An Embedded Device Utilising GPRS for Communications

Kevin B. Mayer, Dr Ken Taylor CPEng, MIEAust

CSIRO Mathematical and Information Sciences Division & The Australian National University
GPO Box 664, Canberra, ACT, 2601
Fax: (02) 6216 7111

Abstract—**The task of creating a cheap and flexible device for supervisory control and data acquisition (SCADA) was the goal of this project. The device is intended to operate as part of a previously defined architecture. By using a very simple and cheap microprocessor, and relying on an existing communications infrastructure – the mobile phone network, the task was accomplished to the point that a functional system now exists. GPRS provides us with a relatively cheap, continuous connection to the Internet, and a web interface hides complexity from the user.**

I. INTRODUCTION

An architecture has been defined [1] for building complex supervisory control and data acquisition (SCADA) applications utilising shared network infrastructure which communicates with simple distributed devices that monitor and act on the world. Cost is minimised by using devices with only minimal processing power and supporting these devices with network based services. This allows complex control and data acquisition applications to be implemented at lower cost than is possible using current techniques. Existing infrastructure supports communication between devices and supporting services using several methods provided by the GSM mobile phone network. It is envisaged other communication mediums particularly those that support TCP/IP will also be suitable for connecting devices to supporting services. Having designed an architecture and provided some network services a cheap, generic and reprogrammable device is required for interacting with the physical world. The device should conform to the following specification:

- *Utilise General Packet Radio Service (GPRS) for communications*. The existing infrastructure used other communication methods available through mobile phones. Using the caller ID of a phone ringing a bank of phone numbers is the most basic method of communication and currently the cheapest, as there is no communication cost if the called number is not answered. SMS text messages and circuit switched GSM calls where the mobile phone acts like a modem have also been employed. GPRS provides an ‘always on’ communications channel without incurring time based charges. Data is packet switched and charges are volume based. GPRS, costing about $25 per megabyte of data transferred[1], is expensive where large volumes of data are to be transferred but can cost no more than a few cents per day for the volumes of data that are required to be transmitted for many applications. GPRS enables a constant TCP/IP connection with the device (for monitoring, controlling or acquiring data) whilst avoiding the prohibitive costs associated with a permanent circuit switched connection.
- *General purpose*. The range of applications the infrastructure supports is broad, and rather than creating a purpose built device for each application, we aimed to create a generic device which could be reconfigured for any particular application as required. The work required to apply the device to a given application is limited to attaching analogue, digital or serial I/O to the spare I/O pins on the device, and programming the required logic into the device.
- *Reprogrammable over the air*. Rather than physically connecting the device to programming hardware (via a serial cable or any other method), it was decided that the device should be ‘over the air’ programmable in situ. This allows code to be easily updated without special equipment when physically separated from the device and where there are tens or even hundreds of devices that all need a similar code upgrade.
- *Simple to use*. Interacting with devices using a web browser provides a familiar interface and avoids the

Kevin. B. Mayer is a student at the Faculty of Engineering and Information Technology, Australian National University, Canberra, ACT 0200, Australia (telephone: +61409688267, e-mail: kmayer@layer9.com).

Ken. Taylor, is with the Division of Mathematical and Information Sciences, CSIRO, GPO Box 664, Canberra ACT 2601 (e-mail: ken.taylor@csiro.au).

ICITA2002 ISBN: 1-86467-114-9

[1] Telstra fee where less than 500Kb transferred in a session

requirement for installing specialist software. Messages should be sent to the device through a web browser interface. It should be possible to write code, compile it, and upload it to a device utilising only a web browser. A networked database is used to archive all data arriving from the device and this too is accessible via a web browser or as an XML web service. Users are therefore able to control and monitor their devices programmatically using HTTP requests from within their own programs.

- *Low Cost.* The devices should be as low cost as possible ideally costing no more than a few dollars to produce in volume so as they can be deployed without adding significantly to the cost of appliances in which they are deployed. It is envisaged that these devices will be deployed in great numbers and the data acquisition and control architecture is aimed at providing as much of the functionality as possible with network based services. Device costs will become more important in future when cheaper communications technologies, for example blue tooth are employed.

II. Existing Technologies

The field of SCADA technologies is evolving rapidly especially with the use of mobile telephones as the communications medium. Flexible and sophisticated SCADA systems suitable for providing industrial solutions can exceed tens of thousands of dollars in cost. In some industries this is a reasonable capital outlay, but it is out of the reach of many other potential users. At the other end of the scale some simple purpose built systems are cheap but inflexible. For example, a back to base home monitoring system is an example of a cheap SCADA technology but it can not be reconfigured for other applications.

Many existing technologies also use custom communication methods. For example, an industrial controller for a water management system might have a dedicated data line between the device and the control office. Specially built communication infrastructure, such as complete cabling, can result in high costs.

Naturally, a flexible system that is both cheap, and relies on existing infrastructure would be preferable. This project works towards that goal.

III. Implementation Details

A. *The SCADA Engine*

1) *Hardware*

The parts used to create the core of the device consist of an Internet Engine (explained below) and a GPRS capable mobile telephone or modem that provides TCP/IP communications routed via the internet. The Internet Engine available from EDTP Electronics [4] is based on Application Note 731[5] by Microchip. It is a rather simple device consisting of the following:

- A PIC16F877 microprocessor.
- Seiko S7600A – a TCP/IP stack implemented in hardware.
- Microchip 24LC256 – 32Kb EEPROM.
- Two RS232 serial connections – one can be used as a direct RS232 connection with the microprocessor, whilst the other is provided for connection to the GSM modem.
- A temperature sensor and battery backed calender/clock.

Without varying the chips used the design of the circuit board could be improved to provide a greater number of I/O channels for the connection of peripherals. The device has the overall design depicted in Fig. 1. Using I/O pins from the PIC, various peripherals are attached such that the device can be used as a remote data-logger, sensor or monitor.

The Internet Engine is a small device, approximately 11×8×2cm and weighing less than 100g. A GPRS capable mobile phone such as the Motorola Timeport is not much bigger, adding another 200g or so to the system and an integrated GSM module would be smaller again. The two components communicate via a serial cable. The Internet Engine initialises the connection utilising an extended AT command set. As the telephone connects to the Internet, a PPP session is established between the telephone and the Internet Engine. At this point, the Internet Engine becomes connected to the Internet via the telephone and is able to directly establish connections with other nodes on the Internet.

Technically, any GPRS enabled GSM modem should work. However, we have found that this is not necessarily the case. We have tried both the Motorola Timeport 7389i as well as the Ericsson T39m for GPRS connectivity. Unfortunately, the handsets behave differently in terms of how they establish a PPP session with the initiating device. We have not yet been able to successfully establish a PPP connection between the Internet Engine and the T39m without the handset appearing to disconnect immediately. Using the Timeport however, a successful connection is made and maintained.

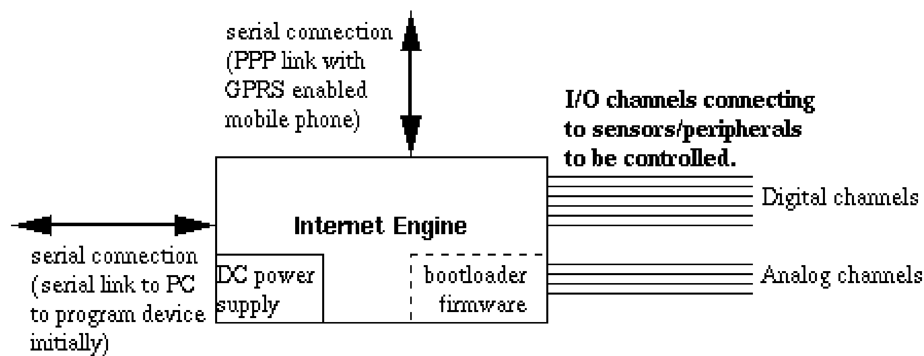

Fig. 1 - A black box view of the device.

2) Software

The software running on the PIC is compiled C code without an operating system. We provide boot loader code to manage communications and enable over the air programming. The program memory on the chip is a mere 8KB, and we are attempting to keep the bootloader code to below 5KB so that there remains a significant amount of memory for the user program. Some functionality in the bootloader is made available to the user program as 'function calls', similar to how an operating system makes system calls available to user programs.

Some functions that the bootloader uses are made available in this manner (such as a function to send a message to the server). Ideally, other 'common' C function such as string to integer conversion should be made available too. This sharing is required due to the minimal memory capabilities of the PIC.

There are a few C compilers available for PIC microprocessors (there are also some BASIC compilers, and a few assemblers, however we chose C since it eased the burden of assembly programming, but had much more power than BASIC programming). The one we chose to use is created by Custom Computer Services [2]. It provides an implementation of C that is similar to ANSI C, and has built in libraries that make complicated tasks such as outputting to a terminal, or communicating with devices over the 1-wire bus, relatively straightforward.

B. Server

The server consists of two parts – a web server to serve the front end to users who want to control their devices (called the front end), and controller software that is in constant contact with each of the devices (called the back end).

1) Back End

As discussed below, there is always a TCP connection open between each device and the back end. Since the front end, being a collection of web pages is essentially stateless, it was necessary to create an intermediate module to maintain the connections with the devices. This is the back end's job.

Due to the limited memory and limited processing power of the PIC16F877, and the relative ease and cheapness of programming on a PC, the decision was made to put as little logic as possible onto the Internet Engine, and put most of the logic into the back end. In order to achieve this, the back end was written from scratch. Java was chosen as the programming language for two primary reasons:

- TCP socket programming is very easy to do in Java, unlike C where a lot of low-level manipulation is required.
- The system would need to scale easily depending on the number of Internet Engines 'out there'. Since each device needs to maintain a constant connection with the server, a new thread would be required to handle each device. Thread programming is very simple and well implemented in Java.

The back end maintains two TCP connections with each device. One of the connections is used for uploading new programs and sending keep-alive signals, and the other is used for message passing. The device initiates both of these connections. The back end simply has server sockets waiting for devices to connect to. The reason for this is that a mobile phone connected to the Internet via GPRS is assigned a private IP address[2] that is not 'visible' to other nodes on the Internet. When it initiates a connection to a public node, traffic passes through a network address translator (NAT), which in our case is on Telstra's network. The remote node sees the NAT, not the originating node as the source of the connection. The gateway acts as a conduit between the public Internet, and all of the Telstra GPRS telephones which have private IP addresses.

Depending on the sophistication of the NAT, the user may be able to access the Internet in almost the same fashion as if they were accessing it from a public IP address – in other words, the operation of the NAT could be abstracted such that as far as the user is concerned, they have a public IP address. The Telstra NAT however, places many restrictions on the connection; one of them is the inability to establish a connection from an outside node on the Internet to the GPRS phone. Thus the device is required to make the connection outwards, rather than waiting for users to contact it. Once the connection is established, data can be sent in both directions (since TCP channels are full duplex). One advantage of the NAT is that devices, being invisible from the Internet are not

[2] Conforming to RFC1918.

subject to direct rogue attacks from third parties. This simplifies security as all communication is via a far more powerful networked computer that can use standard methods for implementing Internet security.

The data channel between the back end and a given device is used to send user messages back and forth. A message to the device is generated by a user on the front end, and is a plain text message forwarded to the user program on the device. Any messages that the user program running on the device sends back to the server are inserted into a database which is then available to the user via the front end.

In addition to communicating with each of the devices, the back end communicates with the front end via a very simple text based, line oriented protocol running over TCP. This is implemented on the back end as a server socket that the front end connects to whenever it needs to communicate with a device. It is important to note that the front end never communicates directly with a device; it always talks to the back end. The reason being, as discussed, that due to the NAT the front end cannot connect to a device itself, it must utilise the open connection that the back end has. The protocol allows the front end to ask questions such as "what devices are alive?" and "has the program been uploaded yet?" as well as perform actions such as send a message to a device, or upload a new program.

2) Front End

The front end to the system consists of a web interface for the user written in PHP. PHP was chosen as it is a relatively basic scripting language, but has many powerful functions for dealing with sockets – this was required for the communication with the back end.

The web interface allows a user to do the following:

- Find out information about writing source code. Since the bootloader will be running on the PIC, the user will not be able to utilise the full capabilities of the PIC, but will only be able to use a limited subset – the most obvious being the fact that the size of the program must not be greater than 8KB minus the size of the bootloader. The API for sending and receiving messages is also made available to the user.
- Provide a mechanism to upload source code and have it compiled. The result of the compilation is shown to the user via their browser. The system will parse the C code prior to compilation to do some basic checks – such as ensuring the user program does not use any of the interrupts used by the bootloader.
- Provide a way to upload any compiled program to the device of the user's choosing.
- Retrieve any messages that a given device has sent back to the server. Currently this returns an XML document consisting of any data in the database related to the given device. The data is put in the database by the back end as it receives messages from each device.
- Allow the user to send data to their program running on a device through the messaging system.

3) Fitting It Together

The back end acts as the system server and device proxy. The front end connects to it when required, and the devices are connected to it whenever they are switched on.

The communication protocol between the back end and the Internet Engines was made as simple as possible such that a minimal amount of logic would be required in the devices in order to implement it. As already mentioned, two dedicated channels are maintained with each device - one for uploading new programs and periodic pinging to test connectivity, and the other for message passing between the user program and the 'outside world'. The protocols for both of these channels have been defined in a way that puts as much of the burden as possible on the back end, and as little as possible on the devices themselves. The protocols are based on the sending and receiving of ASCII strings with a chosen Line Feed termination character to signal the end of a message.

Ultimately the back end will hook into a complete messaging system that will be used for control of and communication with the devices. Since no such system currently exists, a minimal system has been implemented whereby the devices can be controlled and communicated with via the front end, and any messages arriving from the devices are stored in a table in a local database.

Since the front end is implemented as a web service, users can make the HTTP requests to the front end from inside other applications, rather than doing it interactively, if they so wish. For example, if the user wishes to have a program compiled, they can make an HTTP POST to the front end, and pass the source code as one of the parameters. The front end returns any errors that occurred in the compiling stage. Similarly, another HTTP request can be made to tell the front end to upload a program to a given device. This would allow users controlling many devices to create their own 'front end' interfaces.

The front end communicates with the back end via a very basic, text oriented, messaging protocol. Some of the communication sequences are line based, whilst others are XML based. After spending some time working on this, we decided that it was more important to concentrate on the core of the project, and the messaging system could simply be built in later. There is however a basic implementation which currently works. Essentially, communication consists of single line commands that all come from the front end. Responses from the back end can either be line based responses, or XML documents.

Fig. 2 depicts how the complete system works together.

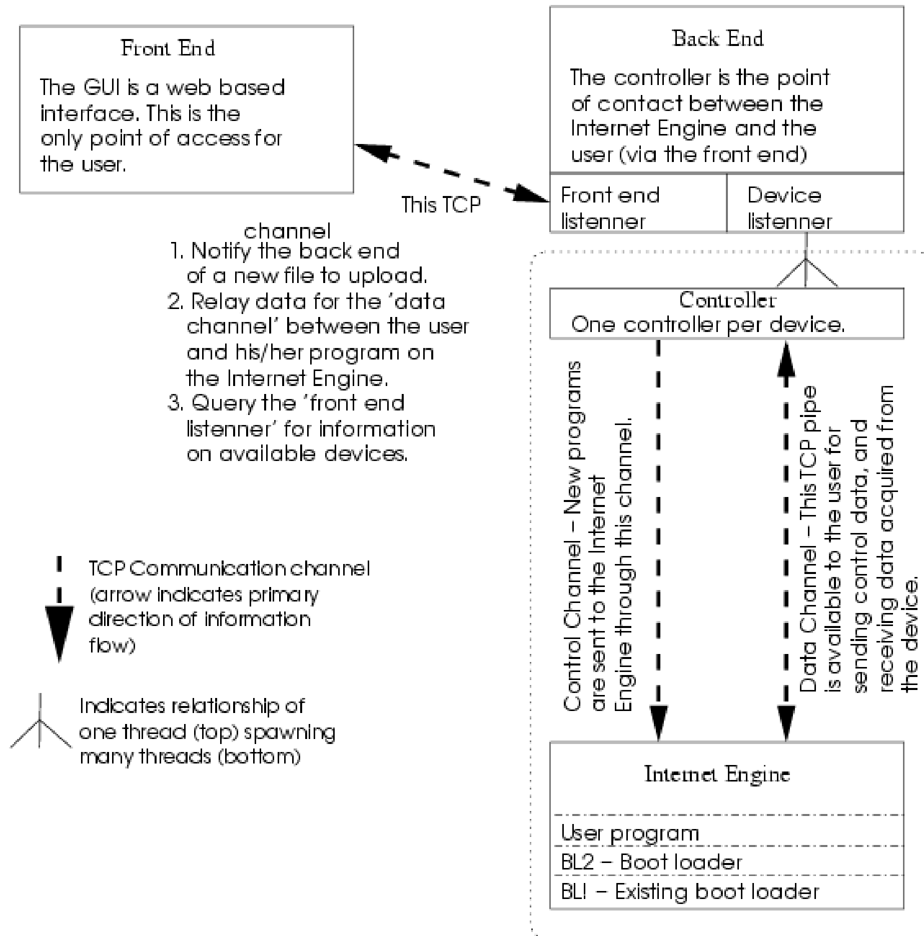

Fig. 2 - Module overview

IV. Discussion

A. Issues Encountered

Initially the CCSC compiler was chosen because of its price (about a quarter the price of the Hi-Tech C compiler). In the course of 5 months of programming with it, at least two bugs were found. In addition, it is known to not optimise code as well as the Hi-Tech compiler.

Many other difficulties were encountered with the development of the bootloader code for the Internet Engine. We expected to have trouble with the S7600A as it was the first hardware TCP/IP stack produced by Seiko. Fortunately however, we experienced no problems that were in any way attributable to the chip.

The most difficult issue to overcome was debugging a system that had a convoluted process for coding and testing. Each time an attempt was made to fix a bug, the code had to be uploaded to the device. Uploading the code using the serial link to a PC (via the bootloader that comes installed on the PIC when the Internet Engine is purchased from EDTP) takes up to 3 minutes for a 6KB program. The other option (which was used more frequently as the project progressed) was to program the PIC using a standard PIC programmer with some custom wiring to connect it up to the Internet Engine. Even so, programming the PIC from a Windows 2000 computer presented a timing issue which meant that programming had to be done slowly in order to overcome any operating system time-slicing that occurred on the PC. This ensured that any disruptions due to context switching would be insignificant with respect to the pin voltage transmission times associated with programming.

We did not have the facilities to use the in circuit debugger, so debugging consisted of inserting `printf()` statements.

B. Improvements To Be Made

While the PIC16F877 is a cheap flexible micro controller with excellent I/O and software driven reprogramming, the bootloader and messaging code utilises more of the available resources than is desirable and alternative devices, particularly with larger code space could be considered.

The issue of interrupts also needs to be addressed. Due to the fact that the interrupt vector is located at a specific address, there is no simple way for the user to make use of interrupts in their program, since the bootloader uses interrupts and occupies the interrupt vector. In order to allow for user interrupts without interfering with the bootloader's interrupts, the bootloader needs to implement an interrupt

service routine which deals with its own interrupts as it does currently. If the interrupt is not one of the bootloader's interrupts, it should call a user implemented function, a virtual interrupt service routine, that gives the user the opportunity to deal with it.

Rather than having the user write this themselves, this function could be written on the fly when the user submits source code to be compiled. Basically, a script could analyse the source code and change any interrupt routines implemented by the user into normal routines, then create an 'interrupt service routine' which will call the relevant interrupt routine. This 'interrupt service routine' will be compiled into a known memory location, such that when uploaded and running, the bootloader will be able to call it when an interrupt occurs.

***Index Terms*--**

GPRS General Packet Radio Service. This is a packet switched method of connecting to the Internet with mobile phones. It relies on existing GSM mobile infrastructure.

GSM Global System for Mobile Communication.

HTTP HyperText Transfer Protocol. The protocol used to transfer documents over the world wide web.

PPP Point-to-Point Protocol. A very simple protocol, which can provide a transport layer for Internet Protocol data between two endpoints.

SCADA Supervisory Control And Data Acquisition. Supervisory control is the control of a device at a distance by specifying the end results and having the device, through a closed loop, achieve those results.

SMS Short Message Service. A method of sending text messages of up to 140 bytes (160 characters) between mobile phones.

V. References

[1] Taylor, K; "Mobile Monitoring and Control Infrastructure", CSIRO Available online at http://mobile.act.cmis.csiro.au

[2] C Compiler by Custom Computer Services. Available from http://www.ccsinfo.com

[3] Fred Eady (February 2001). *An S-7600A/PIC16F877 Journey*, Circuit Cellar Online http://www.chipcenter.com/circuitcellar/january01/c0101fe1.htm

[4] Hi-Tech C Compiler. Available from http://www.htsoft.com/

[5] The Internet Engine, available from EDTP Electronics, http://www.edtp.com/

[6] Rodger Richey and Steve Humberd (2000). *Embedding PICmicro® Microcontrollers in the Internet*, Microchip Application Note 731, http://www.microchip.com