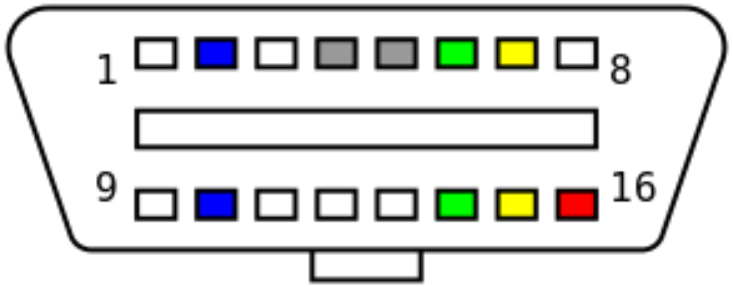
ELM327 OBD II

An overview of the ELM327 microcontroller

Marco Cuoci

OBD II / EOBD Bus

- OBD stands for On Board Diagnostics
- The OBD bus interconnects sensors and control systems inside the car
- By law, the OBD must cover all emission relevant components
- According to the SAE J1962 standard all OBD compliant vehicles must have a standard connector near the driver's seat
- Usually, queries on the bus can just fetch information (no modify)



ELM327 Microcontroller

- Provides a level of abstraction from the underlying OBD II protocol
  - Hexadecimal queries and responses are mapped to strings
- Connects directly to the car's OBD connector
- Presents information via a UART interface
  - Usually sent to a Bluetooth or Wi-Fi antenna

- Supports a plethora of protocols for the underlying OBD system

ELM327 OBD Interface

- Queries on the OBD bus are structured hexadecimal values
- One byte to identify the MODE

  Ex 01 is current, 03 is to request trouble codes…
- One to three bytes to identify the PID (Parameter ID)

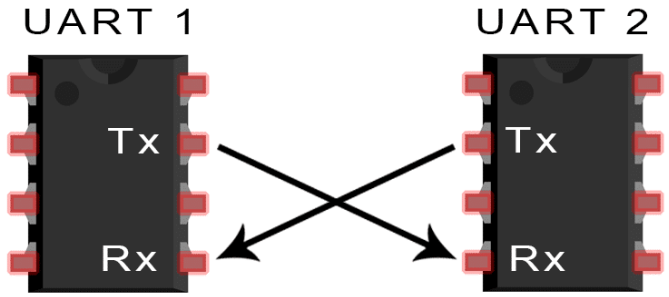
  As defined in this table

Supported OBD protocols

- SAE J1850 PWM (41.6 kbit/s)
- SAE J1850 VPW (10.4 kbit/s)
- ISO 9141-2 (5 baud init, 10.4 kbit/s)
- ISO 14230-4 KWP (5 baud init, 10.4 kbit/s)
- ISO 14230-4 KWP (fast init, 10.4 kbit/s)
- ISO 15765-4 CAN (11 bit ID, 500 kbit/s)
- ISO 15765-4 CAN (29 bit ID, 500 kbit/s)
- ISO 15765-4 CAN (11 bit ID, 250 kbit/s)
- ISO 15765-4 CAN (29 bit ID, 250 kbit/s)
- SAE J1939 (250kbit/s)
- SAE J1939 (500kbit/s)

ELM327 UART Interface

- The ELM 327 can be accessed by UART at baud rates 9600 or 38400
  Fortunately for us, usually the UART interface is wired to an antenna
- The microcontroller acts as an interpreter
  Strings sent via the UART interface are converted into hexadecimal data packets
- It can be programmed to do various tasks by issuing an “AT” command
  Any query starting with “AT” is directed to the microcontroller, otherwise it is assumed to be a query for the OBD bus



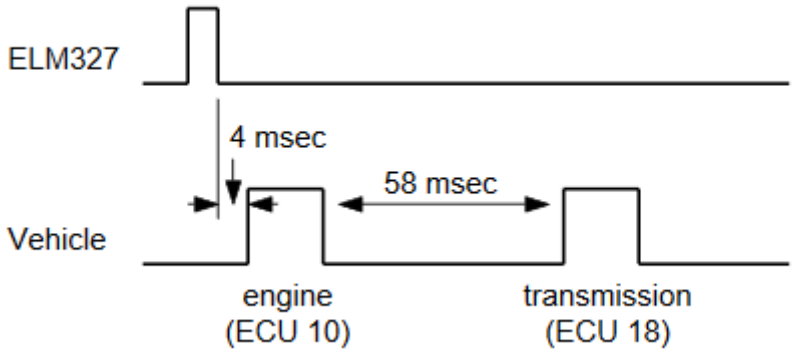
Querying the ELM327

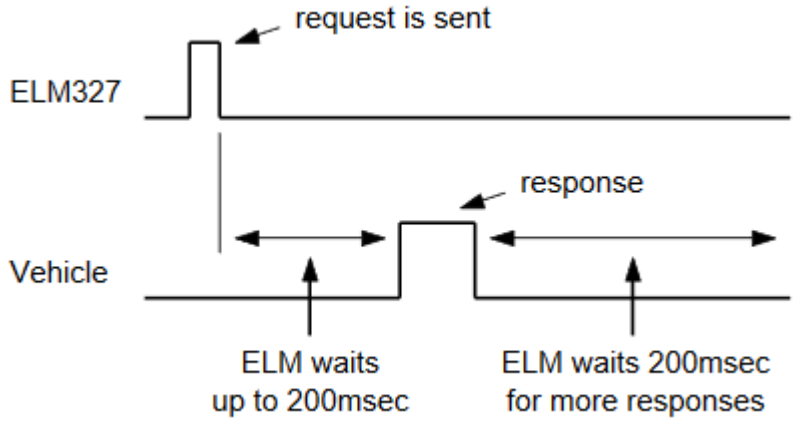
- In order to query the ELM327, an appropriate instruction set must be used
  - There is an AT instruction set for commands directed at the ELM
  - As well as an OBD instruction set for commands meant for the OBD bus
- Commands are to be sent as strings, terminated by a carriage return
  - Spaces are ignored
- The command is interpreted and routed to its destination according to the negotiated protocol

Querying the ELM327

- When the microcontroller receives a response from the OBD bus, it will wait approx. 200ms before returning
  - Diagnostic (mode 03-09) queries involve multiple components
  - The time can be cut to near-zero by issuing the “AT AT2” command

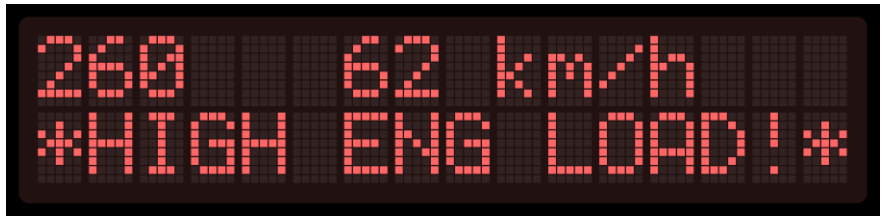
- The value is then returned as a string via the UART interface

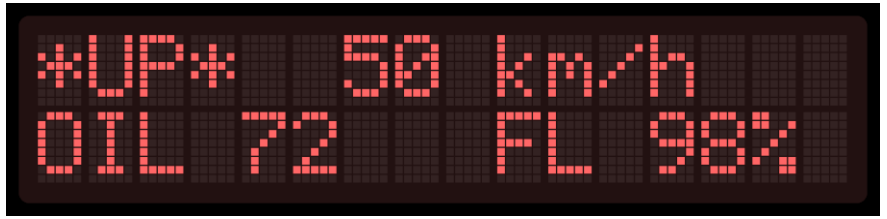


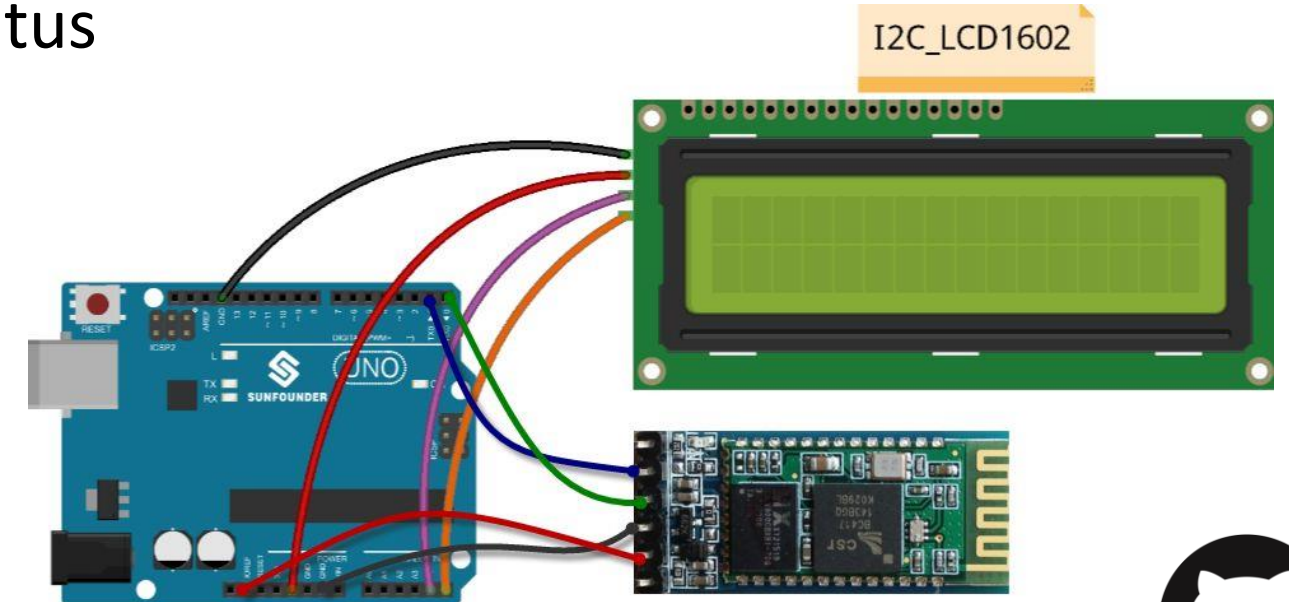


A practical example: Overview

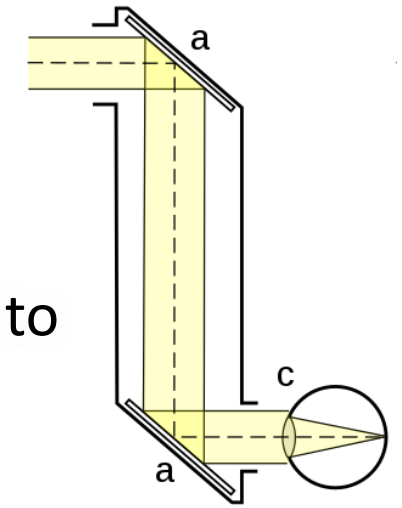
- Use an Arduino to communicate over Bluetooth to the ELM327 device and project real time data to a transparent glass piece using a LCD
- The Arduino should make contextual decisions on what to display according to the vehicle's status







A practical example: Hardware Issues



- First problem: How do mirrors work

  Mirrors reverse the image, if we were to project directly from the LCD to the glass piece, we would have an inverted view of the values

  Solution: Use two mirrors to reverse the image twice

- Second problem: Apparently having loose cables on the dashboard of a car is not a good idea

  During testing, poor road condition was enough for jumper wires to come off the breadboard, breaking the circuit and crashing the software

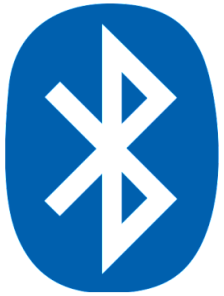
  Solution: 50% duct tape, 50% good *capable* friends

A practical example: Documentation

- Three sets of documentations were used for the project
  - OBD Instruction Set from the Wikipedia page
  - ELM Instruction Set from Sparkfun site
  - ELM Datasheet from ELM electronics site
- Two header files containing definitions for the needed commands were then produced following the instructions from these documents
  - ELM327_defines.h
  - OBD_defines.h

```
//M101
#pragma once

//OBD INTERFACE HEX VALUES FOR QUERIES
//Refer to https://en.wikipedia.org/wiki/OBD-II_PIDs#Mode_01

//USED TO CHECK FOR COMMAND AVAILABILITY
#define OBD_PIDS_A  "00"        //RETURNS 4B Bitmask
#define OBD_PIDS_B  "20"        //RETURNS 4B Bitmask
#define OBD_PIDS_C  "40"        //RETURNS 4B Bitmask

//A BLOCK
#define OBD_LOAD    "04"        //RETURNS 1B percent (100/255)
#define OBD_COOLANT "05"        //RETURNS 1B celsius +40
#define OBD_RPM     "0C"        //RETURNS 2B 4*rpm ((256A+B)/4)
#define OBD_SPEED   "0D"        //RETURNS 1B km/h
#define OBD_THROTTLE    "11"    //RETURNS 1B percent (100/255)

//B BLOCK
#define OBD_FUEL_LEVEL  "2F"    //RETURNS 1B percent (100/255)

//C BLOCK
#define OBD_OIL_TEMP    "5C"    //RETURNS 1B celsius +40
```

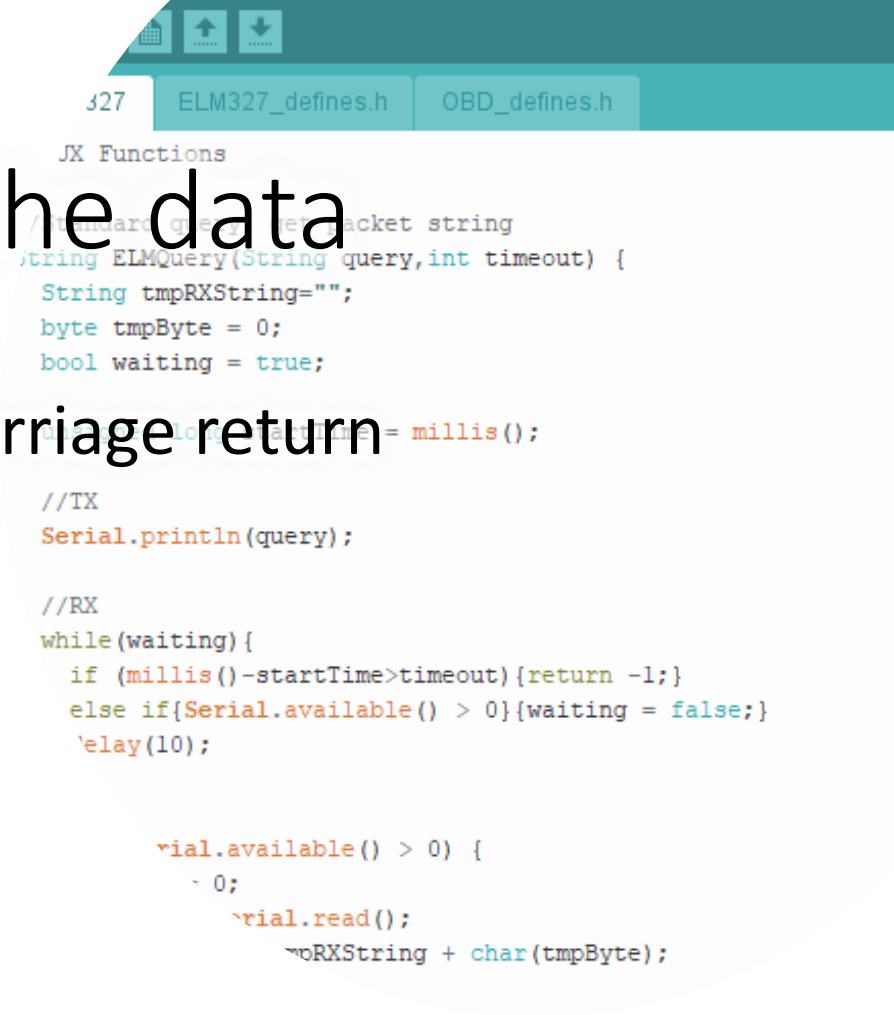
A practical example: Bluetooth Pairing



- For this project the HC-05 Bluetooth module was used
- The Arduino acts as Master, while the ELM327 acts as a slave
- Pairing the HC-05 to the ELM327 requires booting the HC-05 into command mode and sending “AT” commands from a computer

A practical example: Encoding the data

- The ELM327 accepts strings terminated by a carriage return
- Arduino compilers support C++
- We can make use of the Arduino String library
  - Handled automatically
  - Concatenation is an easy supported operation (+)
- Strings are then sent over the Bluetooth interface via the Wire library
  - Use Wire.println() function for an automatic carriage return at the end



A practical example: Decoding the data

- Data is sent from the ELM327 to the Arduino in string format
- The original command is sent back too as an acknowledgment
- Unnecessary data needs to be cleaned from the response
  - Use String.replace()
- The string is a hexadecimal number
  - Use strtol() to convert to a long decimal from a hexadecimal string
- Beware! Different commands have different response lengths!

A practical example: Encoder/Decoder Code

- Transmission is easy
- In order to receive, the data needs to be ready
- The device returns one byte per call
- Output may include previous command or error codes

```
//Standard query, get packet string
String ELMQuery(String query,int timeout) {
  String tmpRXString="";
  byte tmpByte = 0;
  bool waiting = true;

  unsigned long startTime = millis();

  //TX
  Serial.println(query);

  //RX
  while(waiting){
    if (millis()-startTime>timeout){return -1;}
    else if{Serial.available() > 0}{waiting = false;}
    delay(10);
  }

  while(Serial.available() > 0) {
    tmpByte = 0;
    tmpByte = Serial.read();
    tmpRXString = tmpRXString + char(tmpByte);
  }

  tmpRXString.replace(command,"");
  tmpRXString.replace(" ","");
  tmpRXString.replace("OK","");

  //Errors on the bus mapped to empty string
  tmpRXString.replace("STOPPED","");
  tmpRXString.replace("SEARCHING","");
  tmpRXString.replace("NO DATA","");
  tmpRXString.replace("?","");
  tmpRXString.replace(",","");

  return tmpRXString;
}
```

A practical example: OBD Value Formatting

- OBD values are encoded according to well defined formulas

  Temperatures are shifted 40 degrees Celsius up in order to avoid having a sign bit

  RPM is calculated with a .25 granularity

```
//Value formatting according to OBD formulas
long HandleELMMessage(long payload, String query){
  long res = 0;
  switch(query){
    case OBD_SPEED:
      res = payload
    break;
    case OBD_RPM:
      res = payload/4;
    break;
    case OBD_FUEL_LEVEL:
    case OBD_LOAD:
      res = payload*100/255;
    break;
    case OBD_COOLANT:
    case OBD_OIL_TEMP:
      res = payload-40;
    break;
  }
  return res;
}
```

A practical example: Program Logic

- We want a real time device
  - Poll only few devices at a time
- Some data don’t change frequently
  - Temperatures and fuel
- Query different devices with different frequencies

```
//Print speed and advise if gear shift is necessary
case 4:
  resultNMBR = ELMMessagePayload(ELMQuery(OBD_SPEED,1000),OBD_SPEED);
  if (resultNMBR < 10){
    lcd.noDisplay();
    break;
  } else {
    lcd.display();
  }
  resultSTR = StringifyELMMEssage(resultNMBR,OBD_SPEED);
  lcd.setCursor(6,0);
  lcd.print(resultSTR+"  ");
  //
  resultNMBR = ELMMessagePayload(ELMQuery(OBD_RPM,1000),OBD_RPM);
  lcd.setCursor(0,0);
  lcd.print("    ");
  if (resultNMBR > RPM_MAX){
    lcd.setCursor(0,0);
    lcd.print("*UP*");
  } else if (resultNMBR < RPM_MIN){
    lcd.setCursor(0,0);
    lcd.print("*DN*");
  } else {
    resultSTR = StringifyELMMEssage(resultNMBR,OBD_RPM);
    lcd.setCursor(0,1);
    lcd.print(resultSTR.substring(1,3));
  }
  state = 5;
break;
```

Thanks for your attention