring** | Dan M - 50%  TomH - 50% |
| **Chassis** | Morgan J - 20%, Dan M - 20%, Tom H - 20%, Connor L - 20%, Zack R - 20% |
| **Sensor & MCU** | Tom H - 100% |
| **Motor board soldering and mounting** | Dan M - 50%, Tom H - 50% |
| **Finishing touches** | Morgan J - 20%, Dan M - 20%, Tom H - 20%, Connor L - 20%, Zack R - 20% |
| **Testing** | |
| **Recommended hardware testing** | Morgan J - 20%, Dan M - 20%, Tom H - 20%, Connor L - 20%, Zack R - 20% |
| **Coding** | |
| **Research and analysis of code** | Morgan J - 20%, Dan M - 20%, Tom H - 20%, Connor L - 20%, Zack R - 20% |
| **Testing** | |
| **Code testing** | Morgan J - 20%, Dan M - 20%, Tom H - 20%, Connor L - 20%, Zack R - 20% |

|  |  |
| --- | --- |
| **Conclusions of team members contributions** | |
| Morgan James | Morgan was a very effective team member. As team leader he encouraged communication between team members and supported everyone to complete the project. He was always present and punctual for team meetings and kept the team’s enthusiasm high. He proved to be a valuable team member especially during the build stages where he provided much needed knowledge in construction techniques such as cutting and filing. |
| Connor Liggins | Connor was a very effective and knowledgeable team member. His contribution especially during the build stages kept the project running smoothly. As he lives on campus he was chosen to store the car, safe in his room. This meant that we relied upon him to be able to get it to the workshop when anyone wanted to work on it. He was always eager to get involved and whenever we got stuck he was the first to pick up the manual and research the issue we were having. Additionally his nimble fingers proved useful in the construction of the wheels as this required bolting small parts together and ensuring that they weren’t going to separate. |
| Zack Raeburn | Zack was a superb team member providing moral support and asking the questions no one else was willing to ask. He was always present and on time and proved very knowledgeable and capable when dealing with mechanical components when constructing the gearboxes. His soldering was not the cleanest but it was functional. Even though a few mistakes were made, they were easily remedied and didn’t cost us anything. Furthermore, his work was generally of a high standard. |
| Thomas Hopkins | Tom proved to be very capable with the technical aspects of the project. He was very good at fault finding and diagnosing both hardware and software issues. He was knowledgeable enough in electronic hardware components to diagnose burnt out mosfets on the motor driver board. His soldering ability was impressive with very clean connections and no dry joints. Overall he was a very effective team member even if there were a few mistakes made along the way. None of these caused any damage and his work was always to a very high standard. |
| Daniel McCluskey | Dan was an enthusiastic team member. His soldering and construction of the motor driver board was good if a little rushed. This left a short on the board which subsequently burnt out a mosfet. Although this cost us some time in the construction stages it did not harm the project overall and with some support from other team members the board was completed to a high standard. Additionally he proved very capable during the final assembly, providing techniques that made the bolting together of components much simpler. |

References.

No Author (2014). Renesas Program Explanation Manual. MCU Car Rally Secretariat, Renesas Solutions Corp.

No Author (n.d) ‘learn-c.org. While loops’<http://www.learn-c.org/en/While_loops> (accessed 15/01/2016)

Appendices

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\* Supported Microcontroller:RX62T \*/  
/\* File: kit12\_rx62t.c \*/  
/\* File Contents: MCU Car Trace Basic Program(RX62T version) \*/  
/\* Version number: Ver.1.00 \*/  
/\* Date: 2013.09.01 \*/  
/\* Copyright: Renesas Micom Car Rally Secretariat \*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\*  
This program supports the following boards:  
\* RMC-RX62T board  
\* Sensor board Ver. 5  
\* Motor drive board Ver. 5  
\*/  
  
/\*======================================\*/  
/\* Include \*/  
/\*======================================\*/  
#include "iodefine.h"  
  
/\*======================================\*/  
/\* Symbol definitions \*/  
/\*======================================\*/  
  
/\* Constant settings \*/  
#define PWM\_CYCLE 24575 /\* Motor PWM period (16ms) \*/  
#define SERVO\_CENTER 2300 /\* Servo center value \*/  
#define HANDLE\_STEP 13 /\* 1 degree value \*/  
  
/\* Masked value settings X:masked (disabled) O:not masked (enabled) \*/  
#define MASK2\_2 0x66 /\* X O O X X O O X \*/  
#define MASK2\_0 0x60 /\* X O O X X X X X \*/  
#define MASK0\_2 0x06 /\* X X X X X O O X \*/  
#define MASK3\_3 0xe7 /\* O O O X X O O O \*/  
#define MASK0\_3 0x07 /\* X X X X X O O O \*/  
#define MASK3\_0 0xe0 /\* O O O X X X X X \*/  
#define MASK4\_0 0xf0 /\* O O O O X X X X \*/  
#define MASK0\_4 0x0f /\* X X X X O O O O \*/  
#define MASK4\_4 0xff /\* O O O O O O O O \*/  
  
/\*======================================\*/  
/\* Prototype declarations \*/  
/\*======================================\*/  
void init(void);  
void timer( unsigned long timer\_set );  
unsigned char sensor\_inp( unsigned char mask );  
unsigned char startbar\_get( void );  
int check\_crossline( void );  
int check\_rightline( void );  
int check\_leftline( void );  
unsigned char dipsw\_get( void );  
unsigned char buttonsw\_get( void );  
unsigned char pushsw\_get( void );  
void led\_out\_m( unsigned char led );  
void led\_out( unsigned char led );  
void motor( int accele\_l, int accele\_r );  
void handle( int angle );  
  
/\*======================================\*/  
/\* Global variable declarations \*/  
/\*======================================\*/  
unsigned long cnt0;  
unsigned long cnt1;  
int pattern;  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\* Main program \*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
void main(void)  
{  
 /\* Initialize MCU functions \*/  
 init();  
  
 /\* Initialize micom car state \*/  
 handle( 0 );  
 motor( 0, 0 );  
  
 while( 1 ) {  
 switch( pattern ) {  
  
 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 Pattern-related  
 0: wait for switch input  
 1: check if start bar is open  
 11: normal trace  
 12: check end of large turn to right  
 13: check end of large turn to left  
 21: processing at 1st cross line  
 22: read but ignore 2nd time  
 23: trace, crank detection after cross line  
 31: left crank clearing processing ? wait until stable  
 32: left crank clearing processing ? check end of turn  
 41: right crank clearing processing ? wait until stable  
 42: right crank clearing processing ? check end of turn  
 51: processing at 1st right half line detection  
 52: read but ignore 2nd line  
 53: trace after right half line detection  
 54: right lane change end check  
 61: processing at 1st left half line detection  
 62: read but ignore 2nd line  
 63: trace after left half line detection  
 64: left lane change end check  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
 case 0:  
 /\* Wait for switch input \*/  
 if( pushsw\_get() ) {  
 pattern = 1;  
 cnt1 = 0;  
 break;  
 }  
 if( cnt1 < 100 ) { /\* LED flashing processing \*/  
 led\_out( 0x1 );  
 } else if( cnt1 < 200 ) {  
 led\_out( 0x2 );  
 } else {  
 cnt1 = 0;  
 }  
 break;  
  
 case 1:  
 /\* Check if start bar is open \*/  
 if( !startbar\_get() ) {  
 /\* Start!! \*/  
 led\_out( 0x0 );  
 pattern = 11;  
 cnt1 = 0;  
 break;  
 }  
 if( cnt1 < 50 ) { /\* LED flashing processing \*/  
 led\_out( 0x1 );  
 } else if( cnt1 < 100 ) {  
 led\_out( 0x2 );  
 } else {  
 cnt1 = 0;  
 }  
 break;  
  
 case 11:  
 /\* Normal trace \*/  
 if( check\_crossline() ) { /\* Cross line check \*/  
 pattern = 21;  
 break;  
 }  
 if( check\_rightline() ) { /\* Right half line detection check \*/  
 pattern = 51;  
 break;  
 }  
 if( check\_leftline() ) { /\* Left half line detection check \*/  
 pattern = 61;  
 break;  
 }  
 switch( sensor\_inp(MASK3\_3) ) {  
 case 0x00:  
 /\* Center -> straight \*/  
 handle( 0 );  
 motor( 100 ,100 );  
 break;  
  
 case 0x04:  
 /\* Slight amount left of center -> slight turn to right \*/  
 handle( 5 );  
 motor( 100 ,100 );  
 break;  
  
 case 0x06:  
 /\* Small amount left of center -> small turn to right \*/  
 handle( 10 );  
 motor( 80 ,67 );  
 break;  
  
 case 0x07:  
 /\* Medium amount left of center -> medium turn to right \*/  
 handle( 15 );  
 motor( 50 ,38 );  
 break;  
  
 case 0x03:  
 /\* Large amount left of center -> large turn to right \*/  
 handle( 25 );  
 motor( 30 ,19 );  
 pattern = 12;  
 break;  
  
 case 0x20:  
 /\* Slight amount right of center -> slight turn to left \*/  
 handle( -5 );  
 motor( 100 ,100 );  
 break;  
  
 case 0x60:  
 /\* Small amount right of center -> small turn to left \*/  
 handle( -10 );  
 motor( 67 ,80 );  
 break;  
  
 case 0xe0:  
 /\* Medium amount right of center -> medium turn to left \*/  
 handle( -15 );  
 motor( 38 ,50 );  
 break;  
  
 case 0xc0:  
 /\* Large amount right of center -> large turn to left \*/  
 handle( -25 );  
 motor( 19 ,30 );  
 pattern = 13;  
 break;  
  
 default:  
 break;  
 }  
 break;  
  
 case 12:  
 /\* Check end of large turn to right \*/  
 if( check\_crossline() ) { /\* Cross line check during large turn \*/  
 pattern = 21;  
 break;  
 }  
 if( check\_rightline() ) { /\* Right half line detection check \*/  
 pattern = 51;  
 break;  
 }  
 if( check\_leftline() ) { /\* Left half line detection check \*/  
 pattern = 61;  
 break;  
 }  
 if( sensor\_inp(MASK3\_3) == 0x06 ) {  
 pattern = 11;  
 }  
 break;  
  
 case 13:  
 /\* Check end of large turn to left \*/  
 if( check\_crossline() ) { /\* Cross line check during large turn \*/  
 pattern = 21;  
 break;  
 }  
 if( check\_rightline() ) { /\* Right half line detection check \*/  
 pattern = 51;  
 break;  
 }  
 if( check\_leftline() ) { /\* Left half line detection check \*/  
 pattern = 61;  
 break;  
 }  
 if( sensor\_inp(MASK3\_3) == 0x60 ) {  
 pattern = 11;  
 }  
 break;  
  
 case 21:  
 /\* Processing at 1st cross line \*/  
 led\_out( 0x3 );  
 handle( 0 );  
 motor( 0 ,0 );  
 pattern = 22;  
 cnt1 = 0;  
 break;  
  
 case 22:  
 /\* Read but ignore 2nd line \*/  
 if( cnt1 > 100 ){  
 pattern = 23;  
 cnt1 = 0;  
 }  
 break;  
  
 case 23:  
 /\* Trace, crank detection after cross line \*/  
 if( sensor\_inp(MASK4\_4)==0xf8 ) {  
 /\* Left crank determined -> to left crank clearing processing \*/  
 led\_out( 0x1 );  
 handle( -38 );  
 motor( 10 ,50 );  
 pattern = 31;  
 cnt1 = 0;  
 break;  
 }  
 if( sensor\_inp(MASK4\_4)==0x1f ) {  
 /\* Right crank determined -> to right crank clearing processing \*/  
 led\_out( 0x2 );  
 handle( 38 );  
 motor( 50 ,10 );  
 pattern = 41;  
 cnt1 = 0;  
 break;  
 }  
 switch( sensor\_inp(MASK3\_3) ) {  
 case 0x00:  
 /\* Center -> straight \*/  
 handle( 0 );  
 motor( 40 ,40 );  
 break;  
 case 0x04:  
 case 0x06:  
 case 0x07:  
 case 0x03:  
 /\* Left of center -> turn to right \*/  
 handle( 8 );  
 motor( 40 ,35 );  
 break;  
 case 0x20:  
 case 0x60:  
 case 0xe0:  
 case 0xc0:  
 /\* Right of center -> turn to left \*/  
 handle( -8 );  
 motor( 35 ,40 );  
 break;  
 }  
 break;  
  
 case 31:  
 /\* Left crank clearing processing ? wait until stable \*/  
 if( cnt1 > 200 ) {  
 pattern = 32;  
 cnt1 = 0;  
 }  
 break;  
  
 case 32:  
 /\* Left crank clearing processing ? check end of turn \*/  
 if( sensor\_inp(MASK3\_3) == 0x60 ) {  
 led\_out( 0x0 );  
 pattern = 11;  
 cnt1 = 0;  
 }  
 break;  
  
 case 41:  
 /\* Right crank clearing processing ? wait until stable \*/  
 if( cnt1 > 200 ) {  
 pattern = 42;  
 cnt1 = 0;  
 }  
 break;  
  
 case 42:  
 /\* Right crank clearing processing ? check end of turn \*/  
 if( sensor\_inp(MASK3\_3) == 0x06 ) {  
 led\_out( 0x0 );  
 pattern = 11;  
 cnt1 = 0;  
 }  
 break;  
  
 case 51:  
 /\* Processing at 1st right half line detection \*/  
 led\_out( 0x2 );  
 handle( 0 );  
 motor( 0 ,0 );  
 pattern = 52;  
 cnt1 = 0;  
 break;  
  
 case 52:  
 /\* Read but ignore 2nd time \*/  
 if( cnt1 > 100 ){  
 pattern = 53;  
 cnt1 = 0;  
 }  
 break;  
  
 case 53:  
 /\* Trace, lane change after right half line detection \*/  
 if( sensor\_inp(MASK4\_4) == 0x00 ) {  
 handle( 15 );  
 motor( 40 ,31 );  
 pattern = 54;  
 cnt1 = 0;  
 break;  
 }  
 switch( sensor\_inp(MASK3\_3) ) {  
 case 0x00:  
 /\* Center -> straight \*/  
 handle( 0 );  
 motor( 40 ,40 );  
 break;  
 case 0x04:  
 case 0x06:  
 case 0x07:  
 case 0x03:  
 /\* Left of center -> turn to right \*/  
 handle( 8 );  
 motor( 40 ,35 );  
 break;  
 case 0x20:  
 case 0x60:  
 case 0xe0:  
 case 0xc0:  
 /\* Right of center -> turn to left \*/  
 handle( -8 );  
 motor( 35 ,40 );  
 break;  
 default:  
 break;  
 }  
 break;  
  
 case 54:  
 /\* Right lane change end check \*/  
 if( sensor\_inp( MASK4\_4 ) == 0x3c ) {  
 led\_out( 0x0 );  
 pattern = 11;  
 cnt1 = 0;  
 }  
 break;  
  
 case 61:  
 /\* Processing at 1st left half line detection \*/  
 led\_out( 0x1 );  
 handle( 0 );  
 motor( 0 ,0 );  
 pattern = 62;  
 cnt1 = 0;  
 break;  
  
 case 62:  
 /\* Read but ignore 2nd time \*/  
 if( cnt1 > 100 ){  
 pattern = 63;  
 cnt1 = 0;  
 }  
 break;  
  
 case 63:  
 /\* Trace, lane change after left half line detection \*/  
 if( sensor\_inp(MASK4\_4) == 0x00 ) {  
 handle( -15 );  
 motor( 31 ,40 );  
 pattern = 64;  
 cnt1 = 0;  
 break;  
 }  
 switch( sensor\_inp(MASK3\_3) ) {  
 case 0x00:  
 /\* Center -> straight \*/  
 handle( 0 );  
 motor( 40 ,40 );  
 break;  
 case 0x04:  
 case 0x06:  
 case 0x07:  
 case 0x03:  
 /\* Left of center -> turn to right \*/  
 handle( 8 );  
 motor( 40 ,35 );  
 break;  
 case 0x20:  
 case 0x60:  
 case 0xe0:  
 case 0xc0:  
 /\* Right of center -> turn to left \*/  
 handle( -8 );  
 motor( 35 ,40 );  
 break;  
 default:  
 break;  
 }  
 break;  
  
 case 64:  
 /\* Left lane change end check \*/  
 if( sensor\_inp( MASK4\_4 ) == 0x3c ) {  
 led\_out( 0x0 );  
 pattern = 11;  
 cnt1 = 0;  
 }  
 break;  
  
 default:  
 /\* If neither, return to standby state \*/  
 pattern = 0;  
 break;  
 }  
 }  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\* RX62T Initialization \*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
void init(void)  
{  
 // System Clock  
 SYSTEM.SCKCR.BIT.ICK = 0; //12.288\*8=98.304MHz  
 SYSTEM.SCKCR.BIT.PCK = 1; //12.288\*4=49.152MHz  
  
 // Port I/O Settings  
 PORT1.DDR.BYTE = 0x03; //P10:LED2 in motor drive board  
  
 PORT2.DR.BYTE = 0x08;  
 PORT2.DDR.BYTE = 0x1b; //P24:SDCARD\_CLK(o)  
 //P23:SDCARD\_DI(o)  
 //P22:SDCARD\_DO(i)  
 //CN:P21-P20  
 PORT3.DR.BYTE = 0x01;  
 PORT3.DDR.BYTE = 0x0f; //CN:P33-P31  
 //P30:SDCARD\_CS(o)  
 //PORT4:input //sensor input  
 //PORT5:input  
 //PORT6:input  
  
 PORT7.DDR.BYTE = 0x7e; //P76:LED3 in motor drive board  
 //P75:forward reverse signal(right motor)  
 //P74:forward reverse signal(left motor)  
 //P73:PWM(right motor)  
 //P72:PWM(left motor)  
 //P71:PWM(servo motor)  
 //P70:Push-button in motor drive board  
 PORT8.DDR.BYTE = 0x07; //CN:P82-P80  
 PORT9.DDR.BYTE = 0x7f; //CN:P96-P90  
 PORTA.DR.BYTE = 0x0f; //CN:PA5-PA4  
 //PA3:LED3(o)  
 //PA2:LED2(o)  
 //PA1:LED1(o)  
 //PA0:LED0(o)  
 PORTA.DDR.BYTE = 0x3f; //CN:PA5-PA0  
 PORTB.DDR.BYTE = 0xff; //CN:PB7-PB0  
 PORTD.DDR.BYTE = 0x0f; //PD7:TRST#(i)  
 //PD5:TDI(i)  
 //PD4:TCK(i)  
 //PD3:TDO(o)  
 //CN:PD2-PD0  
 PORTE.DDR.BYTE = 0x1b; //PE5:SW(i)  
 //CN:PE4-PE0  
  
 // Compare match timer  
 MSTP\_CMT0 = 0; //CMT Release module stop state  
 MSTP\_CMT2 = 0; //CMT Release module stop state  
  
 ICU.IPR[0x04].BYTE = 0x0f; //CMT0\_CMI0 Priority of interrupts  
 ICU.IER[0x03].BIT.IEN4 = 1; //CMT0\_CMI0 Permission for interrupt  
 CMT.CMSTR0.WORD = 0x0000; //CMT0,CMT1 Stop counting  
 CMT0.CMCR.WORD = 0x00C3; //PCLK/512  
 CMT0.CMCNT = 0;  
 CMT0.CMCOR = 96; //1ms/(1/(49.152MHz/512))  
 CMT.CMSTR0.WORD = 0x0003; //CMT0,CMT1 Start counting  
  
 // MTU3\_3 MTU3\_4 PWM mode synchronized by RESET  
 MSTP\_MTU = 0; //Release module stop state  
 MTU.TSTRA.BYTE = 0x00; //MTU Stop counting  
  
 MTU3.TCR.BYTE = 0x23; //ILCK/64(651.04ns)  
 MTU3.TCNT = MTU4.TCNT = 0; //MTU3,MTU4TCNT clear  
 MTU3.TGRA = MTU3.TGRC = PWM\_CYCLE; //cycle(16ms)  
 MTU3.TGRB = MTU3.TGRD = SERVO\_CENTER; //PWM(servo motor)  
 MTU4.TGRA = MTU4.TGRC = 0; //PWM(left motor)  
 MTU4.TGRB = MTU4.TGRD = 0; //PWM(right motor)  
 MTU.TOCR1A.BYTE = 0x40; //Selection of output level  
 MTU3.TMDR1.BYTE = 0x38; //TGRC,TGRD buffer function  
 //PWM mode synchronized by RESET  
 MTU4.TMDR1.BYTE = 0x00; //Set 0 to exclude MTU3 effects  
 MTU.TOERA.BYTE = 0xc7; //MTU3TGRB,MTU4TGRA,MTU4TGRB permission for output  
  
 MTU.TSTRA.BYTE = 0x40; //MTU0,MTU3 count function  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\* Interrupt \*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
#pragma interrupt Excep\_CMT0\_CMI0(vect=28)  
void Excep\_CMT0\_CMI0(void)  
{  
 cnt0++;  
 cnt1++;  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\* Timer unit \*/  
/\* Arguments: timer value, 1 = 1 ms \*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
void timer( unsigned long timer\_set )  
{  
 cnt0 = 0;  
 while( cnt0 < timer\_set );  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\* Sensor state detection \*/  
/\* Arguments: masked values \*/  
/\* Return values: sensor value \*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
unsigned char sensor\_inp( unsigned char mask )  
{  
 unsigned char sensor;  
  
 sensor = ~PORT4.PORT.BYTE;  
  
 sensor &= mask;  
  
 return sensor;  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\* Read start bar detection sensor \*/  
/\* Return values: Sensor value, ON (bar present):1, \*/  
/\* OFF (no bar present):0 \*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
unsigned char startbar\_get( void )  
{  
 unsigned char b;  
  
 b = ~PORT4.PORT.BIT.B0 & 0x01; /\* Read start bar signal \*/  
  
 return b;  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\* Cross line detection processing \*/  
/\* Return values: 0: no cross line, 1: cross line \*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
int check\_crossline( void )  
{  
 unsigned char b;  
 int ret;  
  
 ret = 0;  
 b = sensor\_inp(MASK3\_3);  
 if( b==0xe7 ) {  
 ret = 1;  
 }  
 return ret;  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\* Right half line detection processing \*/  
/\* Return values: 0: not detected, 1: detected \*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
int check\_rightline( void )  
{  
 unsigned char b;  
 int ret;  
  
 ret = 0;  
 b = sensor\_inp(MASK4\_4);  
 if( b==0x1f ) {  
 ret = 1;  
 }  
 return ret;  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\* Left half line detection processing \*/  
/\* Return values: 0: not detected, 1: detected \*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
int check\_leftline( void )  
{  
 unsigned char b;  
 int ret;  
  
 ret = 0;  
 b = sensor\_inp(MASK4\_4);  
 if( b==0xf8 ) {  
 ret = 1;  
 }  
 return ret;  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\* DIP switch value read \*/  
/\* Return values: Switch value, 0 to 15 \*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
unsigned char dipsw\_get( void )  
{  
 unsigned char sw,d0,d1,d2,d3;  
  
 d0 = ( PORT6.PORT.BIT.B3 & 0x01 ); /\* P63~P60 read \*/  
 d1 = ( PORT6.PORT.BIT.B2 & 0x01 ) << 1;  
 d2 = ( PORT6.PORT.BIT.B1 & 0x01 ) << 2;  
 d3 = ( PORT6.PORT.BIT.B0 & 0x01 ) << 3;  
 sw = d0 | d1 | d2 | d3;  
  
 return sw;  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\* Push-button in MCU board value read \*/  
/\* Return values: Switch value, ON: 1, OFF: 0 \*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
unsigned char buttonsw\_get( void )  
{  
 unsigned char sw;  
  
 sw = ~PORTE.PORT.BIT.B5 & 0x01; /\* Read ports with switches \*/  
  
 return sw;  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\* Push-button in motor drive board value read \*/  
/\* Return values: Switch value, ON: 1, OFF: 0 \*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
unsigned char pushsw\_get( void )  
{  
 unsigned char sw;  
  
 sw = ~PORT7.PORT.BIT.B0 & 0x01; /\* Read ports with switches \*/  
  
 return sw;  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\* LED control in MCU board \*/  
/\* Arguments: Switch value, LED0: bit 0, LED1: bit 1. 0: dark, 1: lit \*/  
/\* \*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
void led\_out\_m( unsigned char led )  
{  
 led = ~led;  
 PORTA.DR.BYTE = led & 0x0f;  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\* LED control in motor drive board \*/  
/\* Arguments: Switch value, LED0: bit 0, LED1: bit 1. 0: dark, 1: lit \*/  
/\* Example: 0x3 -> LED1: ON, LED0: ON, 0x2 -> LED1: ON, LED0: OFF \*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
void led\_out( unsigned char led )  
{  
 led = ~led;  
 PORT7.DR.BIT.B6 = led & 0x01;  
 PORT1.DR.BIT.B0 = ( led >> 1 ) & 0x01;  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\* Motor speed control \*/  
/\* Arguments: Left motor: -100 to 100, Right motor: -100 to 100 \*/  
/\* Here, 0 is stopped, 100 is forward, and -100 is reverse. \*/  
/\* Return value: None \*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
void motor( int accele\_l, int accele\_r )  
{  
 int sw\_data;  
  
 sw\_data = dipsw\_get() + 5;  
 accele\_l = accele\_l \* sw\_data / 20;  
 accele\_r = accele\_r \* sw\_data / 20;  
  
 /\* Left Motor Control \*/  
 if( accele\_l >= 0 ) {  
 PORT7.DR.BYTE &= 0xef;  
 MTU4.TGRC = (long)( PWM\_CYCLE - 1 ) \* accele\_l / 100;  
 } else {  
 PORT7.DR.BYTE |= 0x10;  
 MTU4.TGRC = (long)( PWM\_CYCLE - 1 ) \* ( -accele\_l ) / 100;  
 }  
  
 /\* Right Motor Control \*/  
 if( accele\_r >= 0 ) {  
 PORT7.DR.BYTE &= 0xdf;  
 MTU4.TGRD = (long)( PWM\_CYCLE - 1 ) \* accele\_r / 100;  
 } else {  
 PORT7.DR.BYTE |= 0x20;  
 MTU4.TGRD = (long)( PWM\_CYCLE - 1 ) \* ( -accele\_r ) / 100;  
 }  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\* Servo steering operation \*/  
/\* Arguments: servo operation angle: -90 to 90 \*/  
/\* -90: 90-degree turn to left, 0: straight, \*/  
/\* 90: 90-degree turn to right \*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
void handle( int angle )  
{  
 /\* When the servo move from left to right in reverse, replace "-" with "+". \*/  
 MTU3.TGRD = SERVO\_CENTER - angle \* HANDLE\_STEP;  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\* end of file \*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/