Texas Instruments

Application Report
SPRA944 − December 2003

Bluetopia for RF3 Integration

Texas Instruments
Stonestreet One

HFK DP Software Applications

ABSTRACT

This document is intended to show the integration into and operation of the Bluetopia® Bluetooth™ Protocol Stack and Profiles within the bounds of the eXpressDSP software Reference Framework Level 3 (RF3). This integration incorporates a serial transport layer and Stonestreet One's (SS1) existing Bluetopia core stack and Hands-Free and Headset profiles IP with the framework defined by RF3. The sample application source code and documentation should be used in conjunction with this document for readily viewable examples of the steps described within. This document explains the major components in detail and gives some insight into the architecture of the system. In addition to the detailed explanation, a quick start guide is included to be used as a verification checklist as well as a way to get running quickly without having to sort through too much text at the onset of the integration effort.

Contents

1 Introduction

This Bluetopia Bluetooth protocol stack integration document can be used to guide the user to quickly integrate Bluetopia into standard and custom TI Reference Framework 3 projects.

1.1 Definitions, Acronyms, and Abbreviations

The following list of terms will be used in this application report and other Bluetopia documentation:

- API – application programmer's interface
- Bluetopia – Stonestreet One's Bluetooth Protocol Stack

- DSP – digital signal processor
- GAP – generic access profile
- GOEP – generic object exchange profile
- OTP – object transfer profile
- HCI – host controller interface
- IP – intellectual property
- L2CAP – logical link control and adaptation protocol
- RF3 – Texas Instrument's Flexible Framework: Reference Framework Level 3
- RFCOMM – RFCOMM layer of the Bluetooth Protocol Stack
- SCO – synchronous connection-oriented link
- SDP – service discovery protocol
- SPP – serial port profile
- SS1 – Stonestreet One
- Thread – DSP/BIOS unit of execution, i.e., task (TSK) or software interrupt (SWI)
- TI – Texas Instruments
- UART – universal asynchronous receiver transmitter

2 Quick Start

Following is a list of actions that need to be performed for integration. Use this list as a quick start guide or merely as a checklist to verify the integration work.

Changes to the project:

- Add the stack library file to the project (use the appropriate near or far call version).
- Add the desired profile library files to the project (use the appropriate near or far call version).
- Add the custom UART Driver.

Additions to the configuration database file (.cdb):

- Add SWI Object swiBluetopiaCallbackThread
  - Function: _thrBluetopiaCallbackThread
  - Priority: X
- Add SWI Object swiBluetopiaTimerThread
  - Function: _thrBluetopiaTimerThread
  - Priority: X
- Add SWI Object to act as a transport thread (e.g., _swiBluetopiaTransportThread)
  - Function: _thrBluetopiaTransportThread
  - Priority: >X

- Add a SWI Object to make stack API Calls (e.g., _swiControl)
  - Function: _thrControlRun
  - Priority: <=X
- Add CLK Object clkBluetopiaTimerInterrupt
  - Function: _thrBluetopiaTimerIsr

Software additions:

- Port HCITRANS.C to the custom UART driver to be used.
- Include SS1BTPS.H to the file that will be initializing the stack and making API calls.
- Include profile API header files to the file that will be opening profiles and making API calls.
- Add code to initialize Bluetopia, open desired servers/clients, register SDP records, set modes using GAP, and handle callback events for desired layers and profiles.

3 Software Model/Framework

The software model employed by this design is a layered, thread-oriented, multichanneled, multi-algorithm, static system. These characteristics are defined by eXpressDSP software Reference Framework level 3. All execution in RF3 is performed from the context of HWIs, SWIs, or CLK object processing threads. One common limitation of all of these types of threads is that they cannot block or pend. Since there are several critical instances of interthread synchronization and mutual exclusion within the Bluetooth protocol stack, a method was put in place to accommodate these needs. While invisible to the user, several rules do need to be followed as to not disturb this environment. The rules are described in detail in the following sections.

4 Core Stack

The core stack library is composed of the HCI, L2CAP, RFCOMM, SPP, GOEP, OTP, SCO, GAP, and SDP layers. The stack can by used by adding the library file to the project, including **SS1BTPS.H** to a source file, and successfully initializing the stack. Individual stack layers can be switched on and off via parameters passed to the initialization function. See the *Bluetooth Protocol Stack API Reference Guide* (SPRU648) for complete details.

The core stack uses two integral threads, or software interrupts (SWIs), for operation called swiBluetopiaCallbackThread and swiBluetopiaTimerThread. These two SWIs must be at the same priority. In addition, there are two restrictions for the priority of swiBluetopiaCallbackThread and swiBluetopiaTimerThread. The SWI that is used to transfer data into the stack (swiBluetopiaTransportThread) must be at a higher priority than the stack and timer threads. Second, all function calls into the stack must be made from a software interrupt, and this interrupt must be at a priority at or lower than swiBluetopiaCallbackThread and swiBluetopiaTimerThread. All SWIs that make calls into the stack must be at the same priority. These rules are due to a priority inheritance scheme used for mutual exclusion within the stack. Failure to follow them could result in the corruption of internal stack information or the hanging of API calls into the stack.

The stack also requires one clock object clkBluetopiaTimerInterrupt. This clock object is responsible for posting swiBluetopiaTimerThread at the appropriate times to invoke timeout callbacks internally to the stack.

5 Profiles

Individual Bluetooth profiles can be included by either adding the library or, if provided, appropriate source files to the project. Including the associated header file will allow API calls to be made. Profiles can be added individually as needed as long as the required supporting layers of the core stack for that profile are properly enabled and initialized. The profiles' architecture is similar to that of the stack, and requires the same calling rules to be followed. Normally, any restriction or condition that must be met to make a core stack API call applies to profile API calls. See the sample application in *Hands-Free Kit and Headset for the RF3 Reference Platform Sample Application* (SPRA843), for an example of including and using a Bluetooth profile.

6 Application

It is suggested that the portion of the application that makes stack and profile API calls resides either within a common thread, or in the stack's own callback functions. This helps avoid any mutual exclusion issues that may arise. To aid in mutual exclusion and the protection of local application variables, the API offers a method of "locking" the Bluetooth stack. While the stack is locked, no callback events will be invoked. It is good practice to make these locked sections as short as possible to avoid excessive event latency. At no time can the application be dependent on receiving a stack callback during a locked portion. Since no callbacks are made while the stack is locked, the application will hang waiting for it. See the *Bluetopia Protocol Stack API Reference Guide* (SPRU648) and header files for more details.

It is important to make sure that any API calls are made only from SWIs, and that the SWI is of the same or lower priority than swiBluetopiaCallbackThread. If API calls are made from multiple SWIs, the SWIs must all be at the same priority. As in the sample application, a common place to make API calls is from the control thread. The maintenance of the Bluetooth stack is usually based on asynchronous events, which the control thread was designed to handle. This helps keep these actions separate from the synchronous, block-based processing that is regularly happening in the proc threads.

The control thread is periodically posted by a clock object. This scheme lends itself to driving Bluetooth functionality by a state machine. For example, the initial state could be used to initialize the stack, open any servers that may be required, and place the Bluetooth module into the desired mode. The next state could be used to invoke any other Bluetooth actions that might be desired, or it could move directly to monitoring a user interface. The state of the application would then be driven by Bluetooth events received via callback, timed events, or asynchronous inputs from other sources such as buttons, switches, or interrupts. See the sample application for an example of initializing the stack, making API calls to various layers, and locking and unlocking the stack. The sample application is also driven by a state machine, which is controlled by asynchronous user inputs as well as by Bluetooth stack events received via callback.

7 Host Controller Interface Transport Layer

When calls are made into the stack, after command processing and packet building, the formatted data must be passed on to the Bluetooth module. The protocol stack communicates command and control data to the Bluetooth module via UART. Since this driver is not used by the core RF3 application, but rather is only exposed to the protocol stack, the standard low-level input/output (LIO) driver model is not used. The LIO driver is better suited for drivers that deal with synchronous, frame-based data. As described by its name however, UART data is asynchronous data, with Bluetooth specific frame sizes ranging anywhere from a few bytes up to several hundred bytes. Data of this type is better handled by a generic circular buffer than frame based buffers.

The first step of integrating a custom UART driver with the stack is to modify the HCI Transport file, HCITRANS.C. This file contains a function stub for Open, Close, and Write functions.

The Open function allows calls to be made to configure and open the custom UART driver. A callback function pointer is passed to this function as a parameter. This callback function should be saved in a data structure global to the scope of HCITRANS.C, and is used to apply data received from the UART driver to the protocol stack. Passing received data from the UART driver to the stack can be handled in several ways. Two common examples are interrupt/callback driven and polling.

In the interrupt or callback driven method, the custom UART driver provides a method by which it can notify its user that data has been received. Upon receipt of this indication, the user can post a SWI designated for the purpose of reading data from the UART driver and calling the callback function passed in via HCITRANS Open function. In the sample application, the SWI responsible for doing this is swiBluetopiaTransportThread. If, by chance, the indication given by the UART driver is being called from within the context of a SWI, the callback function into the stack can be called directly.

In the polling method, a periodic event causes a SWI to run that can in turn call the read function of the UART driver, and if data is available, pass the data on to the stack via the callback function. Regardless of what thread is responsible for performing this task, there are two important rules that must be followed. First, the thread must be a SWI. Second, the SWI must be of higher priority than swiBluetopiaCallbackThread.

The Write function is responsible for passing data from the stack to the UART driver for transmission to the Bluetooth module. The Write function is fairly straightforward, having only a buffer pointer and the buffer’s length as parameters. The most important note about the Write function is that the layer of the stack that calls Write is expecting all of the data to be either sent or buffered into the UART driver upon returning from the function. This function must block until all data is processed. If for some reason all of the data cannot be sent, an error must be returned to the caller.

8 References

1. *Hands-Free Reference Platform Requirements v1.4*, Stonestreet One, Inc., 2003.
2. *TMS320C5000 DSP/BIOS Application Programming Interface (API) Reference Guide* (SPRU404)
3. *TMS320C54x Chip Support Library API Reference Guide* (SPRU420)
4. *Writing DSP/BIOS Device Drivers for Block I/O* (SPRA802)
5. *Reference Frameworks 3: A Flexible, Multichannel/Algorithm, Static System* (SPRA793)
6. *Specification of the Bluetooth System, Version 1.1*, Bluetooth Special Interest Group (SIG)., 2001.
7. *Specification of the Bluetooth System Profiles, Version 1.1*, Bluetooth Special Interest Group (SIG)., 2001.
8. *Hands-Free Profile, Version 1.0*, Bluetooth Special Interest Group (SIG)., 2003.
9. *Headset Profile, Version 1.0*, Bluetooth Special Interest Group (SIG)., 2001.
10. *TMS320VC5407/TMS320VC5404 Fixed-Point Digital Signal Processors Data Manual* (SPRS007)
11. *TMS320C54V90/TMS320C54CST/TMS320VC5406 Evaluation Module Technical Reference*, Spectrum Digital, 2001.
12. *TMS320C54x Bluetopia Hands-Free Profile API Reference Guide* (SPRU704)
13. *Bluetopia Headset Profile API Reference Guide* (SPRU649)
14. *Bluetooth Protocol Stack API Reference Guide* (SPRU648)

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265