ONE-NET Porting and Implementation Guide

by Roger Meadows

(Revision 0.1)

Overview

This document is intended to serve as a guide when porting ONE-NET to a new processor or when interfacing to a new transceiver. It is also intended to help serve as a guide when developing new applications for hardware environments for which ONE-NET has already been ported. The ONE-NET source distribution contains ONE-NET code that is independent of processor or transceiver and it includes source code that is specific to various processors and transceivers. The ONE-NET source distribution also contains application code for several ONE-NET devices.

Categories of ONE-NET Code

In order to understand the porting process, we need to understand the various categories of ONE-NET code. This is a logical breakdown of the ONE-NET code into categories that make sense with respect to taking existing ONE-NET code and porting it to a new processor and/or transceiver. To help understand where to find the source code that is assigned to the categories described below, we will assume that the ONE-NET source code has been checked out into a directory name ***ONE_NET***. Below is a list of the directories you would expect to see in the ONE_NET working copy. There maybe other files in this directory, but we will not be discussing them in this document.

- ***applications/***
- ***common/***
- ***one_net/***
- ***processors/***
- ***transceivers/***

Hardware Independent Category (*ONE_NET/one_net*)

The original implementation of code in this category was on the Renesas R8C family of microprocessors. However, a design goal of the code in this category is that it should be easy to compile and link it on any processor. The source code in this category is stored in the ***ONE_NET/one_net*** directory. Below is a list of the files and directories you would expect to see in the ***ONE_NET/one_net*** directory of a ONE_NET working copy.

- ***app/***
- ChangeLog.txt
- ***cli/***
- ***mac/***
- ONE-NET License.pdf
- ***port_specific/***

- ***utility/***

Processor Specific Category (*ONE_NET/processors*)

The source code in this category consists of functions that will be called from the hardware independent ONE-NET code and is processor specific. The interface functions defined in this category were designed to hide the details of the processor hardware from the hardrware independent ONE-NET code.

Below are some examples of the processor specific functions that need to be implemented when porting ONE_NET to a new processor type. The function prototypes for these processor specific functions are in the header file **one_net_port_specific.h** within the ***ONE_NET/one_net/port_specific*** directory. The ***one_net_port_specific.h*** file includes doxygen style function prototype documentation.

Although the header file **one_net_port_specific.h** contains the declaration of the function prototypes for various processor specific functions, the source file containing the implementation of these functions for a particular processor will be in a processor specfic directory. For example, if "<processor_type>" was the abbreviation for a particular processor type, the implementation of processor specific functions would be in the directory ***ONE_NET/processor/<processor_type>/src/common***.

General Purpose Processor Specific Operations (defined in ***ONE_NET/processors/<processor_type>/src/common/one_net_port_specific.c***)

- `void * one_net_memmove(void * dst, const void * SRC, size_t len);`
- `SInt8 one_net_memcmp(const void *vpl, const void *vp2, size_t n);`
- `void one_net_int16_to_byte_stream(UInt16 VAL, UInt8 * byte_stream);`
- `UInt32 one_net_byte_stream_to_int32(const UInt8 * BYTE_STREAM);`
- `void one_net_int32_to_byte_stream(UInt32 VAL, UInt8 * byte_stream);`
- `tick_t one_net_ms_to_tick(UInt32 MS);`
- `UInt32 one_net_tick_to_ms(tick_t TICK);`
- `tick_t one_net_tick(void);`

Non-volatile Memory Processor Specific Operations (defined in ***ONE_NET/processors/<processor_type>/src/common/one_net_client_port_specific.c***)

- `void clr_flash(void);`
- `void erase_param(void);`
- `const UInt8 * read_param(UInt16 * len);`

Transceiver Specific Category (*ONE_NET/transceivers*)

The source code in this category consists of functions that will be called from the hardware independent ONE-NET code and is transceiver specific. The interface functions defined in this category were designed to hide the details of the transceiver hardware from the hardware independent ONE-NET code.

Transceiver Specific Operations (defined in ***ONE_NET/transcievers/***

<transceiver_type>/<transciver_type>.c)

- `void one_net_set_channel(UInt8 CHANNEL);`
- `BOOL one_net_channel_is_clear(void);`
- `void one_net_set_data_rate(UInt8 DATA_RATE);`
- `one_net_status_t one_net_look_for_pkt(tick_t DURATION);`
- `UInt16 one_net_read(UInt8 * data, UInt16 LEN);`
- `UInt16 one_net_write(const UInt8 * DATA, UInt16 LEN);`
- `BOOL one_net_write_done(void);`

Application Specific Category (*ONE_NET/applications*)

The source code in this category implements the application functionality of the ONE-NET master or client. This source code makes calls to public interface functions in the hardware independent ONE-NET code category.

Summary of Ported Code Included in the ONE-NET Source Distribution

As mentioned earlier, in addition to the processor and transceiver independent code, the ONE-NET source code distribution contains ONE-NET source code that has been ported for specific processors and transceivers. This section provides an overview of the source code that is processor and transceiver specific.

Ported Processors (*ONE_NET/processors*)

The ***ONE_NET/processors*** directory contains a ***renesas*** directory to hold the Renesas specific code used on the ONE-NET Evaluation boards and the ONE-NET simple input/output boards. It will also contain (if it does not already) a ***linux*** directory to hold the Linux specific code used in the ONE-NET Linux Test Environment.

- ***ONE_NET/processors/***
  - ***linux/***
  - ***renesas/***

Within the ***ONE_NET/processors/renesas*** directory, the files associated with the Rensas Integrated Development Environment (HEW) are seperated from the C source files. The HEW files are the ***hew*** directory and the processor specific source files for the renesas processors are in the ***src*** directory.

- ***ONE_NET/processors/renesas/***
  - ***hew/***
  - ***src/***

The files in the ***ONE_NET/processors/renesas/hew*** directory are further broken down into categories of ONE-NET devices.

- ***ONE_NET/processors/renesas/hew/***
    - ***app_board/***
    - ***one_net_app_ex/***
    - ***one_net_eval/***
    - ***simple_eval/***

As this document is being written the ***linux*** port is a work in progress.

Ported Transceivers (*ONE_NET/transceivers/*)

The ***ONE_NET/transceivers*** directory contains a directory for each of the transceivers for which ONE-NET has been ported.

- ***ONE_NET/transceivers/***
    - ***adi/*** (Analog Devices ADF7025)
    - ***ia/*** (Integration Associates IA4421)
    - ***micrel/*** (Micrel MICRF505)
    - ***rfm/*** (RF Monolithics TRC102)
    - ***semtech/*** (Semtech XE1205)
    - ***ti/*** (Texas Instruments Chipcon-TI CC1100)

Transceiver Abstraction Layer (TAL)

The Transceiver Abstraction Layer (TAL) is a method of separating the transceiver specific operations from the hardware-independent operations. The transceiver operations that can be performed by the hardware-independent ONE-NET code are identified in the **tal.h** file for each transceiver. Below is a list of the TAL operations that must be implemented for each transceiver. The implementation of the TAL operations is included in a C source file within ***ONE_NET/transceivers/<transceiver_name>/*** directory and is named **<transceiver_name>.c**. For example, the Analog Devices ADF7025 TAL functions are in **ONE_NET/transceivers/adi/adi.c**. In addition to the TAL operations, the Transceiver Specific Operations listed above (declared in ***ONE_NET/one_net/ port_specific/one_net_port_specific.h)*** are usually defined in this source file.

TAL Operations

- TAL_INIT_TRANSCEIVER()
- ENABLE_TRASNCEIVER()
- DISABLE_TRANSCEIVER()
- INIT_RF_INTERRUPTS()
- TAL_TURN_ON_RECEIVER()
- TAL_TURN_ON_TRANSMITTER()

Each transceiver should have a ***tal.h*** header file. In the ***tal.h*** file the TAL operations symbols are mapped either directly to commands to perform the operation or they are mapped to function names that will executed to perform the operations. If a TAL operation is mapped to a function name, that function should be defined in the file ***ONE_NET/ transceivers/<transceiver_name>/<transceiver_type>.c***.

Source Code Portability Techniques Used

Renesas HEW for R8C Platform

Portable Data Types

Portable data types used in ONE-NET are defined in ***ONE_NET/processors/src/common/one_net_types.h***.

R8C Platform Flag

The symbol "_R8C_TINY" should be defined when building ONE-NET on the Rensas R8C HEW environment. When the "_R8C_TINY" symbol is not defined, the Linux GCC development envoronment is assumed.

Use of "inline" function declaration

The Renesas M16C Standard Tool Chain C compiler (as documented in "M16C/60,30,20,10,Tiny,R8C/Tiny Series C Compiler Package V.5.43 C Compiler User's Manual") only supports the "inline" function declaration, not the "static inline" supported by GNU C. As a result, ONE-NET code uses the symbol "ONE_NET_INLINE" for declaring a function to be "inline". In the Renesas HEW development environment, "ONE_NET_INLINE" is mapped to "inline". In the Linux GCC development environment, "ONE_NET_INLINE" is mapped to "static inline".

Design Considerations for Porting

Application Code versus ONE-NET Code

The requirement for allowing ONE-NET code to execute versus allowing application code to execute varies depending on whether a device is a master or a client. If the device is a client, the requirements will vary based on the nature of the client. The requirements will also vary based on the type of communications used between the processor and the transceiver.

One of first design decisions when creating a ONE-NET device has to do with how often and for how long an application needs to be listening for ONE-NET packets. A master needs to be listening for messages whenever it is not sending a message. On the other hand, a client may want to sleep for a period of time with the transceiver powered down and the processor in a low power mode in order to conserve battery power. In this case it will need to wake up periodically to see if the master is trying to send it any messages. It may also wake up due some external event (such as motion in a motion detector) and send a message to the master (or a peer) informing them of the event.

When an application wants to send a packet or listen for a packet, it must call one of the two ONE-NET main functions depending on whether the application is a master or a client. The two main functions for executing the ONE-NET protocol are one_net_master() and one_net_client(). These functions return a tick_t value that indicates how long the application code can wait before the ONE-NET function needs to be called again.

If ONE-NET has a message to transmit (either an application message or an administrative message), it will turn on the transceiver's transmitter and transmit the message. If ONE-NET is expecting a response to the a message it sent, it will turn on the transceiver's receiver and wait for the message. If ONE-NET does not have a message to transmit, it will turn on the transceiver's receiver and listen for messages. ONE-NET waits for a message in ONE_NET_WAIT_FOR_SOF_TIME millisecond increments (currently set to 10 in the file one_net_port_const.h).

The current implementation of ONE-NET is based on the processor communicating with the transceiver at the bit level (aka bit banging). With this style of communications between the processor and the transceiver, no ONE-NET messages will be heard unless the ONE-NET main function is being executed.

ONE-NET Running Under an RTOS

The current implementation of ONE-NET, as distributed, was designed to run on low cost single function devices. It was not designed to operate in a Real Time Operating System (RTOS). There are no multitasking hooks in the current implementation. In the future, people may want to run the ONE-NET protocol in a multitasking environment and the code will be enhanced to support a multitasking environment.

The current implementation of the ONE-NET protocol is based on bit level communications with a transceiver. This implies that for many processors most of the processing power is consumed by i/o with the transceiver. Some transceivers support higher level communications with the processor. These transceivers may support a FIFO that can collect received bytes and pass them to the processor at higher speeds than the speed at which the ONE-NET protocol is operating. In this situation, more of the processor's time may be available for functions other than supporting the ONE-NET protocol. To take full advantage of this processing power a modified interface between the application code and ONE-NET code would be needed.