A Bluetooth protocol stack for embedded use

Magnus Unemyr - June 13, 2002

A Bluetooth protocol stack for embedded use
Magnus Unemyr of IAR Systems, describes a tool designed to make it quicker and easier to configure and optimize a Bluetooth protocol stack for a project.

Bluetooth wireless technology is one of the most interesting new means of communication for embedded systems. Thousands of products from many different manufacturers will most probably contain Bluetooth functionality for short distance data and/or sound transmission.

Bluetooth products communicate over the radio and therefore no optical connection is needed (which is not the case for IrDA). This means communication can be made with devices in another room or in a closed briefcase - see figure 1.

![Diagram of two products communicating over a wireless Bluetooth radio link](image)

*Fig 1: Two products communicating over a wireless Bluetooth radio link*

The communication distance is usually up to 10ms, but there is also a version of the radio interface with signal amplification that can communicate up to 100ms. Maximum communication speed is around 700KB/s.

All products using Bluetooth wireless technology must follow the communication protocols
documented in the Bluetooth specification. The advantage of this is interoperability — all products from different manufacturers function well together.

The Bluetooth specification is a highly technical document (over 1500 pages) describing all the functionality that must be implemented in order to build a product that complies with the Bluetooth specification.

A Bluetooth protocol stack is a software library that implements the communication protocol as defined in the Bluetooth specification. The deep technical expertise and huge need for development resources to develop a Bluetooth protocol stack makes it practically impossible for most companies to develop their own stack.

The stack must also pass the official Bluetooth test suites that have been defined by the Bluetooth SIG (Special Interest Group). In order to include Bluetooth technology in their products, the vast majority of all companies will therefore purchase a commercial Bluetooth protocol stack from a third party supplier. Developing a Bluetooth protocol stack internally cannot be financially motivated.

This article gives an overview of the functionality in a Bluetooth protocol stack, as well as some of the design decisions taken by IAR Systems' development team when developing a compact stack for embedded systems.

A Bluetooth protocol stack is a software component that implements the communication protocols defined in the Bluetooth specification. The stack communicates with the Bluetooth hardware (usually via UART or USB), and presents an easy-to-use high-level API by which the application program can use the different services in the stack.

A Bluetooth protocol stack can be divided into two parts; the core and the profiles.

The core consists of:

- Host Controller Interface which provides a standardized communications protocol between the stack and the Bluetooth module. HCI communication packets can be transmitted over UART, RS232, USB, or PC Card interface.
- Logical Link Control and Adaptation Protocol allows multiple communication channels to share a single Bluetooth link (multiplexing). L2CAP also handles segmentation and assembly of long messages.
- RFCOMM implements the functionality needed for a virtual RS232 link, including modem control signals. Most of the application profiles use RFCOMM to send and receive data.
- Service Discovery Protocol provides functionality to publish supported Bluetooth functionality (SDP server), as well as to query other Bluetooth products for Bluetooth functionality (SDP client).

The Bluetooth specification also defines how Bluetooth should be used within specific application fields (each application usage is defined in its own profile). A number of profiles have already been defined, and more are expected to be developed in the future. Some of the most important profiles
Generic Access Profile is mandatory and is used to manage connections between different Bluetooth products. GAP exposes a high-level API that can be used by the application program to configure the stack and manage connections to different Bluetooth products.

Service Discovery Application Profile is mandatory and is used to query other Bluetooth products for supported services and functionality. SDAP exposes a high-level API that can be used by the application program to query other Bluetooth products for supported functionality.

Serial Port Profile provides emulated virtual serial ports over a Bluetooth radio link. SPP exposes a high-level API that can be used by the application program to transmit and receive data in a way that is very similar to a standard serial port.

Headset Profile provides functionality for linking sound between an audio gateway (usually a GSM phone) and a headset. HSP exposes a high-level API that can be used by the application program to establish and manage the sound link.

There are also many other profiles for different purposes, e.g. file transfer, network access, and dial-up network connections. New profiles are being specified for e.g. cars, audio/video, digital cameras and printer management.

When a Bluetooth protocol stack has been purchased an application program must be developed to use the stack functionality by calling the different API functions. The stack takes care of all the complex communication rules that are defined in the specification and manages the communication with the Bluetooth module.

Table 1 contains a few examples of API functionality provided by a stack (in this case the IAR Systems stack) to an application program for a few different profiles.

<table>
<thead>
<tr>
<th>Profile</th>
<th>API function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial Port</td>
<td>SPP_AddPortRq()</td>
<td>Creates a virtual serial port</td>
</tr>
<tr>
<td></td>
<td>SPP_RemovePortRq()</td>
<td>Deletes a virtual serial port</td>
</tr>
<tr>
<td></td>
<td>SPP_OpenRq()</td>
<td>Opens a virtual serial port</td>
</tr>
<tr>
<td></td>
<td>SPP_CloseRq()</td>
<td>Closes a virtual serial port</td>
</tr>
<tr>
<td></td>
<td>SPP_WriteRq()</td>
<td>Prints data to a virtual serial port</td>
</tr>
<tr>
<td></td>
<td>. . .</td>
<td></td>
</tr>
<tr>
<td>Headset Profile</td>
<td>HSP_OpenVoiceRq()</td>
<td>Opens a sound channel</td>
</tr>
<tr>
<td></td>
<td>HSP_CloseVoiceRq()</td>
<td>Closes a sound channel</td>
</tr>
<tr>
<td></td>
<td>HSP_VolumeRq()</td>
<td>Adjusts the volume</td>
</tr>
<tr>
<td></td>
<td>. . .</td>
<td></td>
</tr>
</tbody>
</table>

There are several Bluetooth protocol stacks on the market, but most of them are not optimized for small embedded systems. Many stacks are developed for work stations or large embedded systems. At a later stage some of them are adapted to small embedded systems, which can often cause problems. Stacks requiring 50-100KB ROM, or even more, are not at all unusual. This is often far too much for a small embedded system.
These are some important issues worth considering when choosing a Bluetooth protocol stack:

- Size
- Portability
- Configurability

Size: The size of a Bluetooth protocol stack is not important for personal computers, work stations, and large embedded systems. For most embedded systems, however, a compact stack using a minimum of memory is very important, as it can reduce production costs when large volumes are produced.

When IAR Systems developed its embedded Bluetooth protocol stack a very important goal was to develop an implementation with a minimal memory footprint. A stack that implements a virtual serial port only requires around 16KB ROM when compiled for an ARM7.

IAR Systems is known as a supplier of C/EC++ compilers for embedded systems, something which of course was exploited when the stack was developed. The compiler R&D group within the company took part in the stack development to make sure the C code was written with a C grammar that makes it possible to compile code to the most compact machine code ever using most compilers on the market. The result was a very compact stack that is also extremely portable between different CPU architectures.

Portability: When you design a stack intended for embedded systems portability must be a part of the design from the very beginning. The most obvious questions to ask are those about adaptations to different hardware and software platforms, e.g. different UART circuits, Bluetooth modules or operating systems.
At the design of the Bluetooth stack IAR's development team isolated all platform-dependent functionality in abstraction layers. The developer can easily make his own adaptations to different operating systems, UART drivers, or support for different Bluetooth modules. In many cases this can even be done automatically using the configuration tool IAR MakeApp.

Less obvious portability issues that are much more important are those about architectural dependencies. These issues are less well known but can make a stack totally useless on certain CPU types. The stack being written in ISO/ANSI C does not help, as the data representation can get a different meaning when the stack is recompiled for another CPU architecture. In other cases the processor bus system cannot handle 16 or 32 bit data at unaligned addresses.

Porting the stack might well become a nightmare if the stack is implemented with different types of architectural dependencies. The following are common problems when porting software for embedded microprocessor systems:

- CPU architectures with big or little-endian byte order
- CPU architectures with 16 or 32 bit aligned address access only
- Compilers with insufficient structure packing

The Bluetooth protocol stack developed by IAR Systems was designed for portability between different processor families. It is implemented in a way that protects the stack from the problems mentioned above. As the stack is implemented in ISO/ANSI C it also works with different types of C compilers.

It is important that the Bluetooth stack chosen is future-proof and can be used even if the system configuration of the product is changed. Therefore it is important that the stack can be used in different use-cases. The IAR Systems stack can be used in the system configurations shown in figure 3.
Fig 3: Possible systems configurations

Configurability

In order to launch an optimized product it is important that the stack can be set up for the actual project. If the stack cannot be set up and optimized for the actual project the stack might contain several functions that are not used, which means a waste of expensive memory. The stack will also most probably become slower, as code not needed still has to be executed.

A stack that is optimal for a certain project should only implement the functionality really used (and the functionality that is mandatory in the Bluetooth specification). Therefore it is important that the developer can easily configure and optimize the stack for the project, without having to spend too much time and money on manual changes in the stack implementation.

As a solution to this problem IAR Systems' development team in Jönköping launched a very powerful software program that can be used to configure and optimize the stack — the IAR MakeApp for the Bluetooth Protocol Stack Using this tool, a development engineer can easily configure and optimize stack functionality in a few minutes, and then watch the tool automatically generate an optimized ISO/ANSI C stack implementation.

IAR MakeApp for the Bluetooth Protocol Stack gives the development engineer a graphical design environment providing intuitive settings for stack functionality, as well as a code generator that can generate thousands of different stack implementations in optimized ISO/ANSI C!
The major advantages of this tool are:

- Within a few minutes a developer can configure the stack core and profiles at a high abstraction level without knowing anything about the internal implementation of the stack.
- Stack functionality (such as client/server position, message size, profile functionality, etc.) can be configured by a few simple mouse clicks.
- An optimized stack implementation is automatically generated as ISO/ANSI C source code. The stack is thus adapted and optimized for the specific project, and all unused functionality is removed in order not to waste memory.
- The tool can also automatically generate platform adaptations (support for different operating systems, UART drivers, Bluetooth modules, etc)
- For certain processor families the tool can also generate complete driver libraries (bus and interrupt controllers, DMA, UART, A/D and D/A converters, timers, etc)

Without a professional configuration and optimization tool such as IAR MakeApp the stack must be manually configured, which can be both time-consuming and costly. This type of configuration tool can reduce a large amount of development time at the configuration stage, at the same time as the stack is thoroughly optimized and requires a minimum of memory.

In addition to the possibility to configure the stack functionality, the developer can also choose the API functionality. API functions that have not been used are deleted from the stack implementation, together with all code that might have been needed by the API functionality at a lower level in the stack.

Published in Embedded Systems (Europe) June 2002