**审定成绩：**

|  |  |  |  |
| --- | --- | --- | --- |
| **重庆邮电大学**  **毕业设计（论文）** | | | |
|  | | | |
|  | **中文题目** | **一种智能OBD解析器设计** |  |
|  |  |
| **英文题目** | **Design of an Intelligent OBD Parser** |
|  |  |
| **学院名称** | **自动化学院工业互联网学院** |
| **学生姓名** |  |
| **专 业** | **自动化** |
| **班 级** |  |
| **学 号** |  |
| **指导教师** | **教授** |
| **答 辩 组**  **负 责 人** | **教授** |
|  | |

**2020年6月**

**重庆邮电大学教务处制**

自动化学院工业互联网学院本科毕业设计(论文)诚信承诺书

本人郑重承诺：

我向学院呈交的论文《 一种智能OBD解析器设计 》，是本人在指导教师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明并致谢。本人完全意识到本声明的法律结果由本人承担。

年级

专业

班级

承诺人签名

年 月 日

学位论文版权使用授权书

本人完全了解重庆邮电大学有权保留、使用学位论文纸质版和电子版的规定，即学校有权向国家有关部门或机构送交论文，允许论文被查阅和借阅等。本人授权重庆邮电大学可以公布本学位论文的全部或部分内容，可编入有关数据库或信息系统进行检索、分析或评价，可以采用影印、缩印、扫描或拷贝等复制手段保存、汇编本学位论文。

（注：保密的学位论文在解密后适用本授权书。）

|  |  |
| --- | --- |
| 学生签名： | 指导老师签名： |
| 日期： 年 月 日 | 日期： 年 月 日 |

# 摘要

OBD(On-Board Diagnostic)即“车载自动诊断系统”，是车辆故障信息采集的一种标准，通过该技术可以实时获取车辆的工作状态信息，对这些信息进行分析处理可以为车辆安全驾驶或者“智慧交通”提供信息服务。论文主要针对车辆驾乘安全信息服务的需要，设计一种支持OBD协议的智能OBD解析器，通过该OBD解析器可以实时采集车辆信息，同时也可以接受远程的查询指令访问指定的车身模块信息。

论文通过分析目前OBD主流解析器各自特点，采用EST527-miniS模组作为OBD接口芯片设计OBD解析器，本解析器主芯片采用ESP32-WROOM-32最小系统开发板，解析器采用了蓝牙或者Wi-Fi的连接方式把数据发送到上位机或者实现远程传输功能。而且上位机或者远程端可以对本智能OBD解析器进行控制，读取特定的传感器数据，故障码信息，也能执行清除故障码的操作。使用的MQTT协议可以轻松实现追踪和管理超大量级的车辆的工作。

本次设计的智能OBD解析器实现了传统的OBD-II设备在新时代的蜕变。通过实体的测试，验证了本次设计的可行性。

**关键词：**OBD-II，蓝牙，Wi-Fi，MQTT，远程传输

Abstract

OBD (On-Board Diagnostic) is a standard for vehicle fault information collection. the working status information of the vehicle can be obtained in real time by OBD interface. Analysis and processing of this information can be used for vehicle safe driving or "smart transportation "Provide information services. The thesis mainly aims at the needs of vehicle driving safety information services, and designs an intelligent OBD parser that supports the OBD protocol. The OBD parser can collect the vehicle information in real time, and can also receive the remote query instruction to access the specified vehicle module information

The paper analyzes the characteristics of the current OBD mainstream parsers, uses the EST527-miniS module as the OBD interface chip to design the OBD parser, the main chip of the parser uses the ESP32-WROOM-32 minimum system development board, and the parser designs Bluetooth or Wi-Fi. The connection method sends the data to the host computer or realizes the remote transmission function. And the host computer or remote end can control the intelligent OBD parser, read specific sensor data, fault code information, and can also perform the operation of clearing the fault code. The MQTT protocol used can easily achieve the task of tracking and managing a large number of vehicles.

The intelligent OBD parser designed this time has realized the transformation of traditional OBD-II equipment in a new era. Through the physical test, the feasibility of this design was verified.

**Keywords**: OBD-II, Bluetooth, Wi-Fi, MQTT, Remote Transmission

# 目录

[第1章 引言 1](#_Toc42277563)

[1.1 研究背景和意义 1](#_Toc42277564)

[1.2 国内外研究现状 2](#_Toc42277565)

[1.3 主要内容和工作安排 3](#_Toc42277566)

[第2章 方案论证与可行性分析 5](#_Toc42277567)

[2.1 设计要求 5](#_Toc42277568)

[2.2 方案论证 5](#_Toc42277569)

[2.2.1 主控芯片 7](#_Toc42277570)

[2.2.2 蓝牙通信模组 9](#_Toc42277571)

[2.2.3 Wi-Fi通信模组 10](#_Toc42277572)

[2.2.4 OBD协议芯片 11](#_Toc42277573)

[2.3 硬件系统架构方案 12](#_Toc42277574)

[2.4 总结 14](#_Toc42277575)

[第3章 智能OBD解析器硬件设计 15](#_Toc42277576)

[3.1 总体设计 15](#_Toc42277577)

[3.2 OBD协议芯片 16](#_Toc42277578)

[3.3 蓝牙通信模块 18](#_Toc42277579)

[3.4 主控芯片与Wi-Fi模组 20](#_Toc42277580)

[3.5 OBD接口 21](#_Toc42277581)

[3.6 电源芯片 22](#_Toc42277582)

[3.7 本章总结 23](#_Toc42277583)

[第4章 智能OBD解析器软件设计 25](#_Toc42277584)

[4.1 主控芯片开发环境搭建 25](#_Toc42277585)

[4.2 Wi-Fi通信协议选择 26](#_Toc42277586)

[4.3 下位机工作流程 29](#_Toc42277587)

[4.4 上位机开发 31](#_Toc42277588)

[4.5 本章总结 36](#_Toc42277589)

[第5章 系统测试 37](#_Toc42277590)

[5.1 硬件平台测试 37](#_Toc42277591)

[5.1.1 印刷电路板测试 37](#_Toc42277592)

[5.1.2 EST527-miniS测试 38](#_Toc42277593)

[5.1.3 实物平台测试 39](#_Toc42277594)

[5.2 通信测试 40](#_Toc42277595)

[5.2.1蓝牙双向通信测试 40](#_Toc42277596)

[5.2.2 Wi-Fi双向通信测试 41](#_Toc42277597)

[5.3 功能测试 42](#_Toc42277598)

[5.3.1 使用蓝牙通信时的功能测试 42](#_Toc42277599)

[5.3.2 使用Wi-Fi通信时的功能测试 45](#_Toc42277600)

[5.4 本章总结 47](#_Toc42277601)

[第6章 总结与展望 49](#_Toc42277602)

[6.1 总结 49](#_Toc42277603)

[6.2 展望 50](#_Toc42277604)

[参考文献 51](#_Toc42277605)

[致谢 53](#_Toc42277606)

[附录A 原理图和程序代码 55](#_Toc42277607)

[附录B 英文翻译 83](#_Toc42277611)

# 引言

一种新想法的出现往往是为了解决生产生活中的一个难题，想要设计一个新的产品的灵感大概率也是来源于生产生活。

## 1.1 研究背景和意义

随着全民的收入水平的持续提高，机动车市场保持着可观的销量。最新数据显示我国的机动车保有量达3.48亿，汽车保有量已经达到2.6亿辆之巨。车辆可以为人们的生产生活带来方便，但数额庞大的车辆在带来便利的同时也带来了很多问题，比如环境污染和交通安全等。

科技的发展是为了成就更美好的生活，随着车辆技术和互联网通信技术的发展，一些新的概念出现在人们的视野中，例如车联网、智慧交通，智能驾驶等。

现在，我们已身处“大数据”时代，我们不仅无时无刻在产生数据，也无时无刻在使用数据，要想把上述构想搬进生活里就离不开与海量的数据打交道。

如今绝大部分车辆还是在独立的个体的状态，车辆产生的数据也被限制在独立的车辆中，在很多情况下这些数据还没来得及产生价值就消失了。虽然目前已经出现了一些实现车辆信息的收集与管理的应用，比如说很多保险公司对投保车辆免费赠送OBD设备，赠送的OBD设备获取到的数据可能成为理赔时的参考依据。但在面对巨大数量的车辆保有量时，这些运用还远远不够，还需要一款容易使用的解析器来方便的对车辆的信息进行采集。

虽然市面上已经出现了一些能够获取车辆信息的设备，但大多数都是工作在脱机模式下，不能实现数据的共享。也有能实现对车辆数据远程管理的设备，但数量比较少，而且多使用2G/3G/4G通信，所以价格也比较昂贵，且其数据接口并不公开，要想在其基础上进行深层次开发是不可能的。所以，本次论文决定设计一个既能满足传统OBD解析器的使用情景，即使用使用蓝牙通信就近读取车辆数据，又能实现大规模数量的车辆的信息的采集和远程传输的智能OBD解析器，即使用Wi-Fi通信实现数据远程传输和管理。对采集到的数据进行分析，并使数据产生价值是让传统的OBD解析器在大数据、智慧交通时代实现了华丽的转身。

## 1.2 国内外研究现状

为了解决洛杉矶地区的雾霾问题，加利福尼亚州开始要求在1966年的模型车上使用排放控制系统。美国联邦政府于1968年在全国范围内扩展了这些控制措施。

美国国会于1970年通过了《清洁空气法》，并成立了环境保护署（EPA）。这开始了一系列渐进的排放标准和长期维修车辆的要求。为了满足这些标准，制造商转向了电子控制的燃油供给和点火系统。内置的传感器可以测量发动机性能，并调整系统以减少污染。还可以访问这些传感器以提供早期诊断帮助。这可以算是OBD的前身了。

最初，标准很少，每个制造商都遵循自己的标准。为了监测排放控制系统，美国通用汽车在1982年提出了OBD概念。OBD系统一旦发现故障就会点亮仪表板上的一个指示灯来提醒驾驶者，并在车载计算机（即ECU或ECM）内记录一个代码，这个代码可被相应设备读取用于排除故障[1]。加州空气资源委员会（CARB）关注到了通用汽车公司提出的这一新概念，并于1985年采用了由汽车工程师协会（SAE）制定的标准，要求从1988起所有在加州销售的车辆都必须具有一些基本的OBD功能。接着EPA 要求自1991年起所有在美国销售的新车必须支持相关OBD技术要求[2]，这就是所说的第一代OBD。

SAE在第一代OBD的基础上做了进一步标准化工作，OBD发展到了第二代。

OBD-II不仅在诊断功能方面有很大的进步，还在标准化方面有明显的提升。 比如规范了诊断接口、故障指示灯、外部设备和ECU的通讯协议及故障码。此外，OBD-II可以获取更多信息。这些信息包含了故障码、一些重要信号或指标的实时数据，以及冻结桢信息等。1998年10月13日ISO组织在OBD-II制定了EOBD标准。

最新一代的无线传输系统OBD-III增强了无线数据传输能力，管理部门能通过移动通信，卫星通信或者GPS系统将车辆的VIN，故障码DTC及所在位置等信息。虽然更先进的OBD-III已经发布，但目前被各国政府及车辆制造厂商普遍使用的依旧是OBD-II协议[3]。

国外已经颁布了OBD相关法规，我国执行的最新标准是“国标六”。OBD技术发展至今，国内外很多研究者做了相关研究。比如国外的M. Farrugia, J. P. Azzopardi等人基于ELM37芯片开发了OBD解析器，通过连接不同的车辆，对获取到的OBD数据进行分析**错误!未找到引用源。**。R. Malekian, N. R. Moloisane等人设计了一套无线车队管理系统，能够对车辆速度、油耗、运行里程等信息进行监测和管理[4]。

我国的OBD技术的研究虽然起步比较晚，但是发展迅速。在上世纪90年代末期，已经有一批科研机构着手OBD相关技术的研究。随着国III排放标准在全国范围内的逐步实施，OBD技术的研究已经全面展开，包括低排放车辆OBD监测原理，欧洲法规对OBD系统的技术要求，OBD功能性演示试验失效零部件或模拟器的研制，OBD试验认证程序制定和OBD与在用车排放管理、排放缺陷召回等制度方面的相关研究[6]。不仅如此，境内高校机构也针对OBD做了很多相关的研究。比如太阳理工大学孟晓亮基于OBD-II标准中ISO9141-2、SAE J1979等协议开发了一款便携式的车辆诊断测试仪，克服了以往OBD测试仪通用性差，价格昂贵，体积笨重的问题[7]。合肥理工大学的匡书池设计了一款基于4G的OBD远程监控系统[8]。还有戴金池,庞海龙,俞妍等做的柴油车DPF系统的OBD故障诊断策略研究[9]等。

## 1.3 主要内容和工作安排

本次设计的主要内容是智能OBD解析器的设计，全文总共分为6章，详细安排如下：

第1章为引言，分析课题研究背景及意义，介绍国内外的研究现状。

第2章为方案的可行性分析，从不同的角度出发，综合考虑各种因素对方案进行对比，确定整体的方案设计，并简要介绍了OBD-II技术。

第3章为硬件系统架构搭建，主要包含主控芯片OBD协议芯片,蓝牙模组，OBD接口的详细介绍及原理图设计。

第4章为软件设计，下位机与上位机开发环境的搭建，设计了软件工作流程并确定了无线通信协议，还介绍了主要使用的功能函数。

第5章为硬件和软件的调试，主要记录了系统的调试过程，主要包含硬件系统和软件以及功能测试，并判断是否达到了设计要求。

第6章为对整个设计过程的总结，回顾了在整个设计中遇到的困难，设计存在的不足，以及对未来的展望。

# 第2章 方案论证与可行性分析

一套完整且具有可行的方案是整个设计的基础，后面的设计都需要参考选择的设计方案。本章主要讨论智能OBD解析器的设计方案。

## 2.1 设计要求

根据任务书的要求，本次设计需要从硬件系统构架和软件系统两方面着手，并使用无线通信技术将下位机与上位机的数据交互。

本次毕业设计的总体要求为实现对车辆实时状态监测的需求，具体要求有以下几点：

1. 能实时监测车辆速度、油耗。
2. 能检测车辆的异常状态。
3. 能通过Wi-Fi或蓝牙通信方式上传数据至其他设备进行监测。
4. 能查询和清除故障码信息。

## 2.2 方案论证

目前主流的OBD解析器可分为三种类型，一种是只采用线缆连接，另一种是采用蓝牙进行通信的，还有一种就是采用Wi-Fi进行的通信的。采用线缆的方案不在本次设计的讨论之中，所以就还剩下两种方案。

方案一：采用蓝牙通信的OBD解析器,硬件系统架构方案如图2.1所示：

|  |
| --- |
| 图2.1 采用蓝牙通信的OBD解析器方案 |

本方案的重要组成分为三部分，它们分别是OBD协议芯片，MCU微解析器和蓝牙模组。这种设计方案能在脱机状态下工作，完成对车辆数据的读取和发送指令的任务。

方案二：采用Wi-Fi通信的OBD解析器，硬件系统架构方案如图2.2所示：

|  |
| --- |
|  |
| 图2.2 采用Wi-Fi通信的OBD解析器方案 |

此方案与方案一只有在通信方式上存有差异，所以把方案一的蓝牙模组换成了Wi-Fi模组。本方案可以通过Wi-Fi上传获取车辆的数据，实现车辆的远程管理，但不能在脱机状态下工作。

对两种方案进行分析后发现，这两种方案均不能满足本次设计的要求，所以需要设计一个新的适用于本次设计的方案。

对本次设计的要求进行解读后，发现本次设计的智能OBD解析器吸取了前两种设计方案的OBD解析器的优点，既能在脱机状态下工作，也能实现远程管理功能。本次设计的初始方案如图2.3所示：

|  |
| --- |
|  |
| 图2.3 智能OBD解析器初始方案 |

在初始方案里，为了实现不仅能在脱机状态下工作，而且还能实现远程管理的功能分别使用了蓝牙模组和Wi-Fi模组，并且其他的方案都应该以此初始方案为蓝图，在此方案的基础上进行调整。

### 2.2.1 主控芯片

在嵌入式领域，全世界的嵌入式处理器种类繁多，体系结构多种多样。嵌入式处理器的选择是根据具体的应用决定的，以下列出了几种常见的微解析器。

方案一：MCS51单片机。以8051为核心的单片机称为MCS51单片机,也就是51单片机[10]，是大多学生在本科阶段最先接触到的单片机。51单片机的主要资源如表2.1所示：

|  |  |  |
| --- | --- | --- |
| 表2.1 51单片机资源概况 | | |
| 序号 | 资源 | 数量 |
| 1 | IO | 32 |
| 2 | 中断数量 | 5 |
| 3 | 寄存器位数 | 8 |
| 4 | 定时器数量 | 2 |
| 5 | 串口 | 1 |

STC公司的常见51系列单片机如图2.4所示：

|  |
| --- |
|  |
| 图2.4 STC公司的51单片机 |

方案二：MSP430系列单片机。美国德州仪器（TI）在1996年推出了一种16位超低功耗的混合信号处理器，即MSP430单片机。其称之为混合信号处理器，主要是由于其针对实际应用需求，把许多模拟电路、数字电路和微处理器集成在一个芯片上，以提供“单片”解决方案。MSP430一直以来都作为超低功耗微处理器的标杆产品，在便携式检测控制仪器、数据采集系统、智能仪表、医疗仪器、节能设备等方面都取得了广泛的应用[11]。常见的MSP430单片机如图2.5所示：

|  |
| --- |
|  |
| 图2.5 TI公司的MSP430单片机 |

方案三：STM32系列单片机。STM32系列是意法半导体公司生产的以Arm Cortex-M系列为内核专为高性能、低成本、低功耗的嵌入式应用而设计[12]。STM32f407ZGT6的主要资源如表2.2所示；常见的STM32F4系列单片机芯片如图2.6所示：

|  |  |  |
| --- | --- | --- |
| 表2.2 STM32单片机资源概括 | | |
| 序号 | 资源 | 数量 |
| 1 | IO | 144 |
| 2 | 中断 | 多种触发方式 |
| 3 | 定时器 | 16 |
| 4 | 寄存器位数 | 32 |
| 6 | 串口 | 6 |

|  |
| --- |
|  |
| 图2.6 ST公司的STM32F407单片机 |

方案四：ESP32单片机。ESP32是一款由乐鑫信息科技（上海）股份有限公司(以下简称乐鑫公司)生产的低功耗Wi-Fi、蓝牙芯片。ESP32在单芯片上集成了Wi-Fi（2.4 GHz频率）和蓝牙4.2解决方案[13]。

不仅如此，ESP2还有普通单片机的外设，如IO口，定时器，ADC等，但总体外设资源数量较少。ESP32系列时钟频率最高可达240 MHz.在需要无线通信的场景下，ESP32单片机非常具有诱惑力。常见的ESP32单片机芯片如图2.7所示。

|  |
| --- |
|  |
| 图2.7 乐鑫公司的ESP32 |

### 2.2.2 蓝牙通信模组

蓝牙这一种短距离无线通信技术被广泛运用到各个领域。目前市面上有三种蓝牙，他们分表示单模蓝牙，经典蓝牙，和双模蓝牙。单模蓝牙为低功耗蓝牙，支持与较新的移动设备连接。经典蓝牙功耗比低功耗蓝牙高，工作流程也有差别，但其支持与较老的蓝牙设备进行连接。双模蓝牙既支持低功耗蓝牙又支持经典蓝牙，结合了两种蓝牙的优势，兼容性更广，扩展能力更强。对目前流行的蓝牙平台进行调研后发现可以分为两种方案。

方案一：ESP32芯片集成的蓝牙。

方案二：独立蓝牙模组。该种多使用AT指令进行控制，使用UART的方式与其他终端进行数据交互，市面上应用较多的有广州汇承信息科技有限公司（以下简称汇承公司）的HC系列蓝牙模组。汇承公司的HC-02蓝牙模组如图2.8所示：

|  |
| --- |
|  |
| 图2.8 HC-02蓝牙模组 |

### 2.2.3 Wi-Fi通信模组

WI-FI是一种基于 IEEE802.11 标准，旨在提高无线网络产品互操作性的短程无线传输技术。Wi-Fi 主要的特点就是覆盖的范围比较广、传输的速率快并且可靠性高、不受布线条件的限制、健康安全，目前在日常生活中使用广泛。Wi-Fi平台的选择，主要有两种方案。

方案一：ESP32或者ESP8266单片机集成的Wi-Fi解决方案。ESP32和ESP8266单片机在互联网产品中使用非常广泛，在国内外都拥有大量的应用案例。

方案二：独立Wi-Fi通信模块。独立Wi-Fi通信模块主要使用AT指令进行配置使用，多使用UART串口通信与其他终端进行数据交互。汇承公司不仅又丰富的蓝牙通信模块的产品线,而且有Wi-Fi通信模块。HC-25Wi-Fi模组如图2.9所示：

|  |
| --- |
|  |
| 图2.9 HC-25Wi-Fi模组 |

### 2.2.4 OBD协议芯片

OBD协议芯片是车辆信息与终端设备交流的桥梁。在选择OBD协议芯片之前，有必要了解OBD协议。

OBD协议体系结构借鉴了OSI参考模型并删除了会话层和表示层，所以OBD协议体系结构分为5层，分别是物理层、数据链路层、网络层、传输层和应用层[14]。

ISO和SAE都制定了OBDII汽车故障诊断协议，它们在应用层上存在些许差异，但其他基本相同，它们的对应关系如表2.3所示。

|  |  |  |
| --- | --- | --- |
| 表2.3 ISO与SAE协议对应关系 | | |
| ISO | SAE | 协议内容 |
| － | 15031-1 | 通用信息 |
| J1930 | 15031-2 | 术语、定义、缩约词 |
| J1962 | 15031-3 | 诊断接口 |
| J1978 | 15031-4 | 外部诊断设备 |
| J1979 | 15031-5 | 排放有关的诊断 |
| J2012 | 15031-6 | 故障码 |
| J2186 | 15031-7 | 数据链路安全 |

ISO与SAE指定的OBD-II协议的主要差异在诊断座接口定义方面。它们的诊断连接器结构是一样的，都采用了16端子诊断连接器，但各端子定义略有差别。OBD诊断接口模型如图2.10所示，OBD诊断接口端子功能定义如表2.4所示。

|  |
| --- |
|  |
| 图2.10 OBD标准结构模型图 |

|  |  |  |  |
| --- | --- | --- | --- |
| 表2.4 OBD标准接口引脚功能[15] | | | |
| 引脚 | 功能 | 引脚 | 功能 |
| 1 | Reserve | 9 | Reserve |
| 2 | J1850+ | 10 | J1850- |
| 3 | Reserve | 11 | Reserve |
| 4 | GND | 12 | Reserve |
| 5 | GND | 13 | Reserve |
| 6 | CAN\_H | 14 | CAN\_L |
| 7 | ISO09141-2(KWP2000)K Line | 15 | ISO09141-2(KWP2000)L Line |
| 8 | Reserve | 16 | +12 V |

市面上可见到的OBD协议芯片种类不多，它们的型号及一些参数如表2.5所示：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表2.5 市面上主流的OBD协议芯片/模组主要指标 | | | | | |
| 型号 | ELM327 | TDA60 | TDA61 | TDA66 | EST527-miniS |
| 指令 | ELM327指令 | 兼容ELM327指令 | 兼容ELM327指令并拓展多种自由指令 | 兼容ELM327指令并拓展多种自由指令 | 自有AT指令，并支持自定义ISO15031-5标准包含PID |
| 内核 | 8bit MCU | 8bit MCU | 8bit MCU | Contex-M3 | 双MCU，其他未知 |
| 通信接口 | RS232 | RS232 | RS232 | RS232,IIC,SPI | UART |
| 支持协议 | OBD-II | OBD-II | OBD-II | OBD-II | OBD-II |

市面上的OBD解析器大多使用ELM327方案，其次是TDA6X系列方案，EST527-miniS方案比较少见。

## 2.3 硬件系统架构方案

在初始设计方案的基础上，结合主要器件/模组型号得出了几种不同的硬件系统架构方案。

方案一：该智能OBD解析器方案主要由STM32F407ZGT6最小系统开发板主控平台，HC-02蓝牙模组，ESP8266最小系统Wi-Fi模组，EST527-miniS 模组，和标准OBD接口组成。该方案是整体硬件架构原始方案的具体实现。本方案的具体架构如图2.11所示：

|  |
| --- |
|  |
| 图2.11 一种硬件系统架构具体实现方案 |

方案二：该智能OBD解析器方案主要由：ESP32最小系统平台，EST527-miniS模组，HC-02蓝牙模组，OBD标准接口组成。该方案把Wi-Fi模组，主控芯片合二为一，是开发效率最高的方案。本方案的具体架构如图2.12所示：

|  |
| --- |
|  |
| 图2.12 一种硬件系统架构具体实现方案 |

方案三：该智能OBD解析器方案主要由：ESP32最小系统主控平台，EST527-miniS模组和OBD标准接口组成。该方案把蓝牙模组，Wi-Fi模组，主控芯片合三为一，是硬件组成最简洁，成本最经济的方案。本方案的具体架构如图2.13所示：

|  |
| --- |
|  |
| 图2.13 一种硬件系统架构具体实现方案 |

经过对比，和初步尝试最终选择了方案二作为本次设计的硬件系统架构。

## 2.4 总结

本章主要介绍了元器件的选型和硬件系统架构方案的确定。硬件系统作为整个智能OBD解析器的基础，元器件的选择至关重要。元器件的选择最重要的因素是满足设计要求的功能，其次才是开发难度，能耗，成本等因素。结合实际情况，本次设计选则了方案二作为硬件系统架构方案，在预选的三个方案中，方案二的硬件结构比较统一，开发难度也适中，硬件成本也在可接受的范围内。综上所述，方案二是本次设计的最优方案。

# 第3章 智能OBD解析器硬件设计

设计方案已经确定，接下来的主要任务就是了解模组的参数及使用方法，并绘制原理图。

## 3.1 总体设计

根据选定的方案，对硬件系统进行总体设计，主控芯片和实现Wi-Fi通信功能使用的是乐鑫公司的ESP32-WROOM-32单片机，该芯片的运算速度，外设资源都满足本次设计的需要，并且其SDK带有FreeRTOS实时操作系统，可以最大程度满足实时性的要求；蓝牙通信的实现使用的是汇承公司的HC-02蓝牙模组，该蓝牙模组为双模蓝牙，其兼容性广，通信速率快，可靠性高，价格适中，通过串口的连接方式和主控芯片进行通信，使用非常方便；OBD协议芯片选择深圳速锐得科技有限公司（以下简称速锐得公司）的EST527-miniS模组，该模组封装了必要的外围电路，而且其体积较小可以降低PCB设计难度;屏蔽了OBD协议，资料详尽，可大幅缩短开发时间。硬件系统的通信连接方式如图3.1所示。

|  |
| --- |
|  |
| 图3.1 硬件系统的通信连接 |

## 3.2 OBD协议芯片

OBD协议芯片作为获取车辆信息的核心部件，本次设计选用的速锐得公司生产的EST527-minis模组，其主要特征如表3.1所示：

|  |  |
| --- | --- |
| 表3.1 EST527-miniS模组的主要特点 | |
| 序号 | 特点 |
| 1 | 标准 OBDII 接口支持 |
| 2 | 上位机无需进行任何运算，所有数据都以数值方式返回 |
| 3 | 双 MCU，处理速度快，是 ELM327的5倍 |
| 4 | 车辆点火自动唤醒，车辆熄火自动休眠 |
| 5 | 支持瞬时油耗、平均油耗及耗油量数据 |
| 6 | 支持车辆故障码诊断，两条指令即可完成故障码的读取和清除，支持实时故障码扫描 |
| 7 | 车规级抗干扰设计 |
| 8 | AT 指令集简单易用 |

EST527-miniS模组支持的协议如表3.2所示：

|  |  |  |
| --- | --- | --- |
| 表3.2 EST527-minis模组支持的OBD通信协议 | | |
| 序号 | 协议名称 | 协议标准 |
| 1 | ISO9141-2 | ISO9141 |
| 2 | KWP2000\_5BPS | ISO14230 |
| 3 | KWP2000\_FAST |
| 4 | CANBUS\_11B\_500K | ISO15765 |
| 5 | CANBUS\_29B\_500K |
| 6 | CANBUS\_11B\_250K |
| 7 | CANBUS\_29B\_250K |

EST527-minis模组具体参数如表3.3所示。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 表3.3 EST527-miniS模组具体参数 | | | | |
| 序号 | 参数 | 最小值 | 最大值 | 单位 |
| 1 | 工作电压 | 5 | | V |
| 2 | 工作电流 | 20 | 40 | mA |
| 3 | 工作功耗 | 100 | 200 | mW |
| 4 | 休眠电流 | 0.5 | 2.0 | mA |
| 5 | 休眠功耗 | 2.5 | 10.0 | mW |
| 6 | 工作温度 | -20 | 70 | ℃ |
| 7 | 存储温度 | -40 | 85 | ℃ |
| 8 | ESD耐压 | -8 | +8 | KV |
| 9 | 模块尺寸 | 26（L）\*16(W)\*3.5(H) | | Mm |
| 10 | 接口标准 | UART | | |
| 11 | 波特率 | 9600/38400/115200 | | Bps |

EST527-miniS模组引脚定义如图3.2所示：

|  |
| --- |
|  |
| 图3.2 EST527-minis模组引脚定义图 |

EST527-miniS模组的1引脚与OBD标准接口的VBAT功能引脚相连；2引脚和3引脚分别于OBD标准接口的ISO\_LH和ISO\_K功能引脚连接；4引脚和5引脚分别和OBD标准接口的CAN\_L和CAN\_H功能引脚连接；6引脚和电源地端连接；7引脚与串口转换电路的TXD端连接，8引脚与串口转换电路的RXD端连接；9引脚与EST527-miniS模组复位按键连接；9引脚和10引脚分别与EST527-miniS模组的外加点火指示灯和状态指示灯电路连接；12引脚与电源处理电路输出端的跳线帽端连接。

EST527-minis模组的原理图如图3.3所示：

|  |
| --- |
|  |
| 图3.3 EST527-minis模组原理图 |

## 3.3 蓝牙通信模块

|  |
| --- |
| 图3.4 HC-02蓝牙模块引脚定义 |

HC-02蓝牙模块引脚定义图如图3.4所示：

HC-02蓝牙引脚说明如表3.4所示：

|  |  |  |
| --- | --- | --- |
| 表3.4 HC-02蓝牙模块引脚功能 | | |
| 引脚编号 | 引脚名称 | 引脚说明 |
| 1 | STATE | 蓝牙连接成功时输出高电平，否则输出低电平 |
| 2 | RXD | UART的接收端 |
| 3 | TXD | UART的发送端 |
| 4 | GND | 电源负极 |
| 5 | VCC | 电源正极 |
| 6 | KEY | 模组上电后接高电平进入普通模式，接低电平进入AT指令模式 |

HC-02模组的1引脚与ESP32-WROOM-32开发板的G21引脚；2引脚与ESP32开发板的25引脚连接；3引脚与ESP32开发板的26引脚连接；3引脚与电源地连接；4引脚与ESP32开发板3.3V电源端连接；5引脚用于设置HC-02的工作模式，正常工作时与3.3V电源端连接。

蓝牙模组的原理图如图3.5所示：

|  |
| --- |
|  |
| 图3.5 蓝牙模组原理图 |

## 3.4 主控芯片与Wi-Fi模组

主控芯片选择的是乐鑫公司的ESP32-WROOM-32芯片，但为了使本次设计更加高效，使用了搭载ESP32-WROOM-32芯片的最小系统开发板。开发板上引出了芯片的所有IO口，也板载了程序下载电路，不必额外的进行电路设计就可以方便的进行程序烧录和使用串口的方式进行调试，使用非常的方便，而且最小系统开发板体积小巧，不会影响系统的硬件结构。ESP32-WROOM-32最小系统开发板引脚及复用功能如图3.6所示：

|  |
| --- |
|  |
| 图3.6 ESP32最小系统开发板引脚及复用 |

|  |  |
| --- | --- |
| 表3.5 ESP32引脚使用情况 | |
| 引脚编号 | 用途 |
| G13 | Wi-Fi模式LED |
| G16 | UART1 TX |
| G17 | UART1 RX |
| G21 | HC-02状态引脚 |
| G25 | UART2 TX |
| G26 | UART2 RX |
| G32 | WI-FI选择按键 |
| G33 | 蓝牙选择按键 |

本次设计使用的IO口及功能说明如表3.5所示：

ESP32-WROOM-32开发板的G13引脚与外加Wi-Fi提示LED灯连接；G16,G17分别与串口转换电路的TX1和RX1端连接；G21与HC-02模组的1引脚连接；G25,G26分别与HC-02模组的2引脚和3引脚连接；G32,G33分别于Wi-Fi和蓝牙选择按钮端连接。

主控芯片与Wi-Fi模组的原理图如图3.7所示：

|  |
| --- |
|  |
| 图3.7 主控芯片与Wi-Fi模组的原理图 |

## 3.5 OBD接口

OBD接口的原理图如图3.8所示。

|  |
| --- |
|  |
| 图3.8 OBD接口的原理图 |

OBD标准接口除了与EST527-miniS模组连接使用了的引脚和4，5引脚需要与电源端地连接外，其他的引脚都作保留，不做任何连接。

## 3.6 电源芯片

本次设计的智能OBD解析器自身不能提供电源，需要从OBD接口获取电源，但是车辆电瓶的电压高于EST527-miniS模组和其他重要器件的工作电压5V，所以还需要添加电源处理电路。本次设计使用的是78M05稳压芯片，该芯片的输入电压最高可达35V，输出端为5V。电源处理部分的原理图如3.9所示：

|  |
| --- |
|  |
| 图3.9 电源处理原理图 |

78M05芯片的1引脚与OBD标准接口的VBAT引脚连接；2引脚与电源地端连接；3引脚为输出端，与EST527-miniS的12引脚和ESP32开发板的5V电源端连接。

此外，在实际测试的过程中发现稳压芯片处的温度叫较高，所以在绘制PCB时应该加强散热处理，比如使芯片区域处镂空，还应远离容易受温度影响的器件。

3.7 本章总结

本章主要介绍了总体设计以及使用到的元件的参数情况，此外并根据选定的硬件系统构架方案和选定的元件型号绘制了电路原理图，并根据模块的使用手册建立了封装库，在此基础上绘制了PCB电路图（详情参见附录A）并通过把生产文件发送给专业的电路板生产厂商进行了打样。在制作印刷电路板时并没有一次成功，在第一次尝试失败后，我就决定把主要元件做成可拆卸式的，这样可以提高试错能力。经历了三次失败后，在PCB电路板上焊接上各个元件后，确定没有影响使用的问题，但还存在改进空间。

# 第4章 智能OBD解析器软件设计

智能OBD解析器的软件设计主要包含下位机和上位机，即主控芯片程序设计和PC端软件设计。

4.1 主控芯片开发环境搭建

主控芯片选择的是乐鑫公司的ESP32-WROOM-32芯片，该公司推荐的开发环境是ESP-IDF，该开发环境集成了FreeRTOS实时操作系统。FreeRTOS作为一个轻量级嵌入式操作系统，提供了一个高层次的可信任代码。源代码以C语言开发，系统实现的任务没有数量限制，FreeRTOS内核支持优先级调度算法，同时支持轮换调度算法[15]。安装此开发环境的理想平台是Linux，由于目前大多数使用的电脑是Windows平台，所以需要用到虚拟机。由于使用了ESP32-WROOM-32最小系统开发板，所以烧录程序非常方便，不用使用额外的工具，只需要用USB数据线把最小系统开发板连接到电脑的USB端口就行。ESP-IDF的安装可分为如下几个步骤：

1. 设置工具链；
2. 从GitHub获取ESP-IDF；
3. 设置环境变量；
4. 安装python包；
5. 尝试运行开始例程，在监视窗口例程现象以验证开发环境是否搭建成[[1]](#footnote-1)。

ESP-IDF只提供了编译工具链，并没有提供代码编辑工具，所以需要自己选择IDE。市面上可供选择的IDE有很多。经过尝试，选择Source Insight软件作为ESP32的代码编辑工具，使用samba软件作为虚拟机和Windows主机文件传输桥梁。

打开esp-idf\examples\get-started\hello\_world\main目录下的hello\_world\_main.c文件，可以发现除了包含有一些头文件，其他的就只有一个app\_main()函数，该函数是ESP32系统的入口函数，在底层初始化完成后从这里开始执行代码。经过编译、链接和烧录程序后可以观察到如图4.1所示的现象，观察到此现象说明ESP-IDF开发环境搭建成功了。该例程主要是周期性输出“hello world！”字符和向显示器输出一些系统信息。

|  |
| --- |
|  |
| 图4.1 ESP32第1个例程运行结果 |

该例程主要是周期性输出“hello world！”字符和向显示器输出一些系统信息。

值得注意的是，由于ESP-IDF集成了FreeRTOS实时操作系统，所以在编写自己的代码时不能与裸机编程一样在入口函数里使用大循环while，否则会给编程带来麻烦。

## 4.2 Wi-Fi通信协议选择

硬件系统的连接方式由元件/模组确定，不能做出选择。但是在使用Wi-Fi进行通信时使用的协议还没有确定，所以需要选择一个合适的方案。目前可供选择的协议有多种，结合ESP32-WROOM-32单片机支持的通信协议，主要有以下四种方案：

方案一：TCP/IP传输协议。TCP/IP传输协议不是指单个协议，而是一个协议簇。TCP最主要的几个特点如表4.1所示：

|  |  |
| --- | --- |
| 表4.1 TCP的主要特征 | |
| 序号 | 特征 |
| 1 | TCP是面向连接 |
| 2 | 每一条TCP连接只能由两端点 |
| 3 | TCP提供可靠交付的服务 |
| 4 | 面向字节流 |

方案二：用户数据报协议UDP。UDP只在IP的数据包服务之上添加了端口的功能和差错检验功能[17]。虽然UDP用户数据报只能提供不可靠的交互，但UDP在某些有其特殊的优点，如表4.2所示：

|  |  |
| --- | --- |
| 表4.2 UDP的优点 | |
| 序号 | 特征 |
| 1 | UDP是无连接的，发送数据之前不必建立连接 |
| 2 | UDP无拥塞控制，网络出现的拥塞不会使源主机的发送速率降低 |
| 3 | UDP是面向报文的 |
| 4 | UDP支持一对一、一对多、多对一和多对多的交互通信 |
| 5 | UDP开销比TCP短 |

方案三：WebSocket。WebSocket的主要特点如表4.3所示：

|  |  |
| --- | --- |
| 表4.3 WebSocket的主要特点 | |
| 序号 | 特点 |
| 1 | 使用 TCP/IP 提供网络连接 |
| 2 | 全双工通信 |
| 3 | 只需要建立一次连接即可永久连接 |

方案四：MQTT。MQTT即消息队列遥测传输，由IBM和Eurotech研发且开源的传输协议[18]。它是一种发布/订阅，极其简单和轻量级的消息传递协议，旨在用于受限设备和低带宽，高延迟或不可靠的网络。设计原则是使网络带宽和设备资源要求最小化，同时还要尝试确保可靠性和一定程度的交付[19]。它的主要特点如表4.4所示：

|  |  |
| --- | --- |
| 表4.4 MQTT的主要特点 | |
| 序号 | 特点 |
| 1 | 使用发布/订阅消息模式，提供一对多的消息发布 |
| 2 | 使用 TCP/IP 提供网络连接 |
| 3 | 小型传输，开销很小（固定长度的头部是 2 字节），协议交换最小化 |
| 4 | 使用 Last Will 和 Testament 特性通知有关各方客户端异常中断的机制 |
| 5 | “至多一次”，消息发布完全依赖底层 TCP/IP 网络，会发生消息丢失或重复 |
| “至少一次”，确保消息到达，但消息重复可能会发生 |
| “只有一次”，确保消息到达一次 |

其中的第5个特性中选择合适的消息质量等级可以保证MQTT通信的可靠性。结合设计需求，综合考虑后决定选取MQTT协议来实现远程的数据传输。MQTT的通信方式如图4.2所示。

|  |
| --- |
|  |
| 图4.2 MQTT通信 |

任意两个MQTT客户端都不能进行点对点的通信，无论是发送消息，还是接收消息都需要通过Broker的转发，这种模式使MQTT能轻易的实现大规模客户端的通信。

4.3 下位机工作流程

根据对任务书要求的分析，下位机主要实现车辆数据的采集工作和执行上位机下达的指令信息。下位机的工作流程如图4.3所示，按照此工作流程拆分功能，编写相应的任务函数，重要的任务函数如表4.5所示：具体的代码参见附录A。

|  |
| --- |
|  |
| 图4.3 下位机工作流程图 |

|  |  |  |
| --- | --- | --- |
| 表4.5 ESP32任务函数说明 | | |
| 序号 | 任务名称 | 功能说明 |
| 1 | taskSwitchMode() | 通信方式选择任务 |
| 2 | taskWatingBluetooth() | 等待蓝牙连接任务 |
| 3 | taskLED\_WIFI() | LED闪烁任务，提示程序运行 |
| 4 | taskLED\_Bluetooth() | LED闪烁任务，提示程序运行 |
| 5 | taskUARTTwo() | ESP32与蓝牙模组通信任务 |
| 6 | taskUartOneWIFI() | Wi-Fi通信方式下ESP32与EST527模组通信任务 |
| 7 | taskUARTOneBluetooth() | 蓝牙通信方式下ESP32与EST527模组通信任务 |
| 8 | mqtt\_event\_handler\_cb() | MQTT回调函数，包含MQTT相关事件处理 |
| 9 | mqtt\_app\_start() | 启动MQTT服务函数，并配置MQTT客户端参数 |
| 10 | app\_main() | 用户程序入口函数 |

## 4.4 上位机开发

根据对任务书要求的分析，本次设计要的上位机主要实现显示数据和下发送指令信息给下位机，所以既需要开发下位机，也需要进行上位机开发。上位机开发流程如图4.4所示：

|  |
| --- |
|  |
| 图4.4 上位机开发流程 |

开发上位机的编程语言众多，结合自身的情况，我选择了C#编程语言。C#是微软公司发布的一种面向对象的、运行于.NET Framework和.NET Core(完全开源，跨平台)之上的高级程序设计语言，由C和C++衍生出来，具有安全、稳定、简单的特点[20]。由于C#具有跨平台的特性，又有丰富的控件可以使用，结合智能OBD解析器的使用场景，以及自身的知识储备，选择C#开发上位机是一种非常合理的决定。由于C#是微公司提出的程序设计语言，所以微软开发的代码编辑软件Visaul Studio是最合适的IDE。Visaul Studio的安装非常简单，在这里简单介绍一下安装过程，具体步骤如下。

（1）使用浏览器打开<https://visualstudio.microsoft.com/zh-hans/downloads/>，然后选择版本进行下载，推荐使用社区版，因为社区版是可以免费使用的；

（2）双击安装包，程序会自动下载并配置必要的环境资源；

（3）环境资源配置完成后就可以选择工作负载，软件界面如图4.5所示；

|  |
| --- |
|  |
| 图4.5 Visual Studio工作负载选择界面 |

选择.NET桌面开发工作负载。

（4）安装完成后的启动界面如图4.6所示：阅读官方的开发文档可以对C#进行快速入门。

|  |
| --- |
|  |
| 图4.6 Visual Studio启动界面 |

开发环境搭建完成后就可以开始编写代码了。上位机的工作流程如图4.7所示：

|  |
| --- |
|  |
| 图4.7上位机工作流程 |

|  |  |  |
| --- | --- | --- |
| 表4.6 上位机使用到的重要功能函数 | | |
| 序号 | 函数名称 | 功能说明 |
| 1 | SerialPortScanf() | 扫描串口使用情况 |
| 2 | Form1() | 函数入口 |
| 3 | SerialPortSend() | 通过串口发送数据 |
| 4 | ComReceiveData() | 串口接收数据函数 |
| 5 | rtnWIFI\_CheckedChanged() | 取消Wi-Fi连接失能控制面板 |
| 6 | rtnBluetooth\_CheckedChanged() | 取消蓝牙连接失能控制面板 |
| 7 | timer1\_Tick() | 定时器函数 |
| 8 | btnScanf\_Click() | 扫面按钮事件 |
| 9 | cbxSerialPort\_SelectedIndexChanged() | 串口号连接点击事件 |
| 10 | btnBdisconnect\_Click() | 断开蓝牙连接按钮事件 |
| 11 | mqttClient.ConnectAsync() | MQTT连接按钮事件 |
| 12 | DealWithData() | 蓝牙和Wi-Fi接收数据处理函数 |
| 13 | ECUIsConnectFunction() | ECU连接成功处理数据函数 |
| 14 | btnMQTTDisconnect\_Click() | MQTT断开连接事件 |
| 15 | buttonInquiry\_Click() | 请求故障码 |
| 16 | MqttPublish() | 通过MQTT发送消息 |
| 17 | buttonClear\_Click() | 清空文本显示区内容 |
| 18 | comboBox1\_SelectedIndexChanged() | 其他信息查询函数 |

上位机使用的主要功能函数如表4.6所示：

在Visual Studio 平台上创建C#窗体应用时Visual Studio 平台会自动配置窗体的尺寸，控件的属性等信息，并进行注册。想要改变窗体尺寸，控件的属性信息也十分方便，直接拖动或者在选择栏进行选择就可以实现，而不用输入代码来操作。上位机的界面如图4.8所示：

|  |
| --- |
|  |
| 图4.8 上位机界面 |

完整代码请参见附录A，上位机界面如图4.5所示。整个上位机界面可分为四个部分，最上面部分为通信方式选择区，左下部分是实时数据流显示区，中下部分为显示区，可以显示下位机上报的状态信息和其他数据，右下部分是参数查询区，可以查询众多的参数信息。

4.5 本章总结

本章是软件设计部分，软件设计与硬件设计同等重要。硬件方案已经选定，难以再做出改动，即使改动，也只是涉及简单的线路问题。但是软件部分是非常灵活的，可以通过不断的优化，设计出更优秀的软件。好的程序不仅能够实现要求的功能，更能在使用计算机资源方面和能耗方面做出合理的安排。鉴于本次设计没有涉及到复杂的算法，所以软件设计部分主要工作是把设计的流程图通过代码的方式实现，并让程序能够正常的工作。在完成软件设计的过程中，使用了对于我来说是新的事物的东西。比如说FreeRTOS实时操作系统，以前只是听说过，但没有尝试过；还有C#这种编程语言，以前没有编写过上位机，但这次明确要求编写上位机程序，对于我来说的确是一个不小的挑战。通过这次设计我的编程能力有了明显的提升。

# 第5章 系统测试

本次设计的智能OBD解析器既包含了硬件部分，也包含了软件部分，所以需要对硬件系统和软件系统进行测试。具体测试内容包括印刷电路板测试，核心模组测试，蓝牙通信测试了和Wi-Fi通信测试。不仅如此，还需要对成品进行功能测试，以此判断本次设计是否符合要求，这个步骤在整个项目里是非常重要的一个环节。

## 5.1 硬件平台测试

### 5.1.1 印刷电路板测试

本次设计设计了印刷电路板作为器件的连接载体，所以在焊接器件之前有必要进行印刷电路板的连通性测试。印刷电路板如图5.1所示：

|  |
| --- |
|  |
| 图5.1 印刷电路板 |

印刷电路板的主要测试方法是利用万用表的蜂鸣器通断测试功能。在该功能模式下，万用表将通过检测两端的电阻来判断是否连通，如果测得的电阻小于设定的阀值，蜂鸣器响起，提示这两端是通路状态；如果两端的电阻大于设定的阀值或者为无穷大，蜂鸣器将不会响起，提示两端是断路状态。参照原理图对印刷电路板进行连通性测试。经过测试，印刷电路板的连接没有错误，且触点均已正确连接。

### 5.1.2 EST527-miniS测试

首先对EST527-miniS OBD协议模组进行通行测试。该模组使用UART通信与其他设备进行数据交换，所以最简单的测试方式就是利用USB-TTL模组对进行通行测试。EST527-miniS模组与USB-TTL模组连接如图5.2所示：

|  |
| --- |
|  |
| 图5.2 EST527-miniS模组与USB-TTL模组连接 |

接下来把USB-TTL模组连接到PC端，并打开串口调试助手，设置通信参数后，就可以发送指令了。测试结果如图5.3所示。

|  |
| --- |
|  |
| 图5.3 EST527-miniS UART通信测试 |

可以看到模组能从串口唤醒，串调试助手也能收到来自模组的消息，因此可以判断模组UART通信正常。

### 5.1.3 实物平台测试

在印刷电路板和主要的模组测试通过后就可以进行实物平台搭建了，搭建的实物平台如图5.4所示：

|  |
| --- |
| 图5.4 实物平台 |

在进行通电测试之前，还应该对触点进行测试，避免存在虚焊的情况。如果发现虚焊情况应及时补焊。

## 5.2 通信测试

### 5.2.1蓝牙双向通信测试

下位机接上电源后选择蓝牙通信方式，然后打开上位机，选择蓝牙通信方式并与下位机建立连接。测试过程如图5.5所示：

|  |
| --- |
|  |
| 图5.5 下位机与上位机蓝牙通信测试 |

可以在显示区看见来自下位机的数据，而“ECU NOT CONNECT”字符串意味着下位机收到上位机的指令数据，但是由于EST527-miniS模组并没连接到ECU，所以才会返回该字符串。因此可以得到下位机与上位机能正常的实现蓝牙通信的结论。

### 5.2.2 Wi-Fi双向通信测试

下位机接上电源后选择Wi-Fi通信测试通过后，紧接着进行上位机与下位机的Wi-Fi通信测试。为了方面进行测试和功能演示，在完成本次设计的过程中使用了本地EMQX服务器来作为MQTT的Broker服务器。由于ESP32-WROOM-32芯片既能支持MQTT的IP接入方式，又能支持MQTT的URL地址接入方式。所以采用本地EMQX服务器并不会影响最终的结果。

上位机与下位机的Wi-Fi通信测试流程如图5.6所示：

|  |
| --- |
|  |
| 图5.6 上位机与下位机的Wi-Fi通信测试流程 |

在这里简要介绍在Windows平台下EMQX服务的搭建的主要步骤。首先在<https://www.emqx.io/cn/downloads#broker>下载压缩包，然后解压到自定义文件夹并在\emqx\etc\plugins\ emqx\_auth\_username添加用户名及密码。接着使用命令窗口启动服务。

上位机与下位机的Wi-Fi通信测试过程如图5.7所示。

|  |
| --- |
|  |
| 图5.7 上位机与下位机的Wi-Fi通信测试 |

可以在显示区看见来自下位机的数据，而“ECU NOT CONNECT”字符串意味着下位机收到上位机的指令数据，但是由于EST527-miniS模组并没有与车辆ECU连接成功，所以才会返回该字符串。因此可以得到下位机与上位机能正常的通过Wi-Fi通信实现数据交互的结论。

## 5.3 功能测试

### 5.3.1 使用蓝牙通信时的功能测试

由于本次测试使用的车辆并没有存在故障码，所以就不能进行故障码读取和故障码清除测试。但是因为在读取车辆实时数据流时能正确的读取到故障码数量为0，所以可以大胆推测这两个功能可以实现。

|  |  |
| --- | --- |
| (a)上位机界面 | (b)仪表盘 |
| 图5.8 蓝牙通信方式下的实时数据流 | |

蓝牙通信方式下的实时数据流测试如图5.8所示，实时数据流不仅能显示车速信息，还能显示引擎转速、引擎负荷、电瓶电压、瞬时油耗、本次点火之后的油耗和冷却液温度信息。处于安全方面的考虑，本次测试并没有让车辆的车速跑到很高，但已经得到明显的测试结果。在此通信方式下的其他PID查询测试结果如图5.9所示：

|  |
| --- |
| 图5.9 蓝牙通信方式下其他PID查询 |

然而，在上位机列出的其他PID参数中，并不是所有的条目都能被ECU支持，有些ECU只支持其中的某部分，支持的会返回参数，不支持的会返回“ECU not support”字符串，具体情境如图5.10所示：

|  |
| --- |
|  |
| 图5.10 蓝牙通信方式下不支持PID测试 |

不仅如此，上位机还支持自定PID查询，此情景的具体结果如图5.11所示：

|  |
| --- |
|  |
| 图5.11 蓝牙通信下的自定义PID查询测试 |

PID 具体请参考 ISO15031-5，在此标准中详细介绍了PID。在测试中还发现当尝试读取车辆VIN码时，如果车辆不支持的话可能导致EST527-miniS模组重启，甚至与ECU的连接中断，如果出现这种情况的应把OBD解析器与车辆断开，等待一会儿再重新连接。至此，使用蓝牙通信时本智能OBD解析器能正常工作。

### 5.3.2 使用Wi-Fi通信时的功能测试

由于Wi-Fi通信时使用的数据处理函数和逻辑条件判断都与使用蓝牙通信时是相同的，换句话说就是两种通信方式下共用了所有的函数。所以只要Wi-Fi能正常通信，功能都应该能实现。为了保持严谨的态度，还是在Wi-Fi通信下进行了测试。在此通信方式下的实时数据流如图5.12所示，其他参数查询结果如图5.13所示，自定义PID查询测试如图5.14所示：

|  |
| --- |
|  |
| 图5.12 Wi-Fi通信方式下的实时数据流测试 |

|  |
| --- |
|  |
| 图5.13 Wi-Fi通信方式下的其他参数查询测试 |

|  |
| --- |
|  |
| 图5.14 Wi-Fi通信方式下的自定义PID测试 |

至此，Wi-Fi通信下测试完成，智能OBD解析器能正常工作。

通过测试，本次设计的智能OBD解析器既能在蓝牙通信实现车辆信息的读取和传输和在上位机端操作下位机，又能使用Wi-Fi通信实现车辆信息的远程传输和在远程端对下位机进行控制，从而实现远程操作车辆数据的功能。

## 5.4 本章总结

本章介绍了系统的测试过程。 在测试过程中发现了很多问题，比如HC-02蓝牙模块在印刷电路板上存在不能正常启动的情况，经排查，发现主要原因是蓝牙模组排母的引脚出了虚焊的情况，进行补焊后蓝牙模块在印刷电路板上就能正常工作了；模式选择按钮按下时发生抖动，首先分别采取了查询和按键中断的方式并修改上拉下拉模式，但抖动依然存在。电路板已经焊接完成了，难以使用硬件消抖的方法，所以就通过修改软件的方式来解决。通过查阅资料，发现按键选择的IO引脚不是优先使用引脚，通过跳线修改按键接入引脚编号才得以解决；在下位机编写串口通信时使用队列进行数据传输，结果系统无限重启，其中的原因未知，采取轮询读取串口的数据的方式问题才得以解决；下位机子功能函数都可以正常工作，但是把他们组合在一起时就不能正常工作，查阅资料后发现是任务调度的问题，这是在使用实时操作系统经常会遇到的问题，修改任务优先级后系统就能够正常工作了。总的来说经过测试，本次设计的智能OBD解析器实现了任务书的要求。

# 第6章 总结与展望

经过几个月的工作，本次设计终于来到了结尾阶段，有必要对之前进行的工作进行回顾，总结，并对未来的工作做出合理的期望。

## 6.1 总结

本次设计的产品是一款智能OBD解析器，主要实现的功能是能够实时监测车辆的速度和油耗等信息，能够查询、清除车辆故障信息，并能通过蓝牙通信和Wi-Fi的方式把数据传送给上位机并显示出来。由于使用了MQTT协议，所以很容易在此基础上开发远程管理系统。本次设计解决了市面上OBD解析器数据接口不开放的难题，也大幅降低了具有远程数据传输能力的OBD解析器的成本。本次设计是获取车辆相关数据变得容易，这对自动驾驶和智慧交通等新领域的实现提供了数据支持。

通过本次设计我接触了很多新鲜的事物，学习了很多新的知识。我的主修专业是自动化，在学习中很少接触到车辆技术相关的东西，这次设计就让我有机会接触到OBD技术。不仅如此，这次设计让我拥有了独立开展小型项目的能力。比如说结合实际情况选择适合自己的方案：在以往的OBD解析器方案中，大多使用ELM327和TDA6X系列芯片，这对于没接触过OBD相关协议的我来说无疑是难度巨大，是一个短期内不能完成的任务。所以我选择了速锐得公司的EST527-miniS模组，该模组屏蔽了OBD底层通信协议，把从车辆ECU读取到的信息通过串口的方式明文发送，再使用串口就能够轻松的获取车辆的信息，而不用花费大量的时间和精力去学习底层的通信协议。学习了新的单片机开发平台：在以往的学习生活里，我接触得最多是STM32系列单片机，其次是51系列单片机，他们都有非常常见的开发环境，非常丰富的参考资料。而这次设计使用的主控平台是乐鑫公司的ESP32系列的单片机，其编程环境和编程方式有了很大的区别，我不得不从头开始学习嵌入式实时操作系统的相关知识。在学习的过程中遇到了很多困难，好在坚持不懈才克服了重重难关；大胆尝试，及时调整方案：本次设计的主控平台选用的是ESP32系列，而不是市面上使用较多的ESP8266，起初是因为其内部集成了Wi-Fi和蓝牙模组，我想用上ESP32的蓝牙功能，这样可以有效的缩小印刷电路板的体积，但在学习蓝牙编程的过程中遇到了很大的阻碍，资料太少，蓝牙工作流程非常复杂，继续使用下去的话会严重拖慢进度。所以我及时调整方案，选用了汇承公司的HC-02蓝牙模组，大幅的加快了开发进度。Wi-Fi通信方式之前从没有使用过，但是经过本次设计让我对Wi-Fi通信有了一定的了解，并在万物互联的今天使用了比较流行的MQTT传输协议，让我在互联网时代不至于掉队。

6.2 展望

本次的毕业设计虽然实现了任务书需求，但还是有很多可以改进的地方。在设计印刷电路板的时候就犯了很多的错误，今后会多学习和绘制原理图及PCB，让自己设计的印刷电路板布局更加合理，外形更加美观，体积更加小巧；此次的上位机功能比较简单，虽然上位机实现了数据显示和与下位机的交互，但并没有使用数据库对历史数据进行管理和分析。在大数据时代，从数据中获取更多的信息就是在创造价值，如果在上位机里添加了对数据的管理，并能在对数据进行处理后给出合适的建议，我相信这才能算得上是真正的智能产品；在完成本次设计时并没有考虑产品功耗的问题，如果能做好功耗的管理，这就让产品更人性化，更环保；考虑到自己的编程能力有限和保证任务进度只开发了PC端的上位机，但是现在的产品都在强调便携性，如果能开发出微信小程序或者智能手机端的应用的话，这次设计会更加完美。知识储备限制了本次毕业设计的品质。学海是无涯的，唯有不断的学习才能不迷失方向。

# 参考文献

1. 庞成立. 车载诊断系统(OBD)的发展与应用[J]. 汽车维修, 2011(09): 40-42.
2. 李志锐, 王青, 郑义, et al. 简述欧盟法规(EU)No 168/2013中摩托车OBD要求[J]. 小型内燃机与车辆技术, 2014, 43(05): 92-96.
3. 李满. 基于OBD系统的车辆状态监测及故障诊断[D]. 清华大学, 2016.
4. Mario F , Paul A J , Emilio X , et al. The Usefulness of Diesel Vehicle Onboard Diagnostics (OBD) Information[C]// Mechatronika, IEEE. IEEE, 2016: 2-3.
5. Malekian R, Moloisane N R, Nair L, et al. Design and Impleentation of a Wireless OBD II Fleet Management System[J]. IEEE Sensors Journal, 2016(02): 1-1.
6. 潘朋, 颜伏伍, 方茂东. OBD系统的现状及其发展趋势[J]. 交通节能与环保, 2007(05): 36-39.
7. 孟晓亮, 牛志刚, 展红亮. 基于ODB-Ⅱ的便携式汽车故障检测仪设计[J]. 中国仪器仪表, 2007(10): 60-63.
8. 匡书池. 基于4G的OBD远程监控系统设计[D]. 合肥工业大学, 2019.
9. 戴金池, 庞海龙, 俞妍, et al. 柴油车DPF系统的OBD故障诊断策略研究[J]. 车用发动机, 2020(01): 80-85.
10. 王云. 51单片机C语言程序设计教程[M]. 北京: 人民邮电出版社, 2018: 13.
11. 丁武锋, 庄严, 周春阳. MCU工程师炼成记: 我和MSP430单片机[M]. 北京: 机械工业出版社, 2013: 23-25.
12. 郭志勇. 嵌入式技术与应用开发项目教程（STM32版）[M]. 北京: 人民邮电出版社, 2019: 54-55.
13. Agus Kurniawan.Internet of Things Projects with ESP32[M]. Birmingham: Packt Publishing, 2019: 27-28.
14. 吴向畅, 石平, 郭文军, et al. 国六轻型车车载诊断系统开发[J]. 汽车工程学报, 2020, 10(01): 28-33.
15. 黄晓林, 曹玉华, 朱涛.基于“互联网+”的汽车在线故障监测诊断系统设计[J].现代机械, 2019(05):53-60.
16. 李志明, 檀永, 徐石明. STM32嵌入式系统开发实战指南:FreeRTOS与LwIPl联合移植[M]. 北京: 机械工业出版社, 2013:241.
17. 谢钧, 谢希仁. 计算机网络教程（第4版）[M]. 北京: 人民邮电出版社, 2014: 353-357.
18. 贾军营, 王月鹏, 王少华. 基于MQTT协议IM的研究和实现[J]. 计算机系统应用, 2015, 24(07): 9-14.
19. 马鑫龙, 艾冬生, 孙发鱼, et al. 基于MQTT的遥测数据无线传输[J]. 探测与控制学报, 2020, 42(02): 61-64.
20. 张瑜, 付智涛. 视频技术在连铸铸坯定位中的应用[J]. 现代工业经济和信息化, 2019, 9(04): 49-50.

# 致谢

“古之学者必有师”，在完成毕业设计（论文）的工作中，离不开老师的帮助。“师者，传道受业解惑也”，特别是老师的悉心指导，不厌其烦的为我解惑，扫平了我在毕业设计（论文）工作中的障碍，使得我的工作能顺利的开展下去。

我在这里向老师表达由衷的感谢，不仅是因为他在学业上对我的指导，还有他对待工作的态度--认真负责，计划有度，灵活变通。我相信这种态度在以后的工作中能给予我很大的帮助。

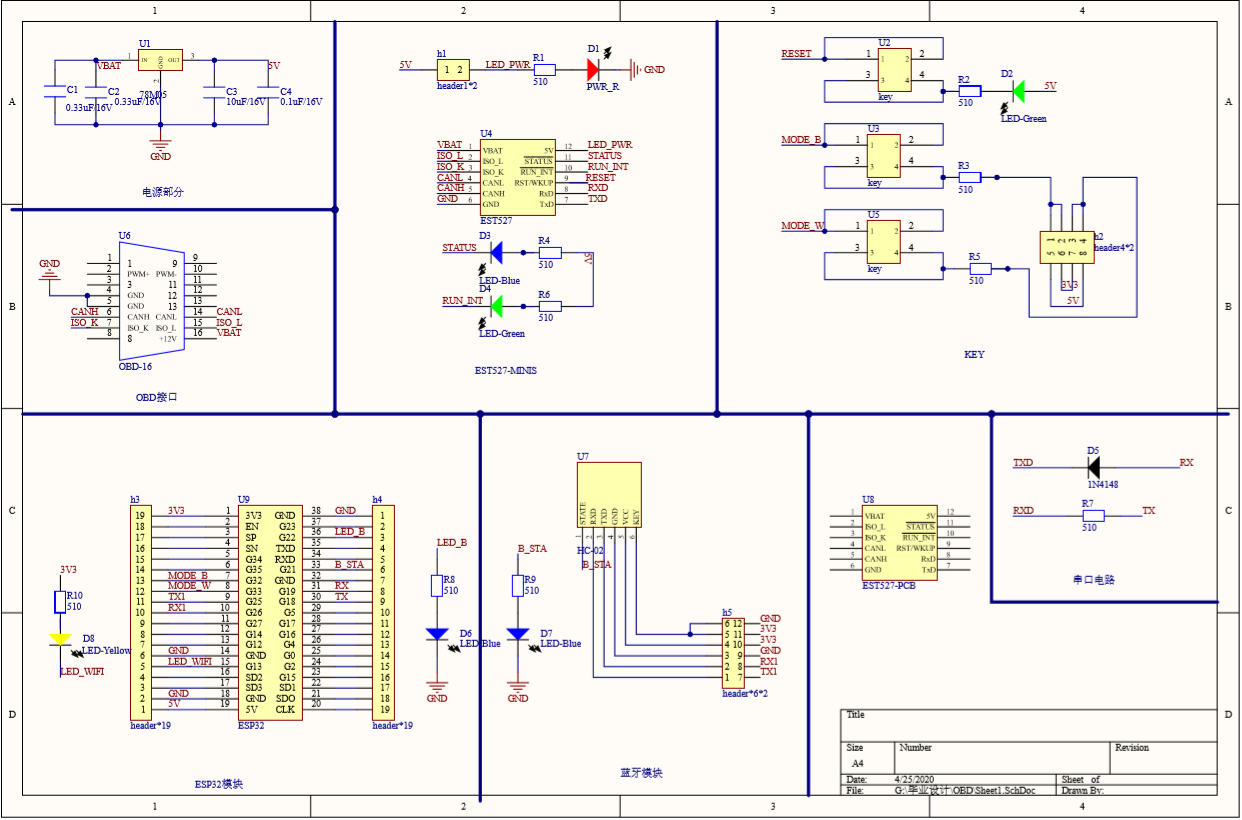
我还想感谢那些互联网上的网友，他们分享的知识解决了我很多的疑惑。

最后，我想感谢我的母校—重庆邮电大学，她不仅为我提供了一个学习的平台，还让我在这里遇见了很多重要的人。

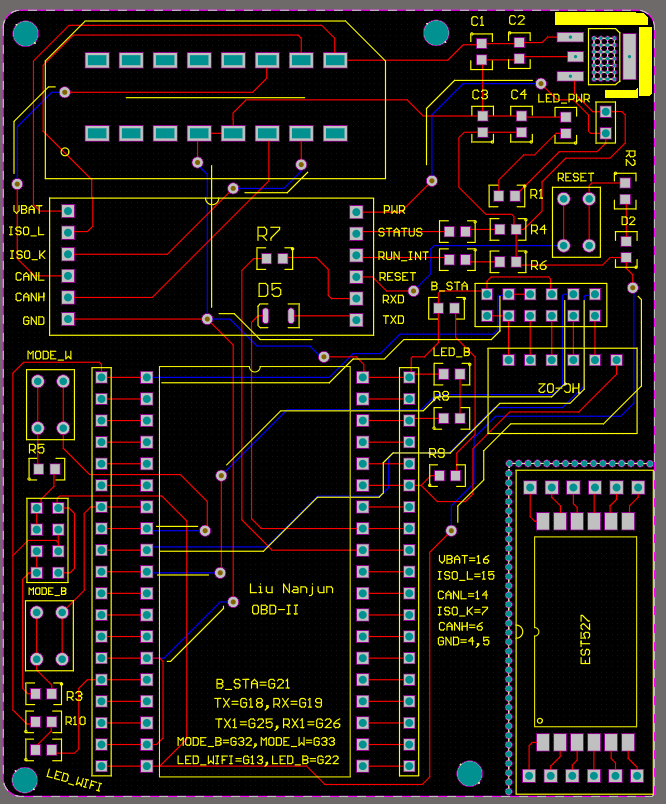
时光真如白驹过隙，行文于此，意味着毕业设计（论文）已经接近尾声了，也意味着大学的学习生活也要告一段落了。大学的生活虽然就要结束，但学习的脚步永远不会停歇，所谓“活到老，学到老”。无论是在校园，还是在离开校园的生活里都应该保持一颗对知识求知若渴的心。感谢那些给予我帮助的人儿！

# 附录A 原理图和程序代码

## 原理图



## PCB图



下位机程序

#include <stdio.h>

#include <stdint.h>

#include <stddef.h>

#include <string.h>

#include "esp\_wifi.h"

#include "esp\_system.h"

#include "nvs\_flash.h"

#include "esp\_event.h"

#include "tcpip\_adapter.h"

#include "protocol\_examples\_common.h"

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/semphr.h"

#include "freertos/queue.h"

#include "lwip/sockets.h"

#include "lwip/dns.h"

#include "lwip/netdb.h"

#include "esp\_log.h"

#include "mqtt\_client.h"

#include "driver/gpio.h"

#include "driver/uart.h"

#define MODE\_Bluetooth 32

#define MODE\_WIFI 33

#define LED\_WIFI 13

#define LED\_Bluetooth 21

#define Bluetooth\_State 23 //蓝牙模块状态引脚

#define UARTOneTX 16

#define UARTOneRX 17

#define UARTTwoTX 25

#define UARTTwoRX 26

#define BufferSize 512

static const char \*TAG1 = "MQTT\_EXAMPLE";

static esp\_mqtt\_client\_handle\_t client = NULL;

void taskLED\_Bluetooth(void \* para);

void taskLED\_WIFI(void \* para);

void taskWatingBluetoothConnect(void \* para);

void taskSwitchMode(void \* para);

void mqtt\_app\_start(void);

static void taskUARTOneBluetooth(void \*para)

{

//配置EST527与ESP32模块通信的串口

uart\_config\_t uart\_conf = {

.baud\_rate = 9600,//波特率

.data\_bits = UART\_DATA\_8\_BITS,//数据位

.stop\_bits = UART\_STOP\_BITS\_1,//停止位

.flow\_ctrl = UART\_HW\_FLOWCTRL\_DISABLE,//硬件流控制

.parity = UART\_PARITY\_DISABLE,//校验

};

//配置UART\_NUM\_1

uart\_param\_config(UART\_NUM\_1, &uart\_conf);

uart\_set\_pin(UART\_NUM\_1,UARTOneTX,UARTOneRX,UART\_PIN\_NO\_CHANGE, UART\_PIN\_NO\_CHANGE);

uart\_driver\_install(UART\_NUM\_1, BufferSize, 0, 0, NULL, 0);

uint8\_t \*data = (uint8\_t \*) malloc(BufferSize);//申请一部分内存空间

bzero(data,BufferSize);//进行写0操作，不然容易出现乱码

for(;;)

{

int len = uart\_read\_bytes(UART\_NUM\_1, data, BufferSize, 20 / portTICK\_RATE\_MS);

if(len) //len不为0时代表者有数据

{

printf("%s",(char \*)data);//测试输出

uart\_write\_bytes(UART\_NUM\_2,(char \*)data,len);

bzero(data,BufferSize);//进行写0操作，不然容易出现乱码

}

}

}

static void taskUartOneWIFI(void \*para)

{

//配置EST527与ESP32模块通信的串口

uart\_config\_t uart\_conf = {

.baud\_rate = 9600,//波特率

.data\_bits = UART\_DATA\_8\_BITS,//数据位

.stop\_bits = UART\_STOP\_BITS\_1,//停止位

.flow\_ctrl = UART\_HW\_FLOWCTRL\_DISABLE,//硬件流控制

.parity = UART\_PARITY\_DISABLE,//校验

};

//配置UART\_NUM\_1

uart\_param\_config(UART\_NUM\_1, &uart\_conf);

uart\_set\_pin(UART\_NUM\_1,UARTOneTX,UARTOneRX,UART\_PIN\_NO\_CHANGE, UART\_PIN\_NO\_CHANGE);

uart\_driver\_install(UART\_NUM\_1, BufferSize, 0, 0, NULL, 0);

uint8\_t \*data = (uint8\_t \*) malloc(BufferSize);//申请一部分内存空间

bzero(data,BufferSize);//进行写0操作，不然容易出现乱码

for(;;)

{

int len = uart\_read\_bytes(UART\_NUM\_1, data, BufferSize, 20 / portTICK\_RATE\_MS);

if(len)

{

printf("%s",(char \*)data); //测试输出

esp\_mqtt\_client\_publish(client, "EST527",(char \*)data, 0, 0, 0);

bzero(data,BufferSize);//进行写0操作，不然容易出现乱码

}

}

}

static void taskUARTTwo(void \*para)

{

uart\_config\_t uart\_conf = {

.baud\_rate = 9600,//波特率

.data\_bits = UART\_DATA\_8\_BITS,//数据位

.stop\_bits = UART\_STOP\_BITS\_1,//停止位

.flow\_ctrl = UART\_HW\_FLOWCTRL\_DISABLE,//硬件流控制

.parity = UART\_PARITY\_DISABLE,//校验

};

//配置UART\_NUM\_2

uart\_param\_config(UART\_NUM\_2, &uart\_conf);

uart\_set\_pin(UART\_NUM\_2,UARTTwoTX,UARTTwoRX,UART\_PIN\_NO\_CHANGE, UART\_PIN\_NO\_CHANGE);

uart\_driver\_install(UART\_NUM\_2, BufferSize, 0, 0, NULL, 0);

uint8\_t \*data = (uint8\_t \*) malloc(1024);//申请一部分内存空间

bzero(data,BufferSize);//进行写0操作，不然容易出现乱码

for(;;)

{

// Read data from the UART

int len = uart\_read\_bytes(UART\_NUM\_2, data, BufferSize, 50 / portTICK\_RATE\_MS);

if(len)

{

printf("%s",data);

uart\_write\_bytes(UART\_NUM\_1, ( char \*) data, len);//把从蓝牙模块 收到的消息通过UART1发送给EST527

bzero(data,1024);//进行写0操作，不然容易出现乱码

}

}

}

static esp\_err\_t mqtt\_event\_handler\_cb(esp\_mqtt\_event\_handle\_t event)

{

client = event->client;

int msg\_id;

int i=0;

switch (event->event\_id) {

case MQTT\_EVENT\_CONNECTED:

ESP\_LOGI(TAG1, "MQTT\_EVENT\_CONNECTED");

if (i==0)//避免重复订阅主题和创建UART1任务

{

msg\_id = esp\_mqtt\_client\_publish(client, "ESP32", "ESP32 is online!", 0, 0, 0);

ESP\_LOGI(TAG1, "sent publish successful, msg\_id=%d", msg\_id);

msg\_id = esp\_mqtt\_client\_subscribe(client, "UP", 0);//订阅上位机主题

ESP\_LOGI(TAG1, "sent subscribe successful, msg\_id=%d", msg\_id);

gpio\_set\_level(LED\_WIFI,0);

xTaskCreate(taskLED\_Bluetooth,"LED\_Bluetooth ",1024\*2,NULL,3,NULL);

xTaskCreate(taskUartOneWIFI,"ESP32，EST527 ",1024\*15,NULL,4,NULL);

}

i++;

break;

case MQTT\_EVENT\_DISCONNECTED:

ESP\_LOGI(TAG1, "MQTT\_EVENT\_DISCONNECTED");

break;

case MQTT\_EVENT\_SUBSCRIBED:

ESP\_LOGI(TAG1,"MQTT\_EVENT\_SUBSCRIBED,msg\_id=%d",event->msg\_id);

msg\_id = esp\_mqtt\_client\_publish(client, "/topic/qos0", "data", 0, 0, 0);

ESP\_LOGI(TAG1, "sent publish successful, msg\_id=%d", msg\_id);

break;

case MQTT\_EVENT\_UNSUBSCRIBED:

ESP\_LOGI(TAG1, "MQTT\_EVENT\_UNSUBSCRIBED, msg\_id=%d", event->msg\_id);

break;

case MQTT\_EVENT\_PUBLISHED:

ESP\_LOGI(TAG1, "MQTT\_EVENT\_PUBLISHED, msg\_id=%d", event->msg\_id);

break;

case MQTT\_EVENT\_DATA:

ESP\_LOGI(TAG1, "MQTT\_EVENT\_DATA");

printf("%s",event->data);

uart\_write\_bytes(UART\_NUM\_1,event->data,event->data\_len);

break;

case MQTT\_EVENT\_ERROR:

ESP\_LOGI(TAG1, "MQTT\_EVENT\_ERROR");

break;

default:

ESP\_LOGI(TAG1, "Other event id:%d", event->event\_id);

break;

}

return ESP\_OK;

}

static void mqtt\_event\_handler(void \*handler\_args, esp\_event\_base\_t base, int32\_t event\_id, void \*event\_data) {

ESP\_LOGD(TAG1, "Event dispatched from event loop base=%s, event\_id=%d", base, event\_id);

mqtt\_event\_handler\_cb(event\_data);

}

void mqtt\_app\_start(void)

{

ESP\_LOGI(TAG1, "[APP] Startup..");

ESP\_LOGI(TAG1, "[APP] Free memory: %d bytes", esp\_get\_free\_heap\_size());

ESP\_LOGI(TAG1, "[APP] IDF version: %s", esp\_get\_idf\_version());

ESP\_ERROR\_CHECK(nvs\_flash\_init());

tcpip\_adapter\_init();

ESP\_ERROR\_CHECK(esp\_event\_loop\_create\_default());

ESP\_ERROR\_CHECK(example\_connect());

esp\_mqtt\_client\_config\_t mqtt\_cfg = {

.host = "192.168.1.5",

.port = 1883,

.username = "client001",

.password = "psw001",

.client\_id = "ESP32",

.event\_loop\_handle = mqtt\_event\_handler\_cb,

};

#if CONFIG\_BROKER\_URL\_FROM\_STDIN

char line[128];

if (strcmp(mqtt\_cfg.uri, "FROM\_STDIN") == 0) {

int count = 0;

printf("Please enter url of mqtt broker\n");

while (count < 128) {

int c = fgetc(stdin);

if (c == '\n') {

line[count] = '\0';

break;

} else if (c > 0 && c < 127) {

line[count] = c;

++count;

}

vTaskDelay(10 / portTICK\_PERIOD\_MS);

}

mqtt\_cfg.uri = line;

printf("Broker url: %s\n", line);

} else {

ESP\_LOGE(TAG1, "Configuration mismatch: wrong broker url");

abort();

}

#endif /\* CONFIG\_BROKER\_URL\_FROM\_STDIN \*/

esp\_mqtt\_client\_handle\_t client = esp\_mqtt\_client\_init(&mqtt\_cfg);

esp\_mqtt\_client\_register\_event(client, ESP\_EVENT\_ANY\_ID, mqtt\_event\_handler, client);

esp\_mqtt\_client\_start(client);

}

void taskLED\_WIFI(void \*para)

{

for(;;)

{

gpio\_set\_level(LED\_WIFI,0);

vTaskDelay(250 /portTICK\_PERIOD\_MS);

gpio\_set\_level(LED\_WIFI,1);

vTaskDelay(250 /portTICK\_PERIOD\_MS);

}

}

void taskLED\_Bluetooth(void \*para)

{

for(;;)

{

gpio\_set\_level(LED\_Bluetooth,0);

vTaskDelay(250 /portTICK\_PERIOD\_MS);

gpio\_set\_level(LED\_Bluetooth,1);

vTaskDelay(250 /portTICK\_PERIOD\_MS);

}

}

void taskWatingBluetoothConnect(void \*para)

{

gpio\_config\_t io\_conf ;

io\_conf.intr\_type = GPIO\_INTR\_DISABLE;

io\_conf.mode = GPIO\_MODE\_INPUT;

io\_conf.pin\_bit\_mask = 1ULL<<Bluetooth\_State;

io\_conf.pull\_down\_en = 1;

io\_conf.pull\_up\_en = 0;

//配置蓝牙状态引脚

gpio\_config(&io\_conf);

for(;;)//以250ms间隔读取蓝牙状态

{

if(gpio\_get\_level(Bluetooth\_State))//蓝牙连接成功

{

xTaskCreate(taskUARTOneBluetooth,"ESP32与EST527通信任务",1024\*2,NULL,4,NULL);//创建ESP32与EST527通信任务

xTaskCreate(taskUARTTwo,"",1024\*4,NULL,4,NULL);//创建ESP32与HC-02模块通信任务

vTaskDelete(NULL);//删除本任务

}

vTaskDelay(250 /portTICK\_PERIOD\_MS);

}

}

void taskSwitchMode(void \*para)//通信方式选择任务，LED\_WIFI和LED\_Bluetooth会快速交替闪烁，同时读取MODE\_Bluetooth和MODE\_WIFI引脚的电平

{

gpio\_config\_t io\_conf ;

io\_conf.intr\_type = GPIO\_INTR\_DISABLE;

io\_conf.mode = GPIO\_MODE\_OUTPUT;

io\_conf.pin\_bit\_mask = 1ULL<<LED\_WIFI;

io\_conf.pull\_down\_en = 1;

io\_conf.pull\_up\_en = 0;

gpio\_config(&io\_conf);//LED\_WIFI配置

io\_conf.pin\_bit\_mask = 1ULL<<LED\_Bluetooth;

gpio\_config(&io\_conf);//LED\_Bluetooth配置

io\_conf.mode = GPIO\_MODE\_INPUT;

io\_conf.pin\_bit\_mask = 1ULL<<MODE\_Bluetooth;

gpio\_config(&io\_conf);//蓝牙模式按钮配置

io\_conf.pin\_bit\_mask = 1ULL<<MODE\_WIFI;

gpio\_config(&io\_conf);//WIFI模式按钮配置

for(;;)

{

gpio\_set\_level(LED\_Bluetooth,0);

gpio\_set\_level(LED\_WIFI,0);

if(gpio\_get\_level(MODE\_Bluetooth))//选择了蓝牙通信方式

{

printf("已选择蓝牙通信方式！\n等待蓝牙连接成功......\n");/

xTaskCreate(taskWatingBluetoothConnect,"",1024,NULL,4,NULL);

xTaskCreate(taskLED\_WIFI,"LED\_WIFI ",1024,NULL,3,NULL);

vTaskDelete(NULL);//删除本任务

}

if(gpio\_get\_level(MODE\_WIFI))//选择了WIFI通信方式

{

printf("已选择WIFI通信方式！\nWIFI连接中......\n");

mqtt\_app\_start();//开启wifi,并连接MQTT服务器

xTaskCreate(taskLED\_Bluetooth,"LED\_Bluetooth",1024,NULL,3,NULL)

vTaskDelete(NULL);//删除本任务

}

vTaskDelay(250 / portTICK\_PERIOD\_MS);//不精确延时250ms

gpio\_set\_level(LED\_Bluetooth,1);

gpio\_set\_level(LED\_WIFI,1);

if(gpio\_get\_level(MODE\_Bluetooth))//选择了蓝牙通信方式

{

printf("已选择蓝牙通信方式！\n等待蓝牙连接成功......\n");//该信息用于调试

xTaskCreate(taskWatingBluetoothConnect,"",1024,NULL,4,NULL);//创建等待蓝牙连接成功任务

xTaskCreate(taskLED\_WIFI,"LED\_WIFI闪烁任务",1024,NULL,3,NULL);//创建LED\_WIFI闪烁任务，用于提示程序正在运行

vTaskDelete(NULL);//删除本任务

}

if(gpio\_get\_level(MODE\_WIFI))//选择了WIFI通信方式

{

printf("已选择WIFI通信方式！\nWIFI连接中......\n"); //该信息用于调试

mqtt\_app\_start();//开启wifi,并连接MQTT服务器

xTaskCreate(taskLED\_Bluetooth,"LED\_Bluetooth闪烁任务",1024,NULL,3,NULL);//创建LED\_Bluetooth闪烁任务，用于提示程序正在运行

vTaskDelete(NULL);//删除本任务

}

vTaskDelay(250 / portTICK\_PERIOD\_MS);//不精确延时250ms

}

}

void app\_main()

{

printf("请选则通信方式......\n");

xTaskCreate(taskSwitchMode," ",1024\*6,NULL,4,NULL);//创建通信方式选择任务

}

上位机程序

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Management;

using System.IO.Ports; //串口

using OBDDemo3;

using MQTTnet;

using MQTTnet.Client;

using MQTTnet.Client.Options;

using System.Windows.Forms;

namespace OBDDemo3

{

public partial class Form1 : Form

{

bool isBluetooth = false; //蓝牙布尔值

bool isWIFI = false; //WIFI布尔值

bool ECUIsConnected = true; //ECU连接布尔值

private IMqttClient mqttClient = new MqttFactory().CreateMqttClient(); //MQTT客户端

string ECUOK = "$EST527,Connect ECU OK.\r\n";

string ECUNotOK = "$EST527,ECU NOT CONNECT.\r\n";

public Form1()

{

InitializeComponent();

serialPort1.DataReceived += new SerialDataReceivedEventHandler(ComReceiveData); //注册串口收到数据事件

timer1.Start(); //开启定时器

}

private void SerialPortSend(string x)

{

byte[] data = Encoding.Default.GetBytes(x);

string str = Encoding.Default.GetString(data);

serialPort1.Write(str);

}

/// <summary>

/// 串口接收事件

/// </summary>

/// <param name="sender"></param>

/// <param name="eventArgs"></param>

private void ComReceiveData(object sender, SerialDataReceivedEventArgs eventArgs)

{

byte[] receiveData = new byte[serialPort1.BytesToRead];

serialPort1.Read(receiveData, 0, receiveData.Length); DealWithData(Convert.ToString(Encoding.Default.GetString(receiveData)));

DealWithData("$OBD-RT,13.5,1426,38,0.00,42.6,82,5.68,8.02,10.42,80.65,0.75,6.15,2,1,3\r\n");

}

private void rtnWIFI\_CheckedChanged(object sender, EventArgs e)

{

isWIFI = true;

isBluetooth = false;

groupBox2.Enabled = true; //使能WIFI面板

groupBox1.Enabled = false; //失能蓝牙面板

if (serialPort1.IsOpen) //如果蓝牙处于连接状态，那么就断开蓝牙

{

serialPort1.Close(); //断开蓝牙

}

rtnBluetooth.ForeColor = Color.Black;

rtnWIFI.ForeColor = Color.Black;

}

private async void rtnBluetooth\_CheckedChanged(object sender, EventArgs e)

{

isBluetooth = true;

isWIFI = false;

groupBox1.Enabled = true;//使能蓝牙面板

groupBox2.Enabled = false;//失能WIFI面板

rtnBluetooth.ForeColor = Color.Black;

rtnWIFI.ForeColor = Color.Black;

cbxSerialPort.Items.Clear();

foreach (string item in ClassScanf.SerialPortScanf())

{

cbxSerialPort.Items.Add(item);

}

if (mqttClient.IsConnected)

{

await mqttClient.DisconnectAsync();//异步断开MQTT

}

}

/// <summary>

/// 定时器任务

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void timer1\_Tick(object sender, EventArgs e)

{

if (isBluetooth)//选择了蓝牙通信模式

{

if (serialPort1.IsOpen)//蓝牙处于连接状态

{

rtnBluetooth.ForeColor = Color.Green;

}

else //蓝牙处于断开状态

{

rtnBluetooth.ForeColor = Color.Black;

}

}

else //选择了WIFI通信模式

{

if (mqttClient.IsConnected)//MQTT处于连接状态

{

rtnWIFI.ForeColor = Color.Green;

}

else //MQTT处于断开状态

{

rtnWIFI.ForeColor = Color.Black;

}

}

if (textBoxErrorCode.Text == "0")

{

labelErrorCode.ForeColor = Color.Green;

}

else

{

labelErrorCode.ForeColor = Color.Red;

}

}

private void btnScanf\_Click(object sender, EventArgs e)

{

cbxSerialPort.Items.Clear();

foreach (string item in ClassScanf.SerialPortScanf())

{

cbxSerialPort.Items.Add(item);

}

}

Private void cbxSerialPort\_SelectedIndexChanged(object sender, EventArgs e)

{

serialPort1.Close();

string getPortNume = cbxSerialPort.Text.Substring(cbxSerialPort.Text.Length - 5, 4);//截取串口的端口号

serialPort1.PortName = getPortNume;

try

{

serialPort1.Open();

if (serialPort1.IsOpen)

{

MessageBox.Show("蓝牙模块LED长亮则连接成功，否则请重新选择！", "重要信息");

}

}

catch (Exception err)

{

MessageBox.Show(Convert.ToString(err));

}

}

/// <summary>

/// 按钮单击事件：断开蓝牙连接

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnBdisconnect\_Click(object sender, EventArgs e)

{

if (serialPort1.IsOpen)

{

serialPort1.Close();

}

}

private async void btnMQTTConnect\_Click(object sender, EventArgs e)

{

var options = new MqttClientOptionsBuilder()

.WithTcpServer(Convert.ToString(tbxIP.Text), Convert.ToInt32(tbxPort.Text))

.WithCredentials(Convert.ToString(tbxName.Text), Convert.ToString(tbxPSW.Text))

// .WithTls()

.WithClientId(Convert.ToString(tbxClientID.Text))

.WithCleanSession()

.Build();

await mqttClient.ConnectAsync(options);//异步连接MQTT

if (mqttClient.IsConnected == true)

{

await mqttClient.SubscribeAsync(new TopicFilterBuilder()

.WithTopic("ESP32")

.WithAtMostOnceQoS()

.Build()

);

await mqttClient.SubscribeAsync(new TopicFilterBuilder()

.WithTopic("EST527")//该主题用于接收OBD模块的消息

.WithAtMostOnceQoS()

.Build()

);

mqttClient.UseApplicationMessageReceivedHandler(E =>

{

Invoke(new Action(() =>

{

if (E.ApplicationMessage.Topic == "ESP32")

{

// textBox11.AppendText("ESP32已上线！\r\n");

// ESPOnline = true;

btnMQTTConnect.ForeColor = Color.Green;

}

// textBox11.AppendText($" payload = {Encoding.UTF8.GetString(E.ApplicationMessage.Payload)}" + "\r\n");

DealWithData(Encoding.UTF8.GetString(E.ApplicationMessage.Payload));

}));

});

}

}

private void DealWithData(string data)

{

if (data == ECUOK)//判断ECU是否连接成功

{

ECUIsConnected = true;

lblECU.ForeColor = Color.Green;//修改颜色

}

if (data == ECUNotOK)

{

ECUIsConnected = false;

lblECU.ForeColor = Color.Red;//修改颜色

}

this.Invoke(new Action(() =>

{

textBoxShow.AppendText(data);

if (ECUIsConnected)//ECU连接成功,进一步处理数据

{

ECUIsConnectFunction(data);

}

}));

}

private void ECUIsConnectFunction(string data)

{

string isRTData = data.Substring(0, 7);

if (isRTData == "$OBD-RT")//数据为实时数据流

{

string[] temp = data.Split(Convert.ToChar(","));//以","拆分字符串

\_ = this.Invoke(new Action(() =>

{

textBoxV.Text = temp[1];//电瓶电压

textBoxRPM.Text = temp[2];//发动机转速

textBoxSpeed.Text = temp[3];//车速

textBoxLoad.Text = temp[5];//发动机负荷

textBoxCold.Text = temp[6];//冷却液温度

if (temp[3] == "0")

{

textBoxCast.Text = temp[7] + "(L/h)";//瞬时油耗

}

else

{

textBoxCast.Text = temp[7] + "(L/100Km)";

}

textBoxCast2.Text = temp[11];//本次耗油量

textBoxErrorCode.Text = temp[13];//当钱故障码数量

}));

}

}

private async void btnMQTTDisconnect\_Click(object sender, EventArgs e)

{

if (mqttClient.IsConnected)

{

await mqttClient.DisconnectAsync();//异步断开MQTT连接

}

}

private void buttonClean\_Click(object sender, EventArgs e)

{

textBoxShow.Clear();//清空显示框文本

}

private void buttonInquiry\_Click(object sender, EventArgs e)

{

if (textBoxErrorCode.Text != "0")//故障码数量不为0

{

if (textBoxSpeed.Text != "0")//目前处于行驶状态

{

this.Invoke(new Action(() =>

{

textBoxShow.AppendText("待车辆处于怠速时，请重新尝试！\r\n");

}));

}

else //处于怠速状态

{

if (isBluetooth && serialPort1.IsOpen)

{

SerialPortSend("AT400\r\n");//查询故障码

}

if (isWIFI && mqttClient.IsConnected)

{

MqttPublish("AT400\r\n");//查询故障码

}

}

}

}

private async void MqttPublish(string data)

{

var message = new MqttApplicationMessageBuilder()

.WithTopic("UP")

.WithPayload(data)

.WithAtMostOnceQoS()

.WithRetainFlag()

.Build();

await mqttClient.PublishAsync(message);

}

private void buttonClear\_Click(object sender, EventArgs e)

{

if (textBoxErrorCode.Text != "0")//故障码数量不为0

{

if (textBoxSpeed.Text != "0")//目前处于行驶状态

{

this.Invoke(new Action(() =>

{

textBoxShow.AppendText("待车辆处于怠速时，请重新尝试！\r\n");

}));

}

else //处于怠速状态

{

if (isBluetooth && serialPort1.IsOpen)

{

SerialPortSend("AT401\r\n");//清楚故障码

}

if (isWIFI && mqttClient.IsConnected)

{

MqttPublish("AT401\r\n");//清楚故障码

}

}

}

}

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

string[] errATArray = comboBox1.Text.Split(Convert.ToChar(","));

string errCode = "AT"+errATArray[1] + "\r\n"; //拼接命令

if (isBluetooth && serialPort1.IsOpen)

{

SerialPortSend(errCode);//清楚故障码

}

if (isWIFI && mqttClient.IsConnected)

{

MqttPublish(errCode);//清楚故障码

}

}

private void labelWebsite\_Click(object sender, EventArgs e)

{

System.Diagnostics.Process.Start("https://www.obd-data.com/cn/");

}

# 附录B 英文翻译

A Study on Remote On-Line Diagnostic System for Vehicles by Integrating the Technology of OBD, GPS, and 3G

Abstract—This paper presents a remote on-line diagnostic system for vehicles via the use of On-Board Diagnostic (OBD), GPS, and 3G techniques. The main parts of the proposed system are on-board computer, vehicle monitor server, and vehicle status browser. First, the on-board computer can obtain the location of deriver and vehicle status from GPS receiver and OBD interface, respectively. Then on-board computer will connect with the vehicle monitor server through 3G network to transmit the real time vehicle system status. Finally, vehicle status browser could show the remote vehicle status including vehicle speed, engine rpm, battery voltage, engine coolant temperature, and diagnostic trouble codes. According to the experimental results, the proposed system can help fleet managers and car knockers to understand the remote vehicle status. Therefore, this system can decrease the time of fleet management and vehicle repair due to the fleet managers and car knockers who find the diagnostic trouble messages in time.

Keywords—Diagnostic Trouble Code (DTC), Electronic Control Unit (ECU), Global Position System (GPS), On-Board Diagnostic (OBD).

1. INTRODUCTION

ON-BOARD Diagnostic (OBD) regulations require passenger cars to report diagnostic information and standardized fault codes for malfunctions detected by the OBD system [1], [2]. As early as in 1985, the California Air Resources Board (CARB) enacted an ordinance that demands the vehicles outfitted with the OBD system. The European Union followed to legislate for a compulsory installation of OBD system in vehicles. The Taiwan (R.O.C.) Environmental Protection Administration also implemented a mandatory regulation in January 2008 for its Phase 4 Emission Standards against all the gasoline vehicles in Taiwan. Therefor all of the vehicles must be outfitted an OBD system from then on as an effort to put the pollution from vehicles into surveillance. Any vehicles without this device mounted shall not be issued the license plate to hit the road. Thus it can be seen that OBD system will be the standard outfit to all of the vehicles in the future. The OBD system can always monitor the running condition of engine. Once there is a malfunctioning element that controls the emission of exhaust, the OBD system will turn on the Malfunction Indicator Lamp (MIL) (as shown in the Fig. 1) in the in-car dashboard to urge the driver to fix it immediately. The element for emission control of exhaust gas can resume its normal operation as soon as possible in order to avoid driving a malfunctioned vehicle continuously that leads to a higher fuel consumption and pollution emission. When the OBD system detects a malfunction, OBD regulations require the Electronic Control Unit (ECU) of the vehicle to save a standardized Diagnostic Trouble Code (DTC) about the information of malfunction in the memory. An OBD Scan Tool for the servicemen can access the DTC from the ECU quickly and accurately to confirm the malfunctioning characteristics and location in accordance with the prompts of DTC that shortens the service time largely. Moreover, currently the number of item for the real-time driving status that OBD can monitor is as high as up to 80 items and more including the vehicle speed, engine rpm, engine coolant temperature, battery voltage, and etc.

Fig. 1 Variety of icons for Engine Malfunction Indicator Lamps

Although OBD can shorten the service time of vehicles, a demand for the immediateness and mobility relevant to the vehicular diagnostics is growing increasingly. Therefore, if it is possible that the DTC can be delivered to the server through the mobile communication, not only the OBD Scan Tool can read the message, but the servicemen also can inquire the real-time malfunction message from a remote vehicle. Thus the deficiency of immediateness and the mobility can be made up consequently [3]–[5]. In addition to the need of concerning the immediateness and the mobility of vehicular diagnostics, currently the logistic industry, bus transportation service providers, taxi service providers, etc. have an uneasy task for their commercial fleet management. For this reason, this study constructed a remote on-line diagnostic system comprised of an on-board computer, a vehicle monitor server and a vehicle status browser. The remote on-line diagnostic system is for the fleet manager to understand the real-time driving status and the OBD DTC of each car on a real-time basis via 3G network. Accordingly, the fleet manager can inform the driver how to take care of the situation in a telecommunication manner. In the future, the fleet manager can not only know the driving locations of thereof fleet, but also acquire the real-time driving status of each car. Hence, in addition to mend the deficiency in immediateness and mobility, the system benefits the commercial fleet largely as well.

1. SYSTEM CONFIGURATION

The remote on-line diagnostic system proposed in this paper includes three major modules, namely the on-board computer, vehicle monitor server, and vehicle status browser. Fig. 2 shows the integrated configuration of these three modules. It follows from Fig. 2 that the on-board computer is responsible for collecting the information of driving, including vehicle speed, engine rpm, battery voltage, engine coolant temperature, OBD DTC, location of vehicle, and etc. This information will be processed with digital encoding and transmitted back through a 3G network to the vehicle monitor server. The driving information will be recorded for user’s immediate or future retrieval. The vehicle status browser for driving real-time information is exclusively developed for the remote on-line diagnostic system and to allow the user to synchronously browse the real-time information, e.g., vehicle speed, engine rpm, battery voltage, engine coolant temperature, OBD DTC, location of vehicle, and etc. The following three sections describe these three modules separately.



Fig. 2 Configuration of remote on-line diagnostic system

1. ON-BOARD COMPUTER

On-board computer is the most important key module brought up in this paper. Its function is to acquire the information of vehicle speed, engine rpm, battery voltage, engine coolant temperature, OBD DTC, and GPS signal. The on-board computer will transmit these digital bit-streams back to the vehicle monitor server through the 3G network. Fig. 3 is the block diagram of the on-board computer proposed in this paper. The system is mainly comprised of OBD-II to RS232 adapter, GPS receiver, encoder, 3G module, and etc. The operating commands of user include the start, run, network setup, and etc. Since the majority of 3G modules support the Windows Operating System only, in order to let the system have a smooth operation, the on-board computer mentioned in this paper is practically implemented by a notebook computer with Windows operating platform.

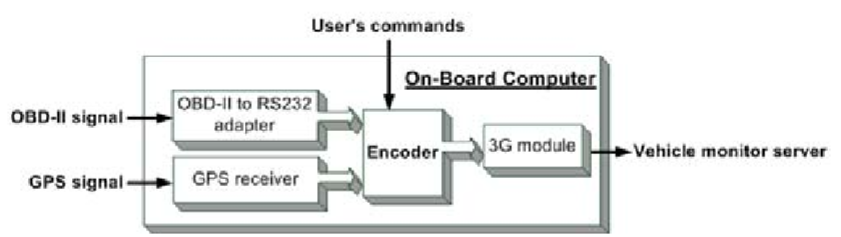
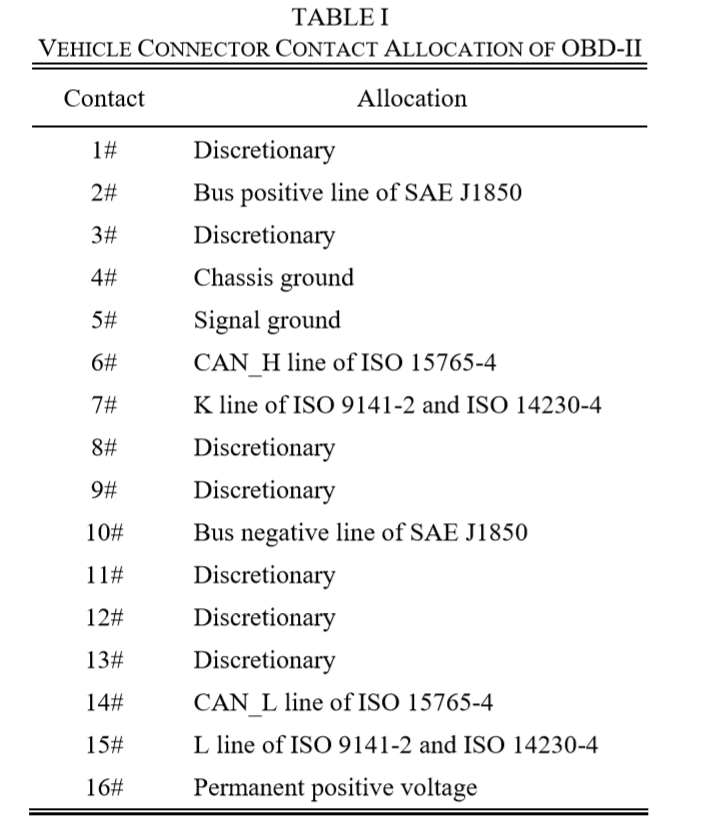


Fig. 3 Block diagram of system for on-board computer

A. OBD-II In the very beginning of 1980s, most car manufacturers in Europe, USA and Japan had incepted producing their injection-engine cars with the OBD system equipped in a bid to monitor the running status of engine. However, the early OBD had a serious problem, that is, these car makers made thereof OBD system incompatible with one another. As an effort to rid off this defect of an incompatible OBD system, the CARB undertook to devise a new OBD system. For the purpose of differentiating it from the used system at early stage, this newly formulated system is called the OBD-II (The second generation of On-Board Diagnostic system). The biggest betterment of OBD-II is its standardization, that is, just one set of instrument is able to perform the diagnosis and scan against variety of vehicles. In addition to the diagnosis against a totally failed element for pollution control of emission, OBD-II is able to further carry out a diagnostic against the pollution of emission exhausted from those aged or partially malfunctioned elements. When the signal from the circuit of electronic control system appears an abnormality, the OBD-II will diagnose to determine if this is a malfunctioned part. Here the abnormality is out of the range of normal deviation while this abnormal phenomenon has not disappeared within a specified amount of time duration. At this moment, the malfunction indicator lamp (MIL) will be turned on. Meanwhile, the monitor will save this malfunction in the memory inside ECU in a code form. Thus the saved DTC can be retrievable through an OBD-II Scan Tool. The vehicle and OBD-II Scan Tool connectors each are capable of accommodating 16 contacts. Nine contacts of the 16 contacts have thereof fixed function, and the function for the rest of contacts is left to the discretion of the vehicle

manufacturer [6]. Table I tabulates the function of OBD-II contacts in details.



There are five codes in total to represent the OBD-II malfunction. The first code is an English alphabet to stand for the established malfunction system. The remaining four codes are digits; the second code indicates the meaning of malfunction formulated by SAE/ISO or customized by the vehicle manufacturer; the third code shows the area of vehicle system; the remaining two codes represent the definition of the subject malfunction (as shown in Fig. 4) [1].

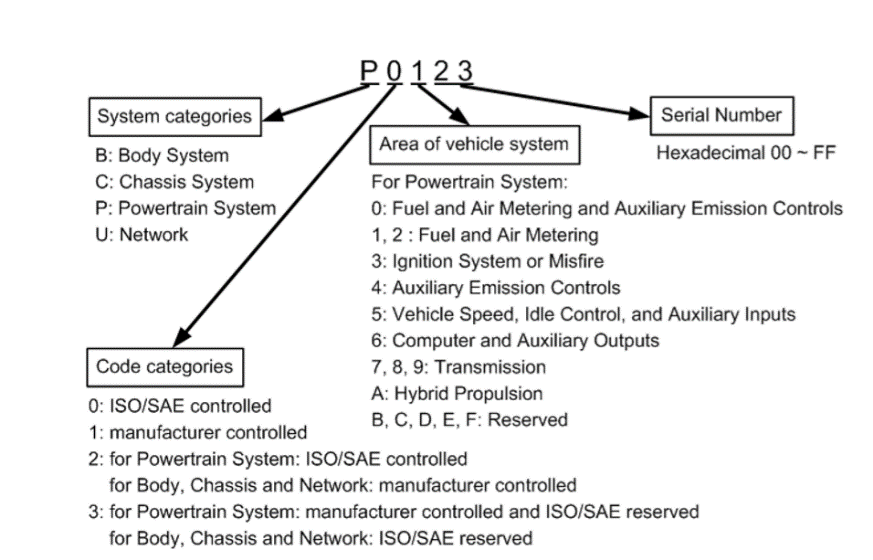


Fig. 4 Definition of OBD-II Diagnostic Trouble Code

B. OBD-II to RS232 Adapter Since the market-sold computer has not been equipped with the OBD-II interface, an OBD-II to RS232 adapter was fabricated to accommodate with the on-board computer for a practical implementation to acquire the real-time driving status.

OBD-II is international standardized to be applicable to the following communication protocols: SAE J1850 PWM, SAE J1850 VPW, ISO 9141-2, ISO 14230-4 KWP, and ISO 15765-4 CAN. The finished product of OBD-II to RS232 adapter that is practically implemented in this paper is shown in Fig. 5. In Fig. 5, the port at left side is to connect the OBD-II interface so that the OBD-II signals can be converted into RS232 electrical signals via the conversion circuit of OBD-II to RS232 adapter in the middle section and output from the port at right side. Now the computer is able to take the advantage of this RS232 interface to communicate with the OBD-II.

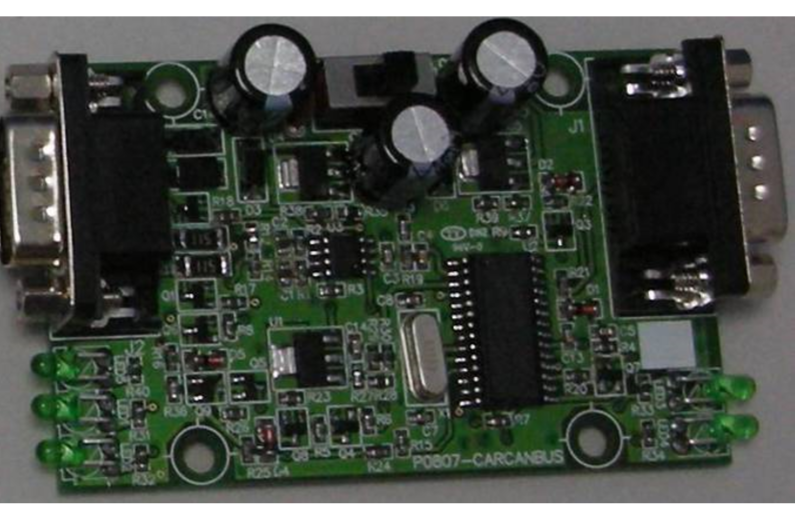


Fig. 5 OBD-II to RS232 adapter

C. GPS Receiver GPS receiver can receive signals from 8-12 sets of GPS satellite at the same time, and the satellite signals to be received by the receiver include: Coordinated Universal Time, ephemeris data, almanac data, Coarse/Acquisition code, etc. GPS receiver can receive, process, and transform the information into time, latitude, longitude, velocity, orientation, altitude, Estimated Position Error, and etc. For example, a sentence amongst the GPS information sentences is as follow: $GPGGA, 055730.367, 2238.2122, N, 12017.7504, E, 1, 06, 7.0, 133.9, M, 10.0, M, 0.0, 0000\*74; therein, the denotations of respective data are explained below [7]: GGA (Global Positioning System Fix Data): Time, position, and fix related data for a GPS receiver. 055730.367: UTC time format, fix was taken at 05:57:30 UTC. 2238.2122, N: Latitude format, it is 22 degree 38.2122’ of north latitude. 12017.7504, E: Longitude format, it is 120 degree 17.7504’ of east longitude. 1: Fix quality, the measured indicator 1 indicates that the information has made a fix using GPS. 06: The number of satellites was tracked. 7.0: Horizontal dilution of position: 0.5 m to 99.9 m, the measured value is 7.0 m. 133.9, M: Altitude above mean sea level, the measured altitude is 133.9 m.

10.0, M: Height of geoid above World Geodetic System 1984 (WGS 84) ellipsoid, the measured height is 10.0 m. 0.0: Time in seconds since last Differential GPS (DGPS) update, the measured value 0.0 shows that this GPS receiver did not use DGPS fix. 0000: DGPS station ID number. \*74: Checksum. Then, these data will be transmitted to a Geographical Information System (GIS), such as the PAPAGO® software of Maction Technologies, Inc., for it to pinpoint and display the position. Since the SiRF Star III chips are capable of promptly and accurately receiving the GPS signals, this paper used the GPS receiver with the SiRF Star III chipset mounted to collect the positioning signals.

D. Encoder The function of encoder is designed to encode and integrate the GPS signals and OBD data in accordance with the preset transmission format. These data will be transmitted to the vehicle monitor server via 3G so that the vehicle monitor server can decode the transmitted data in accordance with the predefined transmission format to acquire the subject vehicle’s speed, engine rpm, battery voltage, engine coolant temperature, OBD DTC, and the GPS coordinates for the position of vehicle. In order to cope with the updating frequency for GPS to receive the signals once per second, this paper also set the frequency at once per second to acquire the OBD DTC and the real-time driving status. Meanwhile, this paper defined a transmission format as follows: On-board computer ID number| GPS data | OBD DTC | vehicle speed, engine rpm, battery voltage, engine coolant temperature With the aim of saving the transmission of bits, this study acquired only the GGA contents amongst the GPS signals as the positioning data and transmitted the DTC, speed, rpm, voltage, and temperature in a decimal system after processing the analytics. For instance: On-board computer ID number: 1043 GGA data from GPS: $GPGGA, 055730.367, 2238.2122, N, 12017.7504, E, 1, 06, 7.0, 133.9, M, 10.0, M, 0.0, 0000\*74 DTC: P0123 Vehicle speed: 57km/hr Engine rpm: 1649 rpm Battery voltage: 13.375V Engine coolant temperature: 95OC The data for transmission after being encoded by the encoder are: 1043 | $GPGGA, 055730.367, 2238.2122, N, 12017.7504, E, 1, 06, 7.0, 133.9, M, 10.0, M, 0.0, 0000\*74 | P0123 | 57, 1649, 13.375, 95.

1. VEHICLE MONITOR SERVER

The driving information acquired by the on-board computer, including the vehicle speed, engine rpm, battery voltage, engine coolant temperature, OBD DTC, and GPS coordinates, will be processed through digital encoding and transmitted to the vehicle monitor server via a 3G network as a provision for the user to retrieve the driving real-time information or to retrieve the information in the future. The vehicle monitor server will save the driving information from the on-board computer into the Access 2003 database. Then the vehicle status browser can retrieve the vehicle information, e.g., speed, engine rpm, battery voltage, coolant temperature, OBD DTC, and location of vehicle. In addition, the vehicle monitor server will take the advantage of Geographical Information System to send out a map file produced according to the GPS coordinates. A. Access 2003 database Microsoft Access 2003 is a relational database. Its user interface is simple, ease of operation and compatible to ANSI SQL-92 standard. Since this system is a prototyping system developed only for authenticating the feasibility of system configuration, therefore the Access 2003 database is adequate for storing the information of this system. This paper established a data table with the “DiagnosticData” titled to cover the vehicle-driving information, while the field name, data type, and description are shown in Fig. 6. Therein, the field names of primary key are “OnBoardComputerID” and “RecordTime” that cannot be repeated. The longitude and latitude in the data table are adopted a format of WGS84 with an accuracy up to the sixth digit after the decimal point. That is, the accuracy can be within 10 cm approximately when converted into distance. Besides the foregoing field, the data table has more fields to further cover the OBD DTC, vehicle speed, engine rpm, battery voltage, and engine coolant temperature.

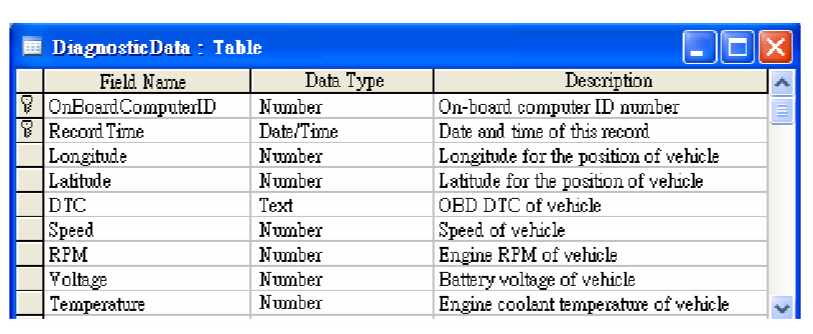


Fig. 6 Data table details relevant to the driving information

B. Geographical Information System

Geographical Information System is a science that integrates the geography, mathematics, land surveying, and computer science and serves as a computer system that can be used for input, storage, inquiry, analysis, and display of geographic data. The Geographical Information System is comprised of the following elements: maps, spatial information, database engine, analytical tools, graphic display techniques, operators, and decision-makers involved, etc. Its scope of application includes the visible objects above and below the surface of earth, such as the traffic transportation, land utilization, exploration of geology, and etc. The PaPaGO! SDK is a Geographical Information System developed by the Maction Technologies. It can be executable in server, desk computer, and PDA. With the future expandability taken into consideration, the PaPaGO! SDK was adopted to undertake an implementation. This paper used the PaPaGO! SDK search engine to locate the driving location with red-star pinpointed and displayed (as shown in Fig. 7), transformed it into a map file and sent it to the vehicle status browser.



Fig. 7 Mapping display of driving location

1. VEHICLE STATUS BROWSER

Vehicle status browser is exclusively used for the remote on-line diagnostic system so that the user can synchronously browse the driving date/time, OBD DTC, vehicle speed, engine rpm, battery voltage, engine coolant temperature, and driving location, etc. The basic operation in this regard is to log on the vehicle monitor server, choose the vehicle to be monitored, choose to browse the real-time information or historical record. Then it can receive the driving information from the vehicle monitor server with the driving date/time, OBD DTC, vehicle speed, engine rpm, battery voltage, engine coolant temperature shown in text form and the driving location displayed in map form in the screen (as shown in Fig. 8). Fig. 8 demonstrates a schematic operation of vehicle status browser. In the figure, the Car ID and driving date/time are shown in the left upper block, the vehicle speed, engine rpm, battery voltage, and engine coolant temperature are shown in the upper middle block, the OBD DTC and corresponding malfunction message are shown in the upper right block, and the driving location at the time is pinpointed with a red star in the major block of the figure.

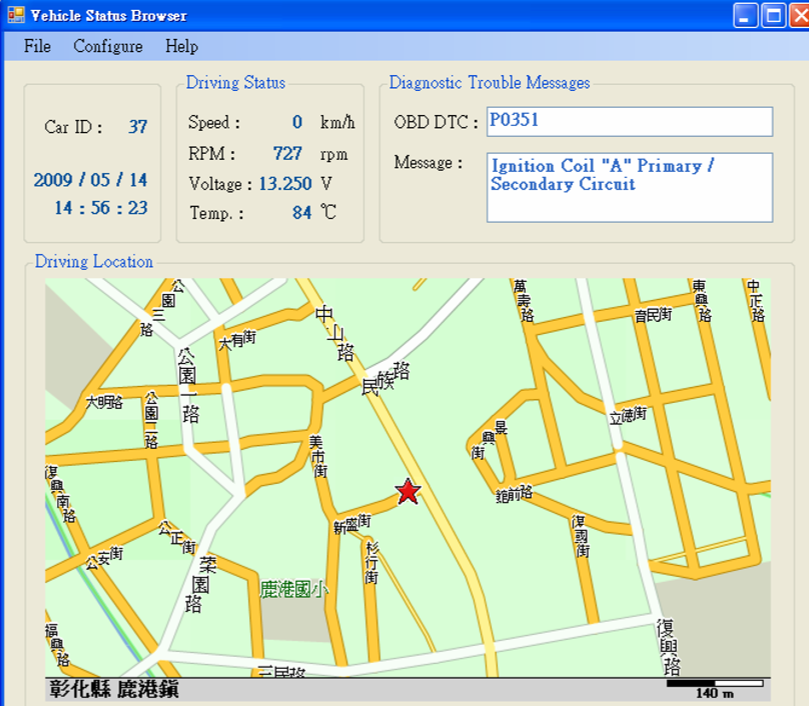


Fig. 8 Schematic operation of vehicle status browser

1. RESULTS OF EXPERIMENT

This paper used a notebook computer configured with Intel Core 2 Duo T5500 1.66GHz CPU, 2GB RAM, Window XP® Professional SP3 operating system as the on-board computer, a BU-353 (USB interface, SiRF Star III chips) from Globalsat Technology Corporation as the GPS receiver, a network interface card of Huawei E220 HSDPA USB from Chunghwa Telecom as the 3G network module, and fabricated an OBD-II to RS232 adapter and took the C# language to implement the on-board computer system. Regard for the vehicle monitor server, an IBM server configured with Intel Xeon® E5335 2.00GHz CPU, 2GB RAM, Windows Server 2003® R2 SP2 operating system is used together with the use of a Microsoft Access 2003 to store the driving information, a PaPaGO! SDK software from Maction Technologies Inc. to generate the map, a 6M/6M FTTB optical network from Chunghwa Telecom for networking, an IP-fixed receiving on-board computer to link with the vehicle status browser, and a C# language to program the functions of vehicle monitor server. The vehicle status browser is likewise programmed by C# language, so the computer is able to execute this browser once installed the Microsoft .NET Framework V2.0. This paper used a desk computer configured with Intel Pentium 4 2.4GHz CPU, 1GB RAM, and Window XP® Professional SP3 operating system to perform the vehicle status browser. This study took a Ford Focus 1.8 vehicle as the test subject, wherein, the OBD-II port was located at the left lower part of the steering wheel (the left part as shown in the Fig. 9) to connect the OBD-II to RS232 adapter (the right part as shown in the Fig. 9). Thus, when obtained the OBD DTC, etc. real-time driving status, and GPS coordinates, the on-board computer sent the received information to the vehicle monitor server via 3G network.



Fig. 9 OBD port (left part), OBD-II to RS232 adapter (right part)

In order to authenticate the feasibility of the proposed system, this study undertook an offline testing against an experimental car parked along the roadside for thereof ignition coil inside the engine chamber (as shown in the Fig. 10). After disconnecting the line of the said ignition coil, for the safety concern, the car was left at a still status (vehicle speed was 0). Then, the vehicle status browser was taken to check up the information of car; a DTC P0351 appeared to indicate that Ignition Coil "A" Primary/Secondary Circuit. Meanwhile, some information relevant to the car was shown as well: driving date/time, vehicle speed, engine rpm, battery voltage, engine coolant temperature, and driving location (as shown in the Fig. 11).



Fig. 10 Offline testing of ignition coil

Through the aid of Remote On-Line Diagnostic System, the management personnel of an automotive maintenance/repair shop or commercial fleet is able to remotely learn the current driving information and whether there is a malfunction message or not via network instead of going to the field.

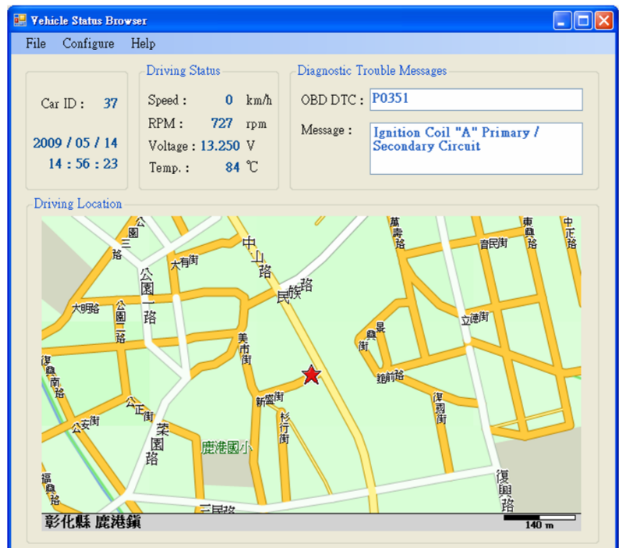


Fig. 11 Vehicle status browser

1. CONCLUSION

This paper has successfully integrated the 3G mobile network, GPS and OBD system to develop a remote on-line diagnostic system while the complete system is comprised of three major elements: on-board computer, vehicle monitor server, and vehicle status browser. According to the results of experiments, the remote on-line diagnostic system proposed in this paper is able to allow the user to know the real-time driving status, malfunction message, and location of a moving car at a remote place. The remote on-line diagnostic system proposed by this paper offers a highly applicable value in the fleet management, remotely automotive diagnosis or the initiative service notification of a maintenance/repair shop. In the future, the linking between the vehicle status browser and vehicle monitor server in the system will use the HTTP (Hypertext Transfer Protocol) protocol. Thus, without mounting the vehicle status browser, the user can execute the Web Browser from a computer, mobile phone or handheld device to inquire about the real-time information of a car in moving. Consequently, the convenience of using the remote on-line diagnostic system is further upgraded.

ACKNOWLEDGMENT This work was funded by the Department of Industrial Technology of MOEA (Ministry of Economic Affairs), Taiwan, R.O.C.

1. REFERENCES

[1] Diagnostic Trouble Code Definitions Equivalent to ISO/DIS 15031-6, SAE Standard J2012, 2002.

[2] E/E Diagnostic Test Modes — Equivalent to ISO/DIS 15031-5, SAE Standard J1979, 2002.

[3] C. E. Lin and C. C. Li, “A Real Time GPRS Surveillance System using the Embedded System,” AIAA J. Aerosp. Comput., Inf. Commun., vol. 1, no. 1, pp. 44-59, Jan. 2004.

[4] C. E. Lin, C. C. Li, and S. F. Tai, “G3 technology for intelligent transportation system,” in Proc. FISITA 2004 World Automotive Congress, Barcelona, 2004, F2004I071.

[5] C. E. Lin, C. C. Li, S. H. Yang, S. H. Lin; C. Y. Lin, “Development of On-Line Diagnostics and Real Time Early Warning System for Vehicles,” in Proc. IEEE Sensors for Industry Conference, Houston, 2005, pp. 45-51.

[6] Diagnostic Connector Equivalent to ISO/DIS 15031-3, SAE Standard J1962, 2002. [7] NMEA data. Available: <http://gpsinformation.org/dale/nmea.htm>

翻译：

集成汽车技术的车辆使用OBD，GPS和3G技术的远程在线诊断系统的研究

摘要--本文介绍了一种通过使用车载诊断（OBD），GPS和3G技术的车辆远程在线诊 抽象 — 本文介绍了一种通过使用车载诊断（OBD），GPS和3G技术的车辆远程在线诊 抽象 — 本文介绍了一种通过使用车载诊断（OBD），GPS和3G技术的车辆远程在线诊 断系统。拟议系统的主要部分是车载计算机，车辆监控服务器和车辆状态浏览器。首先， 车载计算机可以分别从GPS接收器和OBD接口获取派生工具的位置和车辆状态。然后车载 计算机将通过3G网络与车辆监控服务器连接，以传输实时车辆系统状态。最后，车辆状态 浏览器可以显示远程车辆状态，包括车辆速度，发动机转速，电池电压，发动机冷却液温 度和诊断故障代码。根据实验结果，提出的系统可以帮助车队管理者和敲门者了解远程车辆的状态。

关键词 — 诊断故障代码（DTC），电子控制单元（ECU），全球定位系统（GPS），车载诊断（OBD）。

I 引言

车载诊断法规要求乘用车辆报告诊断信息和标准化故障码。早 在1985年，加利福尼亚空气资源委员会（CARB）颁布了一项条例，要求配备O BD系统的车辆。欧洲联盟随后立法强制在车辆中安装OBD系统。台湾（ROC） 环境保护总署还于2008年1月对台湾的所有汽油车实施了其第四阶段排放标准的 强制性规定。因此，从那时起，所有车辆都必须配备OBD系统来对车辆排放污染进行监控。未安装此设备的任何车辆均不得获得牌照上路。因此 可以看出，OBD系统将成为未来所有车辆的标准装备。

OBD系统始终可以监控发动机的运行状况。一旦存在控制排气排放的故障元件 ，OBD系统将打开车内仪表板中的故障指示灯（MIL）（如图1所示），以敦促驾 驶员立即对其进行修理。废气排放控制元件可以尽快恢复其正常运行，以避免连 续行驶故障车辆，从而导致更高的燃油消耗和污染排放。当OBD系统检测到故障 时，OBD法规要求车辆的电子控制单元（ECU）将有关故障信息的标准化诊断故 障代码（DTC）保存在内存中。用于维修人员的OBD扫描工具可以根据DTC的提 示快速而准确地从ECU访问DTC，以确认故障特征和位置，从而大大缩短了维修 时间。此外，当前OBD可以监视的用于实时驾驶状态的项目数多达80个，并且包括车速，发动机转速，发动机冷却液温度，电池电压等。



图1 发动机故障指示灯的各种图标

尽管OBD可以缩短车辆的维修时间，但与车辆诊断相关的即时性和便携性的需求却在不断增长。因此，如果可以通过移动通信将DTC传送到服务器，则不仅O BD扫描工具可以读取该消息，而且维修人员还可以从远程车辆查询实时故障消息 。因此，可以弥补即时性和移动性的不足[3]–[5]。

除了需要考虑即时性和在车辆诊断的机动性方面，当前的物流业，公共汽车运输服务提供商，出租车服务提供商等在其商业车队管理方面任务艰巨。因此，本研究构建了一个远程在线 诊断系统，该系统由车载计算机，车辆监控服务器和车辆状态浏览器组成。远程 在线诊断系统使车队经理可以通过3G网络实时了解每辆车的实时驾驶状态和OB D DTC。相应地，车队管理者可以通知驾驶员如何以电信的方式照顾情况。将来，车队管理者不仅可以知道其车队的行驶位置，而且可以获取每辆车的实时行驶状态。

II 系统配置

本文提出的远程在线诊断系统包括三个主要模块，即车载计算机，车辆监控服务器和车辆状态浏览器。图2显示了这三个模块的集成配置。从图2可以看出，车 载计算机负责收集驾驶信息，包括车速，发动机转速，电池电压，发动机冷却液 温度，OBD DTC，车辆位置等。这些信息将被处理。带有数字编码，并通过3G 网络传输回车辆监控服务器。驾驶信息将被记录下来，以供用户立即或将来检索 。专为远程在线诊断系统开发的用于驾驶实时信息的车辆状态浏览器，并允许用户至同步浏览实时信息，例如车速，发动机转速，电池电压，发动机冷却液温度，O BD DTC，车辆位置等。以下三个部分分别描述了这三个模块。

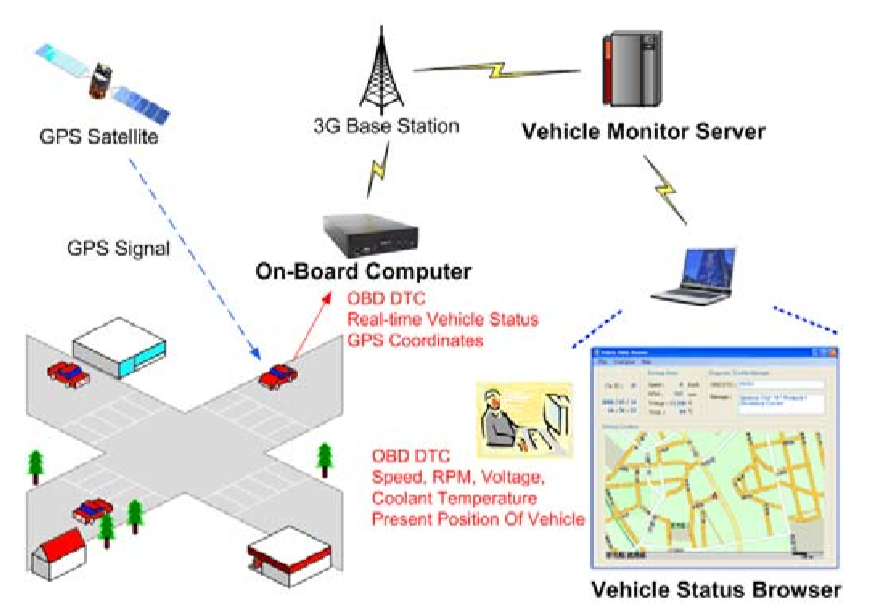


图2 远程在线诊断系统的配置

III 车载计算机

车载计算机是在本文中提出最重要的关键模块。其功能是获取车辆速度，发动机转速，电池电压，发动机冷却液 温度，OBD DTC和GPS信号的信息。车载计算机将这些数字比特流通过3G网络传输回车辆监控服务器。图3是本文提出的车载计算机的框图。该系统主要由OB D-II到RS232适配器，GPS接收器，编码器，3G模块等组成。用户的操作命令包括启动，运行，网络设置等。由于大多数3G模块都支持Windows。仅操作系统，为了使系统平稳运行，本文中提到的车载计算机实际上是由具有Windows操作平 台的笔记本计算机实现的。

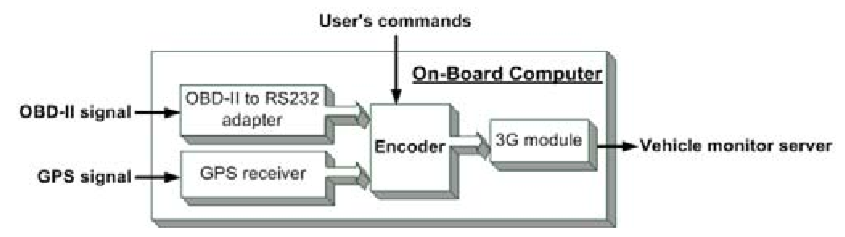


图3 车载计算机系统框图

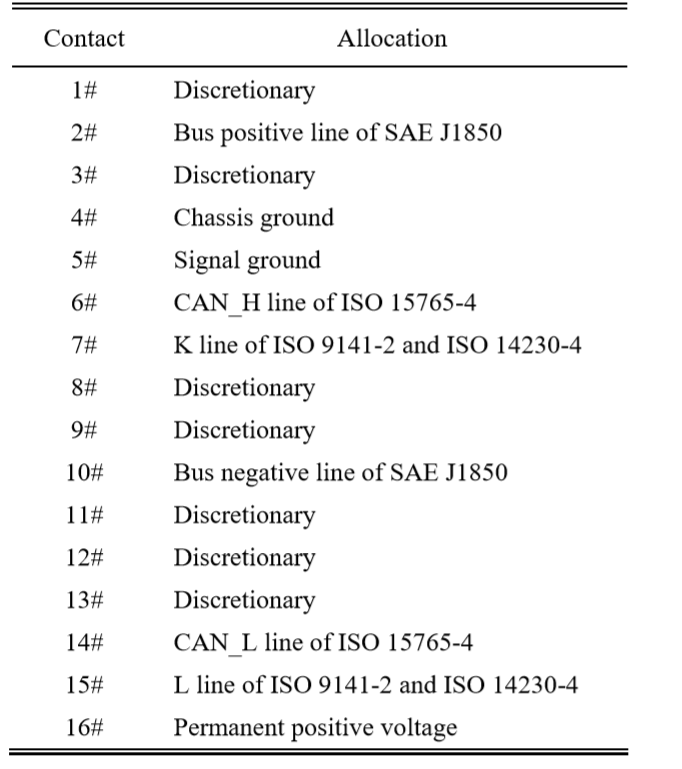
1. OBD-II

在1980年代初期，欧洲，美国和日本的大多数汽车制造商开始接受配备OBD 系统的喷射发动机汽车的生产，以监视发动机的运行状态。但是，早期的OBD存 在一个严重的问题，即这些汽车制造商使它们的OBD系统彼此不兼容。为了消除 不兼容的OBD系统的缺陷，CARB致力于设计新的OBD系统。为了与早期使用的 系统区分开来，此新制定的系统称为OBD-II（第二代车载诊断系统）。

OBD-II的最大改进是它的标准化，即只有一套仪器能够执行诊断并扫描各种车 辆。除了诊断排放污染完全失效的元素，OBD-II还能进一步诊断那些老化或部分 故障的元素排放的排放污染。当来自电子控制系统电路的信号出现异常时，OBD -II将诊断以确定这是否是故障部件。此处，异常不在正常偏差范围内，而此异常现象在指定的持续时间内未消失。此时，故障指示灯（MIL）将点亮。与此同时，监 视器将以代码形式将此故障保存在ECU内部的存储器中。因此，可以通过OBD-II 扫描工具检索保存的DTC。

车辆和OBD-II扫描工具连接器均能够容纳16个触点。16个触点中的9个触点具有固定功能。其余触点的功能由车辆制造商自行决定[6]。表I详细列出了OBD-II触点的功能。

表一 OBD-II接口引脚定义



共有五个代码代表OBD-II故障。第一个代码是代表已建立的故障系统的英文字 母。其余四个代码是数字；第二个代码表示由SAE / ISO制定或由汽车制造商定制 的故障含义；第三个代码显示了车辆系统的区域；其余两个代码代表主题故障的 定义（如图4所示）[1]。

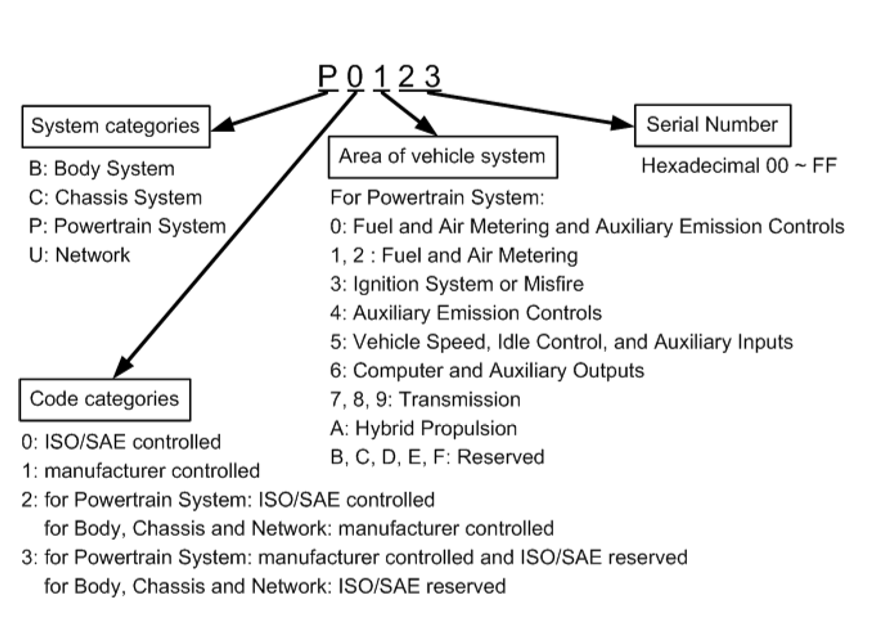


图4 OBD-II诊断故障代码的定义

B． OBD-II转RS232适配器

OBD-II转RS232适配器 由于市场上出售的计算机尚未配备OBD-II接口，因此制造了OBD-II至RS232适 配器以与车载计算机配合使用，以实际实施以获取实时驾驶状态。

OBD-II是国际标准化的，适用于以下通信协议：SAE J1850 PWM，SAE J18 50 VPW，ISO 9141-2，ISO 14230-4 KWP和ISO 15765-4 CAN。本文实际实现 的OBD-II到RS232适配器的成品如图5所示。在图5中，左侧的端口用于连接OBD -II接口，以便可以将OBD-II信号通过OBD-II的转换电路将RS232电信号转换为中 间部分的RS232适配器，并从右侧的端口输出。现在，计算机可以利用此RS232 接口与OBD-II进行通信。

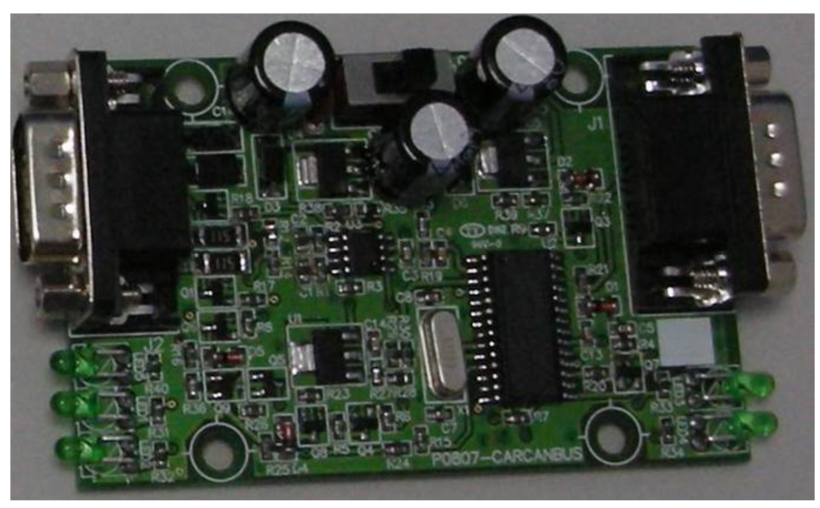


图5 OBD-II到RS232适配器

1. GPS接收器

GPS接收器可以同时接收8-12套GPS卫星的信号，并且接收器接收的卫星信号 包括：协调世界时，星历数据，年历数据，粗略/获取代码等。GPS接收器可以接 收，处理信息并将其转换为时间，纬度，经度，速度，方向，高度，估计位置误差等。

例如，收到的GPS信号如下：$GPGGA, 055730.367, 2238.2122, N, 12017.7504, E, 1, 06, 7.0, 133.9, M, 10.0, M, 0.0, 0000\*74; therein, the denotations of respective data are explained below [7]:数据解释如下：

GGA（全球定位系统修复数据）：GPS接收器的时间，位置和修复相关数据。

055730.367：UTC时间格式，修复时间为05:57:30

2238.2122，N：纬度格式，为22度38.2122' 北纬。

12017.7504，E：经度格式，为120度17.7504' 东经。

1：修复质量，被测指示器1表示信息已使用GPS修复。

06：跟踪了卫星数量。

7.0：水平稀释位置：0.5 m至99.9 m， 测量值为7.0 m。

133.9，M：高于平均海平面的海拔高度，实测 高度为133.9 m。

10.0，M：大地水准面在世界大地测量系统上方的高度 1984年（WGS 84）椭球，测得的高度为10.0 m。

0.0：自上次差分GPS（DGPS）以来的时间（以秒为单位） 更新，测量值0.0表示此GPS接收器未使用DGPS修复。

0000：DGPS站ID号。

\* 74：校验和

然后，这些数据将被传输到地理信息系统（GIS），例如Maction Technologi es，Inc.的PAPAGO®软件，以精确定位并显示位置。由于SiRF Star III芯片能够 迅速而准确地接收GPS信号，因此本文使用安装了SiRF Star III芯片组的GPS接 收器来收集定位信号。

D． 编码器

编码器 编码器的功能旨在按照预设的传输格式对GPS信号和OBD数据进行编码和整合 。这些数据将通过3G传输到车辆监控服务器，以便车辆监控服务器可以根据预定 义的传输格式对传输的数据进行解码，以获取目标车辆的速度，发动机rpm，电池 电压，发动机冷却液温度，OBD DTC，以及车辆位置的GPS坐标。

为了应对GPS每秒接收一次信号的更新频率，本文还设置了每秒一次的频率来 获取OBD DTC和实时驾驶状态。同时，本文定义了一种传输格式，如下所示：

车载计算机ID号| GPS数据| OBD DTC | 车速，发动机转速，电池电压，发动 机冷却液温度。

为了节省比特的传输，本研究仅获取了GPS信号中的GGA内容作为定位数据 ，并在处理完分析后以十进制形式传输了DTC，速度，rpm，电压和温度。例如 ：车载计算机ID号：1043 来自GPS的GGA数据：$ GPGGA，055730.367、2238.2122，N， 12017.7504，E，1，06，7.0，133.9，M，10.0，M，0.0，0000 \* 74 DTC：P0123

车速：57km / hr发动机转速：1649 rpm电池 电压：13.375V发动机冷却液温度：95 Ø C 电压：13.375V发动机冷却液温度：95 Ø C 电压：13.375V发动机冷却液温度：95 Ø C

编码器编码后的传输数据为：1043 | $ GPGGA，055730.367，2238.2122， N，12017.7504， E，1，06，7.0，133.9，M，10.0，M，0.0，0000 \* 74 | P0123 | 57， 1649、13.375、95。

IV车辆监控服务器

车载计算机获取的驾驶信息，包括车速，发动机转速，电池电压，发动机冷却 液温度，OBD DTC和GPS坐标，将通过数字编码进行处理，并通过3G网络传输到车辆监控服务器以便获取驾驶实时数据。。车辆监控器服务器会将驾驶信息从车载 计算机保存到Access 2003数据库中。然后，车辆状态浏览器可以检索车辆信息，例如速度，发动机rpm，电池电压，冷却液温度，OBD DTC和车辆位置。另外，车辆监控服务器将利用地理信息系统的优势发送出根据GPS坐标生成的地图文件。

1. Access 2003数据库

Microsoft Access 2003是一个关系数据库。它的用户界面简单，易于操作并且 兼容ANSI SQL-92标准。由于此系统是仅用于验证系统配置可行性的原型系统， 因此Access 2003数据库足以存储该系统的信息。

本文建立数据与表的“DiagnosticData”标题涵盖车辆驾驶字段名称，数据类型和描述如图6所示。其中，主键的字段名称是“OnBoardCom puterID”和“RecordTime”，不能重复。数据表中的经度和纬度采用WGS84格式， 精度最高至小数点后第六位。即，当转换为距离时，精度可以在大约10 cm之内 。除上述字段外，数据表还有更多字段，以进一步涵盖OBD DTC，车速，发动机 rpm，电池电压和发动机冷却液温度。

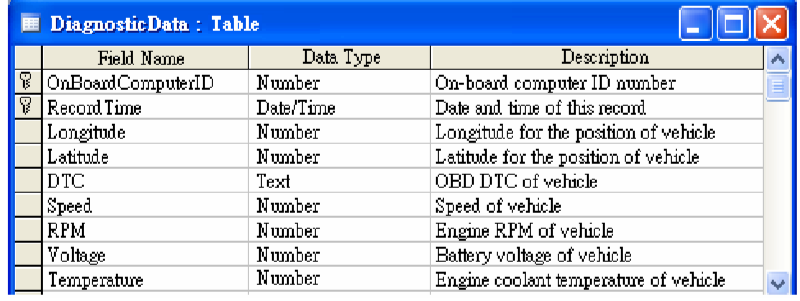


图6 与驾驶信息有关的数据表详细信息

B． 地理信息系统

地理信息系统 地理信息系统是一门综合了地理，数学，土地测量和计算机科学的科学，并且是可用于输入，存储，查询，分析和显示地理数据的计算机系统。地理信息系统由以下元素组成：地图，空间信息，数据库引擎，分析工具，图形显示技术，操 作员和所涉及的决策者等。其应用范围包括地面上方和下方的可见对象。地球， 例如交通运输，土地利用，地质勘探等。

PaPaGO！SDK是Maction Technologies开发的地理信息系统。它可以在服务 器，台式计算机和PDA中执行。考虑到随着未来的扩展，PaPaGO！采用SDK进行实施。本文使用了PaPaGO！SDK搜索引擎可 定位并显示带有红色星标的驾驶位置（如图7所示），然后将其转换为地图文件并 将其发送至车辆状态浏览器。



图7 行驶位置的映射显示

V车辆状态浏览器

车辆状态浏览器专门用于远程在线诊断系统，因此用户可以同步浏览驾驶日期 /时间，OBD DTC，车速，发动机转速，电池电压，发动机冷却液温度和驾驶位 置等。这方面的基本操作是登录车辆监控服务器，选择要监控的车辆，选择浏览 实时信息或历史记录。然后，它可以从车辆监控服务器接收驾驶信息，包括在屏 幕上以文本形式显示的驾驶日期/时间，OBD DTC，车速，发动机转速，电池电压，发动机冷却液温度以及以地图形式显示的驾驶位置（如图8所示）。

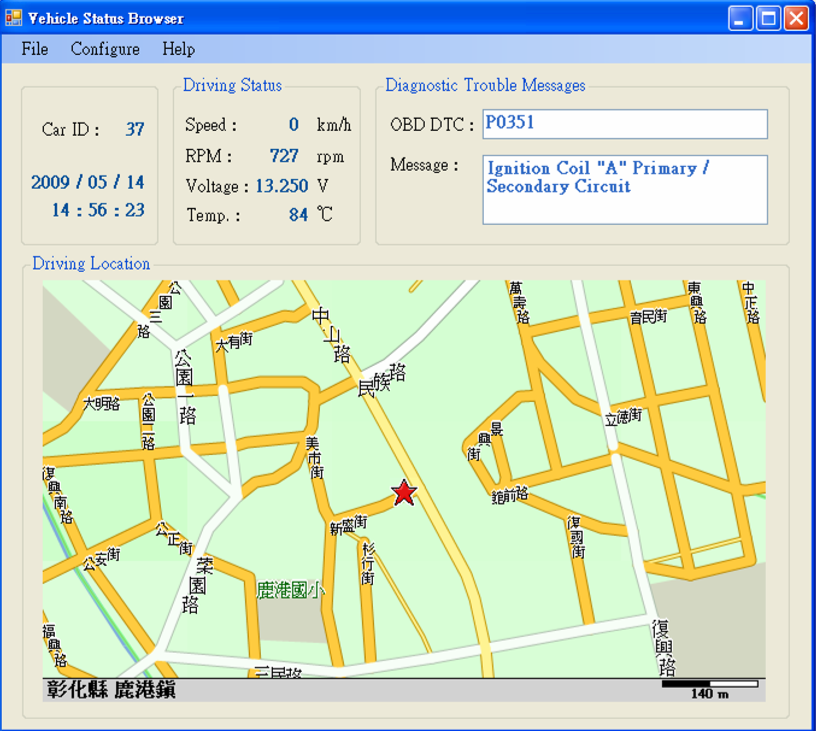


图8 车辆状态浏览器的示意图操作

图8示出了车辆状态浏览器的示意性操作。在该图中，左上方的方框中显示了 汽车ID和驾驶日期/时间，中上方的方框中显示了车速，发动机转速，电池电压和 发动机冷却液温度，OBD DTC和相应的故障消息分别为如右上图所示，当时的行驶位置在图中的大图框中用红色星标指出。

VI试验结果

本文使用的笔记本电脑配置了Intel Core 2 Duo T5500 1.66GHz CPU，2GB R AM，WindowXP®Professional SP3操作系统作为车载计算机，Globalsat Techn ology的BU-353（USB接口，SiRF Star III芯片）公司作为GPS接收器，将中华电 信的华为E220 HSDPA USB的网络接口卡作为3G网络模块，并制作了OBD-II至 RS232适配器，并采用C＃语言实现了车载计算机系统。

关于车辆监控器服务器，将使用配置有IntelXeon®E5335 2.00GHz CPU，2G B RAM，Windows Server2003®R2 SP2操作系统的IBM服务器以及Microsoft Ac cess 2003来存储驾驶信息，即PaPaGO ！Maction Technologies Inc.的SDK软件 生成地图，中华电信提供6M / 6M FTTB光网络进行联网，IP固定接收车载计算机 与车辆状态浏览器链接，以及C＃语言对功能进行编程车辆监控服务器。

车辆状态浏览器同样由C＃语言编程，因此一旦安装了Microsoft .NET Frame work V2.0，计算机便能够执行此浏览器。本文使用配置有Intel Pentium 4 2.4GH z CPU，1GB RAM和WindowXP®Professional SP3操作系统的台式计算机来执 行车辆状态浏览器。

车辆状态浏览器同样由C＃语言编程，因此一旦安装了Microsoft .NET Frame work V2.0，计算机便能够执行此浏览器。本文使用配置有Intel Pentium 4 2.4GH z CPU，1GB RAM和WindowXP®Professional SP3操作系统的台式计算机来执 行车辆状态浏览器。

这项研究以福特福克斯1.8车辆为测试对象，其中OBD-II端口位于方向盘的左 下部（如图9所示的左部分），以将OBD-II连接至RS232适配器（右侧部分如图9 所示）。因此，当获得OBD DTC等实时驾驶状态和GPS坐标时，车载计算机通过 3G网络将接收到的信息发送给车辆监控服务器。



图9 OBD端口（左侧），OBD-II至RS232适配器（右侧）

为了验证所提出系统的可行性，本研究针对在路边停放的实验车在发动机室内 的点火线圈进行了离线测试（如图10所示）。在断开所述点火线圈的线路之后， 出于安全考虑，使汽车保持静止状态（车辆速度为0）。然后，使用车辆状态浏览 器检查汽车信息；DTC P0351出现，表明点火线圈“ A”是初级/次级电路。同时， 还显示了与汽车有关的一些信息：行驶日期/时间，车速，发动机转速，电池电压 ，发动机冷却液温度和行驶位置（如图11所示）。



图10 点火线圈的离线测试

借助远程在线诊断系统，汽车维修/修理店或商业车队的管理人员可以通过网络来远程了解当前的驾驶信息以及是否有故障消息，而无需去现场。

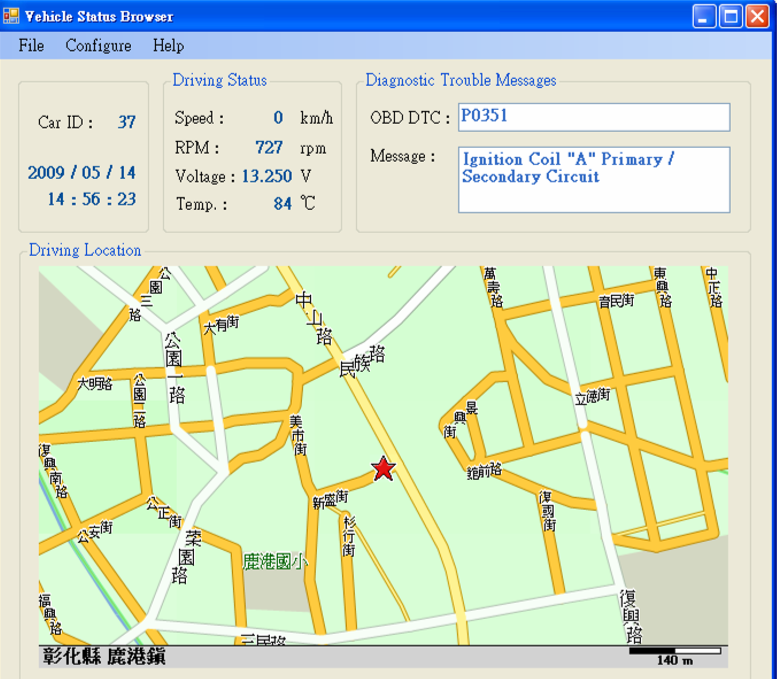


图11 车辆状态浏览器

VII 总结

本文成功地集成了3G移动网络，GPS和OBD系统，以开发一个远程在线诊断 系统，而整个系统则由三大要素组成：车载计算机，车辆监控服务器和车辆状态 浏览器。根据实验结果，本文提出的远程在线诊断系统能够使用户了解实时驾驶 状态，故障消息以及在偏远地区行驶中的汽车的位置。本文提出的远程在线诊断 系统在车队管理，远程汽车诊断或维修/修理车间的主动服务通知中具有很高的应 用价值。

将来，系统中车辆状态浏览器和车辆监控服务器之间的链接将使用HTTP（超 文本传输​​协议）协议。因此，用户无需安装车辆状态浏览器，就可以从计算机， 移动电话或手持设备执行Web浏览器，以查询正在行驶的汽车的实时信息。因此 ，进一步提高了使用远程在线诊断系统的便利性。

致谢

这项工作由台湾经济部（MOEA）工业技术部资助。

参考文献

[1] Diagnostic Trouble Code Definitions Equivalent to ISO/DIS 15031-6, SAE Standard J2012, 2002.

[2] E/E Diagnostic Test Modes — Equivalent to ISO/DIS 15031-5, SAE Standard J1979, 2002.

[3] C. E. Lin and C. C. Li, “A Real Time GPRS Surveillance System using the Embedded System,” AIAA J. Aerosp. Comput., Inf. Commun., vol. 1, no. 1, pp. 44-59, Jan. 2004.

[4] C. E. Lin, C. C. Li, and S. F. Tai, “G3 technology for intelligent transportation system,” in Proc. FISITA 2004 World Automotive Congress, Barcelona, 2004, F2004I071.

[5] C. E. Lin, C. C. Li, S. H. Yang, S. H. Lin; C. Y. Lin, “Development of On-Line Diagnostics and Real Time Early Warning System for Vehicles,” in Proc. IEEE Sensors for Industry Conference, Houston, 2005, pp. 45-51.

[6] Diagnostic Connector Equivalent to ISO/DIS 15031-3, SAE Standard J1962, 2002. [7] NMEA data. Available: http://gpsinformation.org/dale/nmea.htm

1. https://docs.espressif.com/projects/esp-idf/zh\_CN/stable/get-started [↑](#footnote-ref-1)