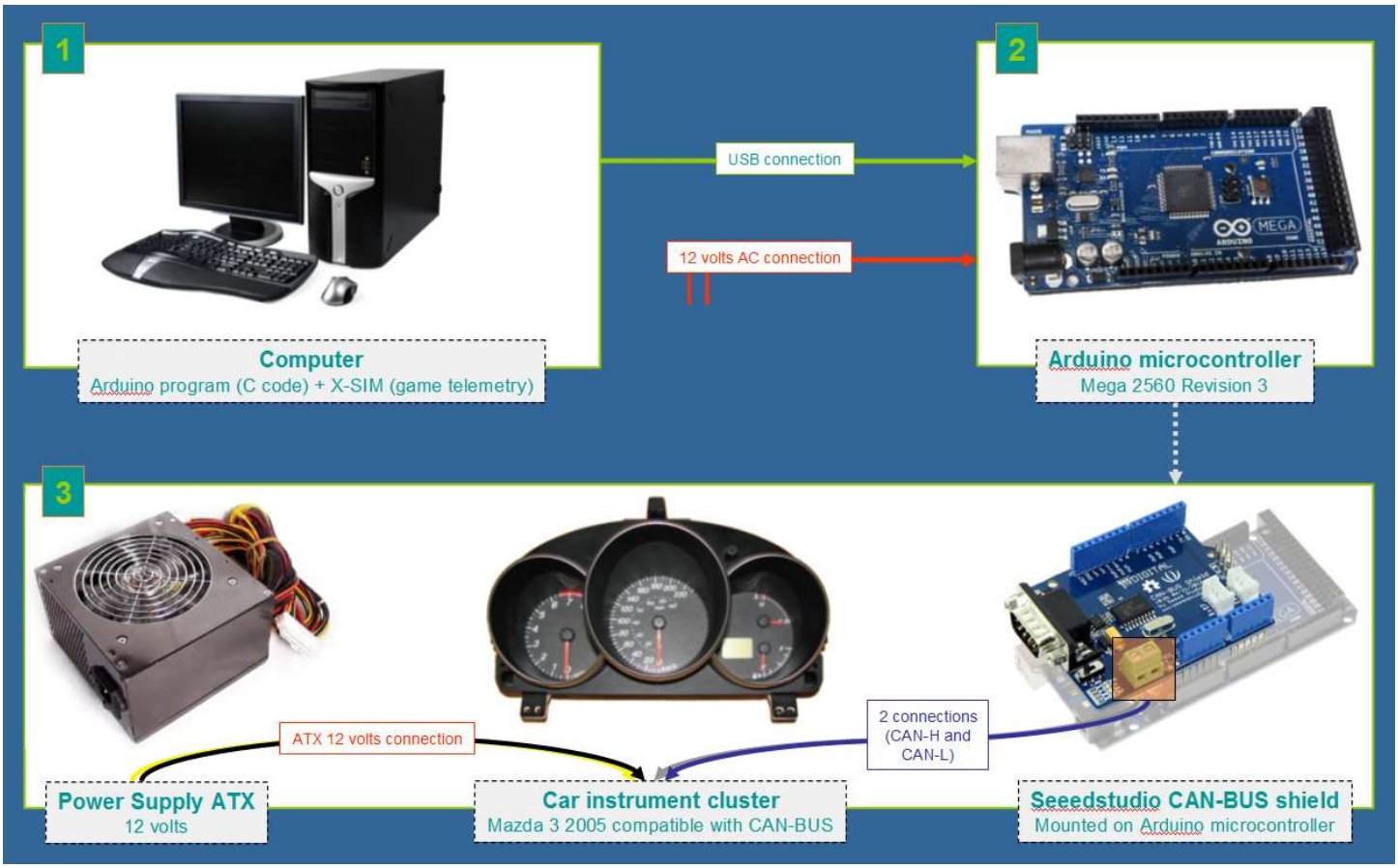
Build a real car instrument cluster for racing simulation on a PC

Components required:

1- PC computer with Windows 7, Windows 8 or Windows 10
2- Arduino Mega 2650 revision 3 micro-controller
3- Seeedstudio CAN-BUS shield
4- 12 volts ATX Power Supply (equipped ideally with a an interrupter)
5- AC/DC power adapter with 1-2 amp 12 volts output
6- Mazda 3 2005 instrument cluster including its two cable harnesses
7- Program like Sim-Tools or X-Sim to feed the micro-controller
8- Arduino program to upload the micro-controller with the proper code

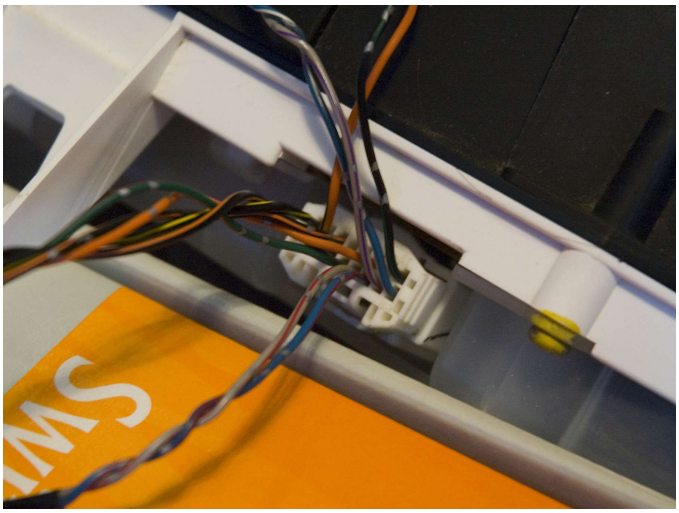
General connection diagram:



Instructions:

The following instructions will guide you through the many steps involved to properly connect the components mentioned earlier. The included pictures will help you to get the right configuration. With the proper game telemetry plugins the cluster should react to all the inputs related to speed, rpm and gear shift.

Step 1



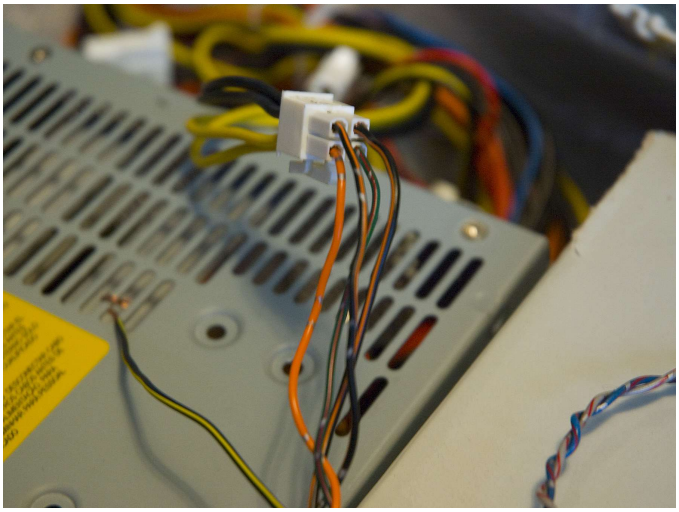
Pick the smaller cable harness, seat it in the proper port of the instrument cluster and isolate the four following wires:

Black/Orange, Black/Orange, Green/Red, Orange, Black/Yellow

Isolate the two Blue/Red and Grey/Red wires and twist them to form a twisted-pair cable on the whole length of the wire.

Remove 2 cm of sheath from the end of each seven cables. Fold in half the bare part of each cable except for the Black/Yellow one.

Step 2

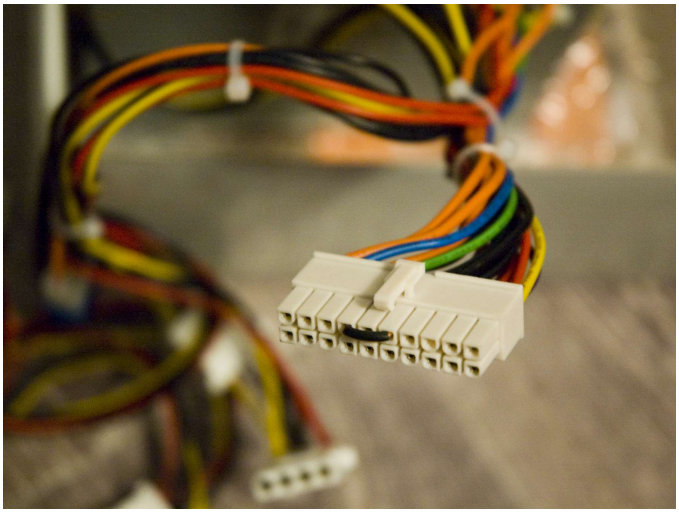


Take the 12 volt connector of the power supply and identify the two black-wired ports and the two yellow-wired ports.

Insert the folded bare ends of the two Black/Orange cables in the two black-wired ports, and both Green/Red and Orange cables in the two yellow-wired ports. Add electric tape around the joined wires to secure the connection.

Attach the bare end of the Black/Yellow cable to a metal part of the power supply for ground.

Step 3

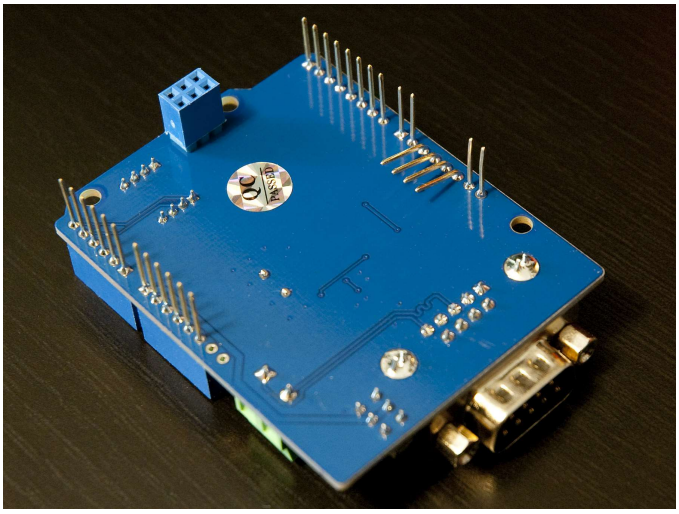


Take the 20-pin connector of the power supply and jump-connect the green and black-wired ports in order to start the power supply as soon as it's connected to a AC wall plug.

If the power supply is equipped with an interrupter that will allow to leave the cluster and the power supply connected to the AC outlet without always being in function.

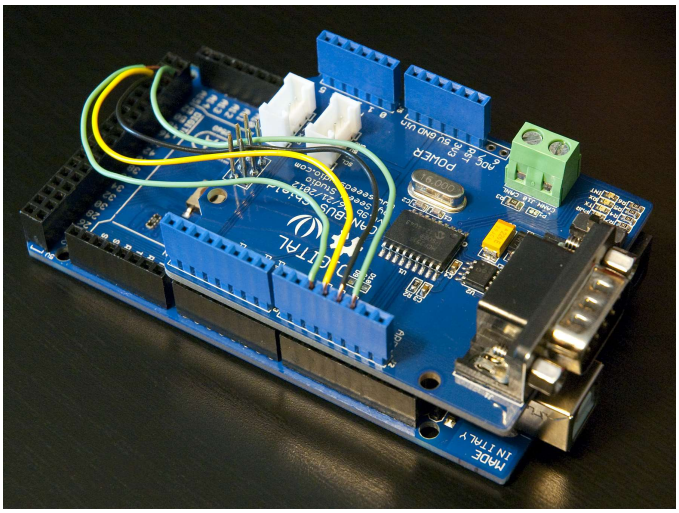
Cover the 20-pin connector with electric tape.

Step 4



Take the CAN-BUS shield and place it upside-down. Locate the 4 pins numbered 10-11-12-13 and bend them delicately with tweezers like depicted in the picture above.

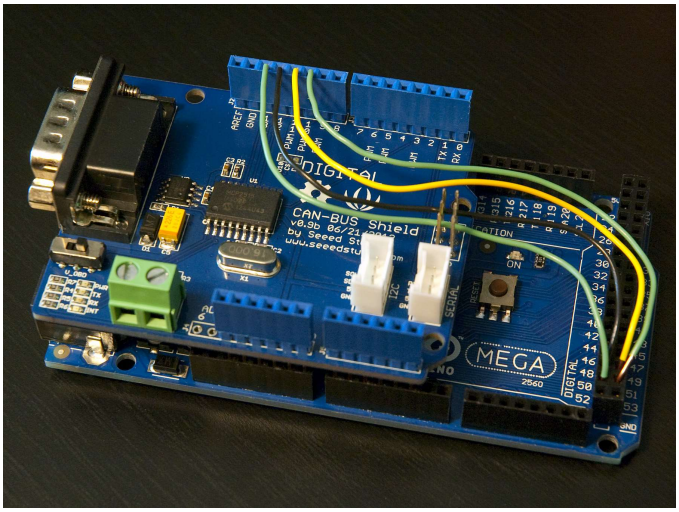
Step 5



Install the CAN-Bus shield on top of the Arduino Mega board.

Prepare four wires of 10 cm each and remove 1 cm of sheath at the tip of each cable. Bend in half the bare ends of the wires.

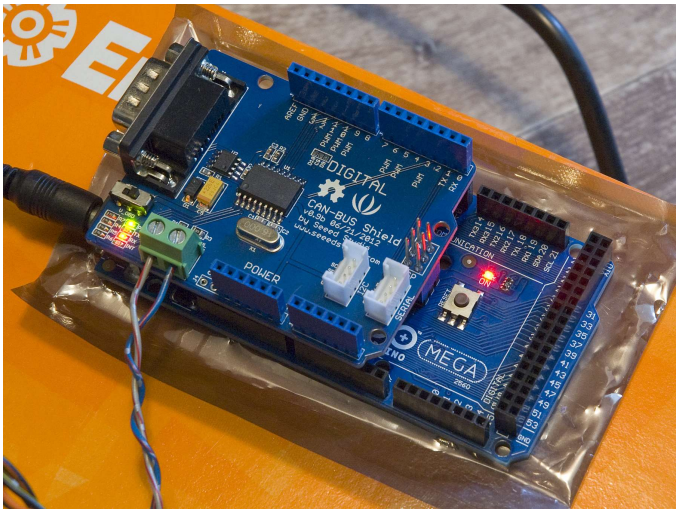
Step 6



Create a special bridge between the CAN-BUS shield and the Arduino by connecting the 4 wires in the following sequence:

CAN-BUS port 10 to Arduino port 51
CAN-BUS port 11 to Arduino port 49
CAN-BUS port 12 to Arduino port 48
CAN-BUS port 13 to Arduino port 50

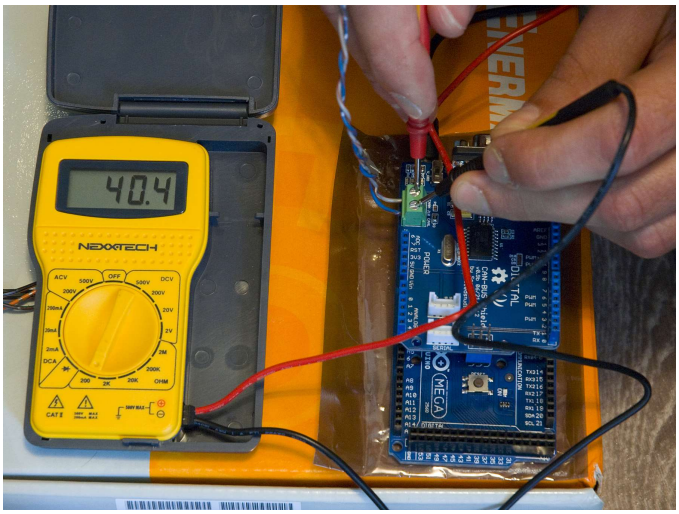
Step 7



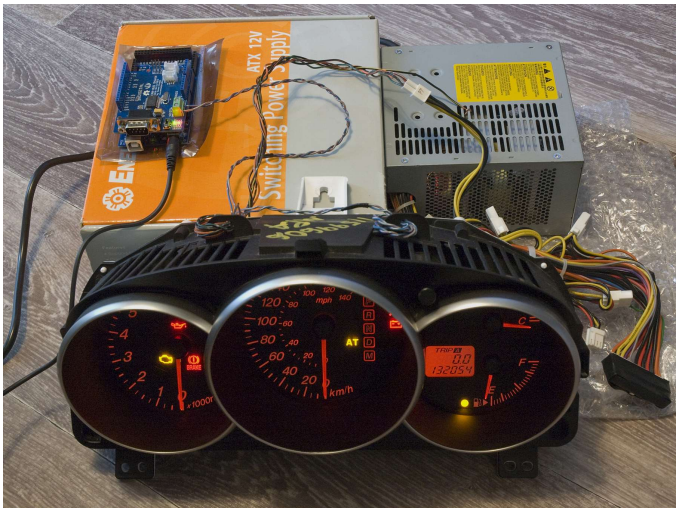
Take the twisted-pair cable and insert the bare end of the Grey/Red wire in the CANH port of the CAN-BUS shield. Insert the bare end of the Blue/Red wire in the CANL port. Secure the wires with the port screws.

Note: The picture above was taken before the installation of the four wires providing the CAN-BUS/Arduino bridge and it is not showing the final configuration. Same goes for the next pictures. Keep in mind that at this stage the four wires should be there.

Step 8



The resistance on the CAN-BUS shield CANH and CANL ports should indicate 40 Omhs.

Step 9	Step 10
	
The final configuration should look like in the picture above once all the wires and components are connected. At this stage you can connect the Arduino board to a USB port of the PC and to a wall outlet using the AC/DC 12 volt power adapter.	Now turn on the power supply and verify that the cluster is doing is starting routine. The needles should go all the way up and then down.

The next steps consist in configuring the Arduino micro-controller to make it work with the pc platform:

Step 1:

Download the latest Arduino driver for the COM3 port (Windows 10 should already have the driver pre-installed):
http://www.driverscape.com/download/arduino-mega-2560-r3-%28com3%29

Visit the Arduino website for more info about initial configuration and to download the software package:
https://www.arduino.cc/en/Guide/ArduinoMega2560
https://www.arduino.cc/en/Main/Software

Step 2:

Download the library for the CAN-BUS shield:
http://wiki.seeed.cc/CAN-BUS_Shield_V1.2/

Step 3:

Follow all the instructions in the above websites to properly configure the Arduino/CAN-BUS boards and install the software/libraries onto your PC.

Step 4:

Download the Arduino sketch from this OneDrive folder and upload it to your micro-controller.
https://1drv.ms/u/s!AtwKGzBk3VOoitITk6T_Rhp0ycaEPA

You can also copy the code directly from the next page, paste it in a new sketch and upload to the board.

This code will send SPEED, RPM and GEAR data to the dashboard and the TM1638 digital display. It will also actuate the TEMP clock needle to its middle position but that indicator is not based on game telemetry. It is purely cosmetic.

```
#include <mcp_can.h>
#include <mcp_can_dfs.h>
#include <SPI.h>
#include <TM1638.h> // can be downloaded from http://code.google.com/p/tm1638-library/

const int SPI_CS_PIN;

// -- Set CS pin -- //
MCP_CAN CAN(SPI_CS_PIN);

// -- Declare RPM, Speed, Gear and Temp -- //
int RPM;
int Speed;
int Gear;
int Temp;

// -- LedState used to set the LED -- //
int ledState = LOW;
long previousMillis = 0; // will store last time LED was updated

unsigned long currentMillis = millis();
unsigned int CarRPM;
unsigned int CarSpeedKM;

// -- Interval at which to blink (milliseconds) -- //
long interval = 40;

// -- Associate a LED with its respective pin and determine its brightness -- //
int ShiftLED = A0;
int ShiftLEDbrightness = 255;

// -- Associate TM1638 LED display with its respective data pins -- //
TM1638 module1(8, 7, 9);

char kind_of_data;
byte my_data[8];

void setup()
{
  Serial.begin(115200);

  // -- Send a diagnostic test to the Serial Monitor for the CAN shield -- //
  if (CAN.begin(CAN_500KBPS) == CAN_OK) Serial.print("can init ok!!\r\n");
  else Serial.print("Can init fail!!\r\n");

  // -- Initialize the LED pin for Shift LED -- //
  pinMode(ShiftLED, OUTPUT);

  // -- Initialize the TM1638 display -- //
  module1.clearDisplay(); // clears the display from garbage if any
  String name1 = "HELLO jj"; // sets a custom logo start up banner
  module1.setDisplayToString(name1); // prints the banner

  module1.setLEDs(0b10000000 | 0b00000001 << 8 );
  delay(50);
  module1.setLEDs(0b11000000 | 0b00000011 << 8 );
  delay(50);
  module1.setLEDs(0b11100000 | 0b00000111 << 8 );
  delay(50);
  module1.setLEDs(0b11110000 | 0b00001111 << 8 );
  delay(50);
  module1.setLEDs(0b11111000 | 0b00011111 << 8 );
  delay(50);
  module1.setLEDs(0b11111100 | 0b00111111 << 8 );
  delay(50);
  module1.setLEDs(0b11111110 | 0b01111111 << 8 );
  delay(50);
  module1.setLEDs(0b11111111 | 0b11111111 << 8 );
  delay(200);
  module1.setLEDs(0b00000000 | 0b00000000 << 8 );
  delay(50);
  module1.setLEDs(0b11111111 | 0b11111111 << 8 );
  delay(100);
  module1.setLEDs(0b00000000 | 0b00000000 << 8 );
  delay(50);
  module1.setLEDs(0b11111111 | 0b11111111 << 8 );
  delay(100);
  module1.setLEDs(0b00000000 | 0b00000000 << 8 );
  delay(50);
  module1.setLEDs(0b11111111 | 0b11111111 << 8 );
  delay(100);
  module1.setLEDs(0b00000000 | 0b00000000 << 8 );
  delay(2000); // small delay 2 sec
  module1.clearDisplay(); // clears the 1st display
}

void loop()
{
  int i;

  // -- Holds all serial data into a array -- //
  char bufferArray[20];

  // -- Holds gear value data -- //
  byte gear;

  // -- Marker that new data are available -- //
  byte geardata = 0;

  // -- If 6 bytes available in the Serial buffer -- //
  if (Serial.available() >= 9) {
    for (i = 0; i < 9; i++) { // for each byte...
      bufferArray[i] = Serial.read(); // put into array
    }
  }

  if (bufferArray[7] == 'G' ) {
    gear = bufferArray[8]; // retrieves the single byte of gear (0-255 value)
    geardata = 1; // we got new data!
  }

  if (geardata == 1) {
    char* neutral = "n"; // sets the character for neutral
    char* reverse = "r"; // sets the character for reverse

    if (gear >= 1 and gear < 10 ) {
      module1.setDisplayDigit(gear, 0, false); // displays numerical value of the current gear
    }
    if (gear == 0) {
      module1.setDisplayToString(neutral, 0, 0); // displays the character for neutral
    }
    if (gear == 255) { // -1 that represents reverse rollover to 255 so...
      module1.setDisplayToString(reverse, 0, 0); // displays the character for reverse
    }
    geardata = 0;
  }

  // -- Parsers used by Simtools to send game datas for Gear, RPM, Speed, Temperature and the external LEDs to the cluster over serial port (USB) -- //
  while (Serial.available() < 1)
  {
    // -- Send Gear data to the cluster -- //
    kind_of_data = Serial.read(); // kind_of_data = Serial.read();
    if (kind_of_data == 'G' ) Read_Gear();

    // -- Send Temp data to the cluster -- //
    kind_of_data = Serial.read();
    if (kind_of_data == 'T' ) Read_Temp();

    // -- Send RPM data to the cluster and flash RPM red LED when RPM hits 6500 -- //
    kind_of_data = Serial.read();
    if (kind_of_data == 'R' ) Read_RPM();

    // -- Send Speed data to the cluster -- //
    kind_of_data = Serial.read();
    if (kind_of_data == 'S' ) Read_Speed();
  }
}

// -- Temp Data and Maths -- //
void Read_Temp()
{
  // -- Read from serial -- //
  delay(1);
  int Temp100 = Serial.read() - '0';
  delay(1);
  int Temp10 = Serial.read() - '0';
  delay(1);
  int Temp1 = Serial.read() - '0';

  Temp = 100 * Temp100 + 10 * Temp10 + Temp1;

  my_data[0] = 0x98; // Temp Gauge Data
  my_data[1] = 0x00;

  CAN.sendMsgBuf(0x420, 0, 8, my_data); // send Message
}

// -- Gear Data and Maths -- //
void Read_Gear()
{
  delay(1);
  int Gear100 = Serial.read() - '0';
  delay(1);
  int Gear10 = Serial.read() - '0';
  delay(1);
  int Gear1 = Serial.read() - '0';

  Gear = 100 * Gear100 + 10 * Gear10 + Gear1;

  Gear = map(Gear, 127, 255, 0, 7);

  if ( Gear == 0 )
  {
    // -- Show 'N' (Neutral) on dashboard -- //
    my_data[2] = 0x3; // Gear: Neutral
    my_data[3] = 0xFF;

    CAN.sendMsgBuf(0x228, 0, 8, my_data); // send Message
  }
}

// -- RPM Data and Maths -- //
void Read_RPM()
{
  // -- Read from serial -- //
  delay(1);
  int RPM100 = Serial.read() - '0';
  delay(1);
  int RPM10 = Serial.read() - '0';
  delay(1);
  int RPM1 = Serial.read() - '0';
  int RPM = 100 * RPM100 + 10 * RPM10 + RPM1;

  // -- Set values -- //
  my_data[0] = (RPM * 4) / 256; // rpm
  my_data[1] = (RPM * 4) % 256; // rpm

  CAN.sendMsgBuf(0x201, 0, 8, my_data); // send Message
  delay(60);

  // -- Engine and battery lights will turn ON when engine has stopped -- //
  if (RPM < 100)
  {
    my_data[2] = 0x00; // check engine light + coolant, oil and battery ON

    CAN.sendMsgBuf(0x420, 0, 8, my_data); // send Message
  }

  // -- Engine and battery lights will turn off when engine has started -- //
  if (RPM > 100)
  {
    my_data[3] = 0xFF; // check engine light + coolant, oil and battery OFF
  }

  // -- Fuel gauge needle goes up when engine is off (fake value not based on game telemetry) -- //
  if (RPM < 300) // 0RPM
  {
    // -- Initialize the pin 2 to drive the fuel gauge needle up -- //
    pinMode(2, OUTPUT); // pin to which the fuel gauge is connected
  }

  // -- Fuel gauge needle goes down when engine is on (fake value not based on game telemetry) -- //
  if (RPM > 300) // 1000RPM
  {
    // -- Initialize and empty pin to drive the fuel gauge needle down -- //
    pinMode(4, OUTPUT); // empty pin
  }

  // -- RPM LEDs will turn on at increasing RPM and Fuel Gauge needle goes up when engine is off -- //
  if (RPM < 100) // 0RPM
  {
    module1.setLEDs(0b00000000 | 0b00000000); // All TM1638 LED off
  }
  if (RPM >= 101 && RPM <= 400) // 1000RPM
  {
    module1.setLEDs(0b00000001 | 0b00000000); // TM1638 Green LED 1 on
  }
  if (RPM >= 401 && RPM <= 700) // 2000RPM
  {
    module1.setLEDs(0b00000011 | 0b00000000 << 8 ); // TM1638 Green LED 1 and 2 on
  }
  if (RPM >= 701 && RPM <= 1000) // 3000RPM
  {
    module1.setLEDs(0b00000111 | 0b00000000 << 8 ); // TM1638 Green LED 1, 2 and 3 on
  }
  if (RPM >= 1001 && RPM <= 1200) // 4000RPM
  {
    module1.setLEDs(0b00001111 | 0b00000000 << 8 ); // TM1638 Green LED 1, 2, 3 and 4 on
  }
  if (RPM >= 1201 && RPM <= 1450) // 5000RPM
  {
    module1.setLEDs(0b00011111 | 0b00010000 << 8 ); // TM1638 Green LED 1, 2, 3, 4 and Red LED 1 on
  }
  if (RPM >= 1451 && RPM <= 1600) // 6000RPM
  {
    module1.setLEDs(0b00111111 | 0b00110000 << 8 ); // TM1638 Green LED 1, 2, 3, 4 and Red LED 1, 2 on
  }
  if (RPM >= 1601 && RPM <= 1940) // 7000RPM
  {
    module1.setLEDs(0b01111111 | 0b01110000 << 8 ); // TM1638 Green LED 1, 2, 3, 4 and Red LED 1, 2, 3 on
  }
  if (RPM >= 1941 && RPM <= 2200) // 8000RPM
  {
    module1.setLEDs(0b11111111 | 0b11110000 << 8 ); // TM1638 Green LED 1, 2, 3, 4 and Red LED 1, 2, 3 on
  }

  // -- RPM Red LED will turn on at 6500 RPM -- //
  if (RPM > 1600) analogWrite(ShiftLED, ShiftLEDbrightness); // turn the LED on
  if (RPM < 1600) analogWrite(ShiftLED, LOW);  // turn the LED off
}

// -- Speed Data and Maths -- //
void Read_Speed()
{
  // -- Read from Serial -- //
  delay(1);
  int Speed100 = Serial.read() - '0';
  delay(1);
  int Speed10 = Serial.read() - '0';
  delay(1);
  int Speed1 = Serial.read() - '0';
  int Speed = 100 * Speed100 + 10 * Speed10 + Speed1;

  // -- Set values -- //
  my_data[4] = (Speed + 1000) / 256; // speed
  my_data[5] = (Speed + 1000) % 256; // speed
  my_data[6] = 0x00;
  my_data[7] = 0x00;

  CAN.sendMsgBuf(0x201, 0, 8, my_data); // send Message
  delay(40);

  // -- Show digital speed on TM1638 display -- //
  CarSpeedKM = Speed / 100 + 10; // calculation accounting for Simtools values used with the Mazda dashboard
  module1.setDisplayToDecNumber(CarSpeedKM, 8, false);
}
```

Step 5:

In order to add the Gamedash functionnality to your XSimulator environment you need to upgrade to the Pro version of SImtools. When done, download and install the pro version of the program and you will have a new component which is called GameDash Command Editor.

Enter the following commands into Gamedash Command Editor boxes:

SPEED :

MATH * 1
PERCENT -34 220
SCALE 0 255
ROUND 0
TOCHR

RPM :
MATH + 1200
MATH * 1.20
PERCENT -20000 100000
SCALE 0 255
ROUND 0
TOCHR

GEAR :

MATH - 1
REPLACE -1 255
TOCHR

Also add these parsers in the output field:

S<Dash1>R<Dash2>G<Dash3>T<Dash4> Output Rate: 1ms

In the Output Type, choose 115200 (BitsPerSec), 8 (Data Bits), None (Parity) and 1 (Stop Bits)

These commands were tested with Assetto Corsa but they should also work with or without minor tweaks in other racing games that support gamedash inputs.

Step 6:

Make sure that all the electronic components are powered and the micro-controller is plugged to a USB port of the pc. Then launch Simtools GameEngine, GameManager and GameDash. Launch the game and once the race map has loaded the tachometer clock needle should indicated the RPM value of the car in realtime.
Then test the SPEED and GEAR.

OneDrive link to the complete documentation and instructions in PDF format, hi-res photos, Arduino sketch and Simtools configuration: https://1drv.ms/f/s!AtwKGzBk3VOohJdX5ZtdmwFRk_GPcw