wireless control for everyone

ONE-NET DEBUGGING TOOLS AND TECHNIQUES

Version 2.3.0

August 14, 2012

Table of Contents

1 Overview

ONE-NET provides support for a wide variety of networks and devices. When setting up a ONE-NET network, one should consider what types of devices will be in this network and how the network will be used. This document discusses different options for devices and networks and the corresponding options in the code that need to match these options.

2 Intended Audience

This document is geared mainly towards students wishing to learn the ONE-NET packet structure, programmers, engineers, system designers, system testers, and anyone who wants to monitor or analyze ONE-NET packets in detail. Generally this document will be most useful when developing and debugging ONE-NET networks and devices. Most of the debugging options will not be present in a completed, functional network.

3 Adjusting the verbosity Level Of Printouts

When debugging, it can be useful to print very detailed messages. At the same time, very detailed printouts can interfere with program functionality and can be overwhelming (there is such a thing as too much detail in printouts). ONE-NET allows you to adjust the verbosity level of printouts with the following command.

```
verbose level:value
```

where value is a non-negative integer. The higher the value, the more detailed the printout. The following are examples using different verbosity levels.

Verbosity Level 1

```
ocm-m> set pin:002:3:1

OK

ocm-m> Single transaction with 002; return status: SUCCESS

Unit 3 of 002 has state 1.


ocm-c> Pin 3 has changed to state 1
```

Verbosity Level 6

```
ocm-m> set pin:002:3:1

OK

ocm-m> ehanr : NACK : Nack Reason-->0x0E(No Resp) : Handle-->0x05(TIMEOUT MS) :
Payload : 50 ms


ehanr : NACK : Nack Reason-->0x0E(No Resp) : Handle-->0x05(TIMEOUT MS) : Payload : 50
ms


ehanr : NACK : Nack Reason-->0x0E(No Resp) : Handle-->0x05(TIMEOUT MS) : Payload : 50
ms


ehanr : ACK : Nack Reason-->0x00(No Err) : Handle-->0x0D(APP) : Payload :

App payload : 0x0043F00001 : Type-->0x00 : Class-->0x4 : Src Unit-->0x3 : Dst Unit--
>0xF : Data-->1


ests star:Hdr-->PID=0x100,Msg ID=0x30,Msg Type=0x0(App)

Message Data:005F300001, retry=3,hops=0,dst=0020

ack_nack-->ACK : Nack Reason-->0x00(No Err) : Handle-->0x0D(APP) : Payload :

App payload : 0x0043F00001 : Type-->0x00 : Class-->0x4 : Src Unit-->0x3 : Dst Unit--
>0xF : Data-->1


Single transaction with 002; return status: SUCCESS

ests end

Unit 3 of 002 has state 1.

Device 002 has checked in.

ocm-m>



ocm-c> eval_hdl_sng: App payload : 0x005F300001 : Type-->0x00 : Class-->0x5 : Src
Unit-->0xF : Dst Unit-->0x3 : Data-->1
```

```
Pin 3 has changed to state 1

eval_hdl_sng: App payload : 0x005F300001 : Type-->0x00 : Class-->0x5 : Src Unit-->0xF
: Dst Unit-->0x3 : Data-->1


Pin 3 has changed to state 1

eval_hdl_sng: App payload : 0x005F300001 : Type-->0x00 : Class-->0x5 : Src Unit-->0xF
: Dst Unit-->0x3 : Data-->1


Pin 3 has changed to state 1
```

A much higher level of detail is available at this higher level. For instance, it is immediately apparent what how many and what type of packets have been received by each device, and just as important, it is apparent that packets have NOT been received several times. For example, see this part of the master's printout…

```
ocm-m> ehanr : NACK : Nack Reason-->0x0E(No Resp) : Handle-->0x05(TIMEOUT MS) :
Payload : 50 ms


ehanr : NACK : Nack Reason-->0x0E(No Resp) : Handle-->0x05(TIMEOUT MS) : Payload : 50
ms


ehanr : NACK : Nack Reason-->0x0E(No Resp) : Handle-->0x05(TIMEOUT MS) : Payload : 50
ms


ehanr : ACK : Nack Reason-->0x00(No Err) : Handle-->0x0D(APP) : Payload :

App payload : 0x0043F00001 : Type-->0x00 : Class-->0x4 : Src Unit-->0x3 : Dst Unit--
>0xF : Data-->1
```

Note that "ehanr" stands for eval_handle_ack_nack_response. It is the response handler for the Evaluation Board project and is called every time a packet receives a response or times out without a response. From this, we can determine that the master sent three packets and received no response, then finally on the fourth packet, it received a response. This can provide useful clues when debugging. The packets are timing out.

Sniffed packets can also have different levels of verbosity.

Verbosity Level 1

```
1156225 received 30 bytes:
55 55 55 33 B4 BC 66 B4 B3 B4 B4 B4 B4 B4 BC B4
BC B5 B4 D6 59 93 A9 6A D5 B2 A9 34 94 BC
```

Verbosity Level 6

```
1195851 received 30 bytes:
55 55 55 33 B4 BC 96 B4 B3 B4 B4 B4 B4 B4 BC B4
BC B5 B4 D9 39 56 D3 C6 B4 A5 DA AA D3 39

Raw PID=100(00)--MH=False SA=False

Enc. Msg CRC : 0x96 -- Decoded Msg. CRC : 0xB8
Calculated Raw Msg CRC : 0xB8
Repeater DID : 0xB4BC -- Decoded : 0x001
Dest. DID : 0xB4B3 -- Decoded : 0x002
NID : 0xB4B4B4B4B4BC -- Decoded : 0x000000001
Source DID : 0xB4BC -- Decoded : 0x001
Encoded Payload Length : 11
D9 39 56 D3 C6 B4 A5 DA AA D3 39
Decoded Payload (# of Bytes = 9)
F5 59 BA 38 07 3B 6F A5 40
Decrypted using key (00-01-02-03) - (04-05-06-07) - (08-09-0a-0b) - (0c-0d-0e-0f)
3A 03 30 00 5F 30 00 01
Payload CRC = 0x3A, Calculated Payload CRC = 0x3A, CRCs match.
Msg. ID=033
Single -- Msg Type=0x0(App)
App payload : 0x005F300001 : Type-->0x00 : Class-->0x5 : Src Unit-->0xF : Dst Unit--
>0x3 : Data-->1
Decrypted using key (04-05-06-07) - (08-09-0a-0b) - (0c-0d-0e-0f) - (44-44-44-44)
D8 4D D8 A9 3E CD 3B 28
Payload CRC = 0xD8, Calculated Payload CRC = 0x9A, CRCs do not match.
Msg. ID=4DD
```

Using verbosity level 1, the packet has been decoded, decrypted, parsed, and tested for validity. Sniffed packets with verbosity levels of 2, 3, 4, and 5 will have greater detail than sniffed packets with verbosity level 1, but less detail than sniffed packets of verbosity level 6.

3.1 Evaluation Board Sniffer Default Keys

The Evaluation Board requires default keys to attempt decryption with for high verbosity sniffing. These are determined at compile-time. They are located in the file sniff_eval.c and can and should be changed if desired. An array of both invite and non-invite keys to try, as well as the array sizes are all that needs to be changed. Relevant portions of sniff_eval.c are as follows. Change, recompile, and re-download as needed…

```
#if DEBUG_VERBOSE_LEVEL > 2
enum
{
    //! number of known invite keys to try for decryption.
    NUM_SNIFF_INVITE_KEYS = 2,

    //! number of known encryption keys to try for decryption.
    NUM_SNIFF_ENCRYPT_KEYS = 2,
};
#endif
#if DEBUG_VERBOSE_LEVEL > 2
//! Place any known invite keys in the array below
static const one_net_xtea_key_t sniff_invite_keys[NUM_SNIFF_INVITE_KEYS] =
{
    {'2','2','2','2','2','2','2','2','2','2','2','2','2','2','2','2'},
    {'3','3','3','3','3','3','3','3','3','3','3','3','3','3','3','3'}
};

//! Place any known encryption keys in the array below
static const one_net_xtea_key_t sniff_enc_keys[NUM_SNIFF_ENCRYPT_KEYS] =
{
    {0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,
     0x08,0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F},

    // use with "change key:44-44-44-44" command
    {0x04,0x05,0x06,0x07,0x08,0x09,0x0A,0x0B,
     0x0C,0x0D,0x0E,0x0F,0x44,0x44,0x44,0x44}
};
#endif
```

There are two default invite keys that the sniffer will attempt to decrypt with: 2222-2222 and 3333-3333. Change if needed. Note that 2222-2222 is the default invite key for the Evaluation Board project.

There two default encryption keys that the sniffer will attempt to decrypt with:

1. 00-01-02-03-04-05-06-07-08-09-0A-0B-0C-0D-0E-0F
2. 04-05-06-07-08-09-0A-0B-0C-0D-0E-0F-44-44-44-44

Note that the first twelve bytes are the same in each. The first is the default encryption key for the Evaluation Board project. The second is 44-44-44-44. If a "change key:44-44-44-44" command is given, the sniffer will be able to decrypt the entire process.

```
225884 received 30 bytes:
55 55 55 33 B4 BC C3 B4 B3 B4 B4 B4 B4 B4 BC B4
BC B5 B4 BA 5C 62 CA B5 99 B6 35 56 BC C9

Raw PID=100(00)--MH=False SA=False

Enc. Msg CRC : 0xC3 -- Decoded Msg. CRC : 0x28
Calculated Raw Msg CRC : 0x28
Repeater DID : 0xB4BC -- Decoded : 0x001
Dest. DID : 0xB4B3 -- Decoded : 0x002
NID : 0xB4B4B4B4B4BC -- Decoded : 0x000000001
Source DID : 0xB4BC -- Decoded : 0x001
Encoded Payload Length : 11
BA 5C 62 CA B5 99 B6 35 56 BC C9
Decoded Payload (# of Bytes = 9)
0E 1D CB 12 D1 94 98 13 40
Decrypted using key (00-01-02-03) - (04-05-06-07) - (08-09-0a-0b) - (0c-0d-0e-0f)
66 00 74 02 44 44 44 44
Payload CRC = 0x66, Calculated Payload CRC = 0x66, CRCs match.
Msg. ID=007
Single -- Msg Type=0x4(Admin)
Admin type : 02 : Admin payload : 0x44444444
Decrypted using key (04-05-06-07) - (08-09-0a-0b) - (0c-0d-0e-0f) - (44-44-44-44)
EF 05 2D 45 82 67 8A 1B
```

```
Payload CRC = 0xEF, Calculated Payload CRC = 0x9A, CRCs do not match.
Msg. ID=052





226045 received 30 bytes:
55 55 55 33 B4 B3 92 B4 BC B4 B4 B4 B4 B4 BC B4
B3 B5 B3 33 9C 6A 69 DA B4 56 C2 69 65 69

Raw PID=102(02)--MH=False SA=False

Enc. Msg CRC : 0x92 -- Decoded Msg. CRC : 0xBC
Calculated Raw Msg CRC : 0xBC
Repeater DID : 0xB4B3 -- Decoded : 0x002
Dest. DID : 0xB4BC -- Decoded : 0x001
NID : 0xB4B4B4B4B4BC -- Decoded : 0x000000001
Source DID : 0xB4B3 -- Decoded : 0x002
Encoded Payload Length : 11
33 9C 6A 69 DA B4 56 C2 69 65 69
Decoded Payload (# of Bytes = 9)
4A 9C F5 EC 09 8F D7 4D 40
Decrypted using key (00-01-02-03) - (04-05-06-07) - (08-09-0a-0b) - (0c-0d-0e-0f)
CA 00 73 0F 00 00 00 28
Payload CRC = 0xCA, Calculated Payload CRC = 0xCA, CRCs match.
Msg. ID=007
NACK : Nack Reason-->0x0F(Inv Msg ID) : Handle-->0x03(VLUE) : Payload : 40

Decrypted using key (04-05-06-07) - (08-09-0a-0b) - (0c-0d-0e-0f) - (44-44-44-44)
7C B7 02 96 34 33 CE 98
Payload CRC = 0x7C, Calculated Payload CRC = 0x26, CRCs do not match.
Msg. ID=B70





226394 received 30 bytes:
55 55 55 33 B4 BC C3 B4 B3 B4 B4 B4 B4 B4 BC B4
BC B B4 BA 5C 62 CA B5 99 B6 35 56 BC C9

Raw PID=100(00)--MH=False SA=False
```

```
Enc. Msg CRC : 0xC3 -- Decoded Msg. CRC : 0x28
Calculated Raw Msg CRC : 0x28
Repeater DID : 0xB4BC -- Decoded : 0x001
Dest. DID : 0xB4B3 -- Decoded : 0x002
NID : 0xB4B4B4B4B4BC -- Decoded : 0x000000001
Source DID : 0xB4BC -- Decoded : 0x001
Encoded Payload Length : 11
BA 5C 62 CA B5 99 B6 35 56 BC C9
Decoded Payload (# of Bytes = 9)
0E 1D CB 12 D1 94 98 13 40
Decrypted using key (00-01-02-03) - (04-05-06-07) - (08-09-0a-0b) - (0c-0d-0e-0f)
66 00 74 02 44 44 44 44
Payload CRC = 0x66, Calculated Payload CRC = 0x66, CRCs match.
Msg. ID=007
Single -- Msg Type=0x4(Admin)
Admin type : 02 : Admin payload : 0x44444444
Decrypted using key (04-05-06-07) - (08-09-0a-0b) - (0c-0d-0e-0f) - (44-44-44-44)
EF 05 2D 45 82 67 8A 1B
Payload CRC = 0xEF, Calculated Payload CRC = 0x9A, CRCs do not match.
Msg. ID=052




226556 received 30 bytes:
55 55 55 33 B4 B3 92 B4 BC B4 B4 B4 B4 B4 BC B4
B3 B5 B3 33 9C 6A 69 DA B4 56 C2 69 65 69

Raw PID=102(02)--MH=False SA=False

Enc. Msg CRC : 0x92 -- Decoded Msg. CRC : 0xBC
Calculated Raw Msg CRC : 0xBC
Repeater DID : 0xB4B3 -- Decoded : 0x002
Dest. DID : 0xB4BC -- Decoded : 0x001
NID : 0xB4B4B4B4B4BC -- Decoded : 0x000000001
Source DID : 0xB4B3 -- Decoded : 0x002
Encoded Payload Length : 11
33 9C 6A 69 DA B4 56 C2 69 65 69
Decoded Payload (# of Bytes = 9)
4A 9C F5 EC 09 8F D7 4D 40
Decrypted using key (00-01-02-03) - (04-05-06-07) - (08-09-0a-0b) - (0c-0d-0e-0f)
CA 00 73 0F 00 00 00 28
```

```
Payload CRC = 0xCA, Calculated Payload CRC = 0xCA, CRCs match.

Msg. ID=007

NACK : Nack Reason-->0x0F(Inv Msg ID) : Handle-->0x03(VALUE) : Payload : 40


Decrypted using key (04-05-06-07) - (08-09-0a-0b) - (0c-0d-0e-0f) - (44-44-44-44)

7C B7 02 96 34 33 CE 98

Payload CRC = 0x7C, Calculated Payload CRC = 0x26, CRCs do not match.

Msg. ID=B70





226717 received 30 bytes:

55 55 55 33 B4 BC C9 B4 B3 B4 B4 B4 B4 B4 BC B4

BC B5 B4 C4 9A CA 6C 92 33 C9 AC B5 B9 9C


Raw PID=100(00)--MH=False SA=False


Enc. Msg CRC : 0xC9 -- Decoded Msg. CRC : 0x34

Calculated Raw Msg CRC : 0x34

Repeater DID : 0xB4BC -- Decoded : 0x001

Dest. DID : 0xB4B3 -- Decoded : 0x002

NID : 0xB4B4B4B4B4BC -- Decoded : 0x000000001

Source DID : 0xB4BC -- Decoded : 0x001

Encoded Payload Length : 11

C4 9A CA 6C 92 33 C9 AC B5 B9 9C

Decoded Payload (# of Bytes = 9)

22 B2 F1 BD 23 59 10 5A 40

Decrypted using key (00-01-02-03) - (04-05-06-07) - (08-09-0a-0b) - (0c-0d-0e-0f)

E6 02 84 02 44 44 44 44

Payload CRC = 0xE6, Calculated Payload CRC = 0xE6, CRCs match.

Msg. ID=028

Single -- Msg Type=0x4(Admin)

Admin type : 02 : Admin payload : 0x44444444Decrypted using key (04-05-06-07) - (08-09-0a-0b) -
(0c-0d-0e-0f) - (44-44-44-44)

3F BB 75 1C 6A 21 63 67

Payload CRC = 0x3F, Calculated Payload CRC = 0xF0, CRCs do not match.

Msg. ID=BB7





227225 received 30 bytes:
```

```
55 55 55 33 B4 BC C9 B4 B3 B4 B4 B4 B4 B4 BC B4
BC B5 B4 C4 9A CA 6C 92 33 C9 AC B5 B9 9C

Raw PID=100(00)--MH=False SA=False

Enc. Msg CRC : 0xC9 -- Decoded Msg. CRC : 0x34
Calculated Raw Msg CRC : 0x34
Repeater DID : 0xB4BC -- Decoded : 0x001
Dest. DID : 0xB4B3 -- Decoded : 0x002
NID : 0xB4B4B4B4B4BC -- Decoded : 0x000000001
Source DID : 0xB4BC -- Decoded : 0x001
Encoded Payload Length : 11
C4 9A CA 6C 92 33 C9 AC B5 B9 9C
Decoded Payload (# of Bytes = 9)
22 B2 F1 BD 23 59 10 5A 40
Decrypted using key (00-01-02-03) - (04-05-06-07) - (08-09-0a-0b) - (0c-0d-0e-0f)
E6 02 84 02 44 44 44 44
Payload CRC = 0xE6, Calculated Payload CRC = 0xE6, CRCs match.
Msg. ID=028
Single -- Msg Type=0x4(Admin)
Admin type : 02 : Admin payload : 0x44444444
Decrypted using key (04-05-06-07) - (08-09-0a-0b) - (0c-0d-0e-0f) - (44-44-44-44)
3F BB 75 1C 6A 21 63 67
Payload CRC = 0x3F, Calculated Payload CRC = 0xF0, CRCs do not match.
Msg. ID=BB7




55 55 55 33 B4 B3 C4 B4 BC B4 B4 B4 B4 B4 BC B4
B3 B5 BC 62 A9 34 3C 55 62 B6 52 63 54 AC

Raw PID=101(01)--MH=False SA=False

Enc. Msg CRC : 0xC4 -- Decoded Msg. CRC : 0x20
Calculated Raw Msg CRC : 0x20
Repeater DID : 0xB4B3 -- Decoded : 0x002
Dest. DID : 0xB4BC -- Decoded : 0x001
NID : 0xB4B4B4B4B4BC -- Decoded : 0x000000001
Source DID : 0xB4B3 -- Decoded : 0x002
Encoded Payload Length : 11
62 A9 34 3C 55 62 B6 52 63 54 AC
```

```
Decoded Payload (# of Bytes = 9)
DD D4 11 93 71 A7 CA 06 40
Decrypted using key (00-01-02-03) - (04-05-06-07) - (08-09-0a-0b) - (0c-0d-0e-0f)
B9 46 65 38 60 98 62 DA
Payload CRC = 0xB9, Calculated Payload CRC = 0x48, CRCs do not match.
Msg. ID=466
Decrypted using key (04-05-06-07) - (08-09-0a-0b) - (0c-0d-0e-0f) - (44-44-44-44)
BE 02 8A 44 44 44 44 44
Payload CRC = 0xBE, Calculated Payload CRC = 0xBE, CRCs match.
Msg. ID=028
ACK : Nack Reason-->0x00(No Err) : Handle-->0x0A(KEY) : Payload : 4444444444
```

When in doubt about the validity of a message or which key printout to look at, look at the CRC. You should see either "CRCs do not match." Or "CRCs match.". The payload is only valid if the CRCs match. Hence a good way to check which key is used is to check which CRC is used and look at that printout.

3.2 Verbosity Levels And Timing

Verbosity levels can affect the actual message flow. It takes time to output characters to the serial port. During that time, other messages could be missed and processing slows down while data is displayed. Hence when sniffing with high verbosity levels, many packets can be missed. In addition, even when not sniffing, the extra time it takes to print can cause timeouts to be missed. Hence developers should be aware that high verbosity can not only DETECT errors, but can actually CAUSE errors. The most obvious example is an example mentioned earlier in the document. A command to set a pin was sent. The high verbosity level caused a slowdown sufficient to cause several timeouts and packet re-sends. Had the verbosity level been lower, the packet would have been sent and received within the 50 ms response timeout period.

There are several options that can be used to deal with missing packets using the sniffer. They are covered briefly in this section, then more thoroughly later in the document.

1. Filter Packets – The sniffer can throw away packets based on certain criteria. You can filter on the Packet type or the Repeater DID of the packets. Assume that 001 is sending 002 a command application message.
    a. Filtering on the PID – Have one sniffer device only listen to data packets. Have another sniffer device only listen to response packets. Between the two sniff logs, no packet should be lost.
    b. Filtering on the Repeater DID of the packet. Have one sniffer device only listen to 001. Have another sniffer device only listen to 002. Between the two sniff logs, no packet should be lost.
2. Add a “write pause” and increase the response timeout. – These are both done using the “interval” command. By default there is no write pause and the response timeout is 50 milliseconds. If the write pause is changed to 200 milliseconds and the responds timeout to 400 milliseconds on both 001 and 002, a single sniffer should be able to capture all packets, even with high verbosity.
3. Slow down the clock with the “csdf” (Clock Slow-Down Factor).
4. Parse the packets using minimal verbosity, then run the packets through the C++ command line sniffer on a desktop computer.

3.3 Options to define in config_options.h

Note that DEBUGGING_TOOLS does NOT have to be defined for the “verbose level” command.

However, make sure that DEBUG_VERBOSE_LEVEL is defined as a high enough value. For example, if you want to use a verbosity level of 6, the DEBUG_VERBOSE_LEVEL compile-time #define option must be defined as at least 6.

To use the sniffer, SNIFFER_MODE must be defined.

4 Pausing And Slowing Down Transactions

Computer chips run faster than the human eye can process events. You send a quick “set pin” message. It’s over before you can tell what happens. Did device 002’s transmission LED block three times or four? Was it blinking every 20th of a second or every twentieth of a second? It’s hard to tell. Very often developers want to change something on one device while another is transmitting and see what effect that has, if any. Usually we are not fast enough.

If you are using ONE-NET with an IDE with the ability to adjust memory, slow down processor speed, add breakpoints, etc., this section may not be useful. This section is intended for situations where that capability is not present.

We would like to be able to pause or slow things down. First, let’s slow things down.

4.1 Slowing Down The Clock – “Clock Slow Down Factor”

ONE-NET can “slow down” the clock to make transactions seem longer. Let’s assume that you want one second to seem like it takes ten seconds. You should define a “clock slow-down factor” of 10. For every ten processor ticks, the official ONE-NET “tick count”, which keeps track of all timers, will only increase by 1.

4.1.1 Adjusting “Clock Slow Down Factor” without using the debugging tools

If DEBUGGING_TOOLS is not defined, you can still have a clock-slow-down factor as described. However, this will be a compile-time constant and hence unadjustable at run-time.

1. Define CLOCK_SLOW_DOWN in config_options.h
2. Define CLOCK_SLOW_DOWN_FACTOR as an integer value greater than or equal to 2.

4.1.2 Adjusting “Clock Slow Down Factor” using the debugging tools

If DEBUGGING_TOOLS is defined, the values above are ignored. Instead, use the “csdfq” command.

`csdq:10` -- Ten processor tick equals an increase of 1 in the ONE-NET tick count variable. ONE-NET will slow down.

Note that since this a command-line option, the clock slow-down factor can be changed at any time. The default is 1, which means there is no slow-down. Change it to this value if you want things to run as normal.

4.2 Pausing And Unpausing

ONE-NET provides three command line options for pausing ONE-NET activity

1. ratchet – When ratcheting is on, execution will pause every time a packet is about to be sent.
2. proceed – Releases the “ratchet” (packet is written)
3. pause - Pauses when a packet is about to be written. Unlike “ratchet”, this is a ONE-TIME pause.

You might use the “ratchet” command to only send one packet out at a time. For example, if “ratchet” is enabled, instead of sending an invite packet every 250 ms, a packet will be sent every time the “proceed” command is issued.

The “pause”, “ratchet”, and “proceed” commands take no arguments. “pause” and “ratchet” are toggling commands. Hence typing “ratchet” or “pause” once turns ratcheting or pausing on. Typing it again turns them off.

4.3 Options to define in config_options.h

DEBUGGING_TOOLS must be defined in config_options.h.

5 Viewing / Changing Volatile Memory

The “memory”, “memdump”, “interval”, and “memload” commands allow you to view and change memory. First a memory segment must be chosen using the “memory” command. Options of memory to adjust include the following. An asterisk at the end of the word indicates that the memory segment is an array. A # sign indicates “Master Only”. A $ indicates client only.

1. on_state
2. timer*
3. send_list*$
4. master$
5. peer
6. base_param
7. client_list*#
8. invite_txn
9. response_txn
10. single_txn
11. bs_txn
12. bs_msg

5.1 “memory” command

The “memory” command takes one, two or three arguments. Non-array elements cannot take three arguments. The first argument is one of the words above specifying the memory type. Note the “master-only” and “client-only” values. For array values, you can specify an array index as a parameter (optional). For the last parameter, you can specify an offset (also optional). Examples are as follows.

```
ocm-m> memory:timer
Address = (00:046C) : Length = 70
OK
ocm-m> memory:timer:5
Address = (00:0485) : Length = 5
OK
ocm-m> memory:peer
Address = (00:0718) : Length = 32
OK
ocm-m> memory:peer:3
Address = (00:0724) : Length = 4
OK
ocm-m> memory:timer:5:2
Address = (00:0487) : Length = 3
OK
ocm-m> memory:base_param
Address = (00:05E8) : Length = 39
OK
ocm-m> memory:base_param:3
Address = (00:05EB) : Length = 36
OK
```

A quick look at the results…

```
ocm-m> memory:peer
Address = (00:0718) : Length = 32
OK
```

on_peer_unit is a four byte structure. The peer variable is an array of eight on_peer_unit elements. Hence the size is 32 bytes, as seen above. The address of the head of the array is 00:0718. Now to specify a specific element, we add a “3”.

```
ocm-m> memory:peer:3
Address = (00:0724) : Length = 4
OK
```

Now the memory is pointing to 00:0718, which is the address of the 3rd element in the array. Note that the length is now 4. We are only looking at one particular element, not the array.

Now we look at a non-array (the base parameters). A quick check of the on_base_param_t structure will show it has 3 bytes

```
ocm-m> memory:base_param
Address = (00:05E8) : Length = 39
OK
ocm-m> memory:base_param:3
Address = (00:05EB) : Length = 36
OK
```

We now added an offset of 3. The memory address (00:05EB) is now three bytes beyond what it was before and the length is 3 bytes short.

5.2 “memdump” command

The “memdump” command takes no arguments. It simply dumps the memory that has been pointed to by the “memory” command. You cannot run the “memdump” command without having run the “memory” command first. It will display all timers, intervals, the current state of the pausing variables (see Section 4.2), and the hex bytes of the memory.

```
ocm-m> memdump
Ratchet = off
Proceed = off
Pause = off
Pausing = off


Timer 0:        inactive        0
Timer 1:        inactive        0
Timer 2:        inactive        0
Timer 3:        active          0
Timer 4:        active          0
Timer 5:        inactive        0
Timer 6:        inactive        0
Timer 7:        active          578590
Timer 8:        active          39
Timer 9:        inactive        0
Timer 10:       active          0
Timer 11:       inactive        0
Timer 12:       inactive        0
Timer 13:       inactive        0
Interval 0(Resp. Timeout):      50
Interval 1(Write Pause):        0
Interval 2(Invite Send Time):   250
Interval 3(Channel Scan Time):  10000
Interval 4(Invite Trans. Timeout):      10000


00:05E8: 00 09 00 b4 b4 b4 b4 b4 bc b4 bc 01 6f c1 38 87
00:05F8: 00 00 00 00 00 00 01 02 03 04 05 06 07 08 09 0a
00:0608: 0b 0c 0d 0e 0f 02 01
OK
```

5.2.1 Intervals

Five intervals are defined on the master, three on clients.

1. Response Timeout – Amount of time that a device waits for a response before assuming the packet was lost (default = 50 ms) – see ONE_NET_RESPONSE_TIME_OUT in the code.
2. Write Pause – For Debugging only. The amount of time to wait before sending a packet. Generally packets are sent immediately. Adding a write pause will cause a lag, useful for timing experiments and sniffing.
3. Invite Send Time – Master Only. Invite Packets are sent out at this interval. Default is 250 ms. See ONE_NET_MASTER_INVITE_SEND_TIME in the code.
4. Channel Scan Time – Master Only. Used to find clear channels. See ONE_NET_MASTER_CHANNEL_SCAN_TIME in the code. Default is 10 seconds.

5. Invite Transaction Timeout – The time allowed to complete an invite transaction AFTER the first response is sent by the joining client. Default is 10 seconds. See INVITE_TRANSACTION_TIMEOUT in the code.

5.2.2 Timers

Timers correspond to the timers defined in one_net_timer.h. For example, on the Evaluation Board, The Invite Timer is number 7. Note that the value is 578,590 ms, or 579 seconds, or 21 seconds less than 10 minutes. This is good because an invite command with a timeout of ten minutes was sent 21 seconds before that printout.

5.2.3 Ratchet / Pause / Proceed / Pausing Values

The “ratchet”, “pause”, and “proceed” commands do not have output, so their values are listed with the “memdump” command. “Pausing” shows wehther the chip is currently paused.

5.2.4 Hex Value Raw Memory

Finally, there is the raw memory in hexadecimal. Note that no endian adjustment is made in the printout, so developers need to go the structure to determine offsets and values, plus do any endianness adjustments. A quick “list” command gives the following.

```
ocm-m> list
ONE-NET Evaluation Version 2.3.0 (Build 107)
# of Network MH Devices : 2
# of Network MH Repeaters : 1
Message key : (00-01-02-03) - (04-05-06-07) - (08-09-0a-0b) - (0c-0d-0e-0f)
Old Message key : (00-00-00-00) - (00-01-02-03) - (04-05-06-07) - (08-09-0a-0b)

NID: 0x000000001
DID: 0x001

Channel: US 2
Client count: 1

  Client 1 : DID: 0x002

Peer table:
  001:0:002:1
  001:2:002:3

User pins:
  0 output state: 0
  1 input state: 1
  2 output state: 0
  3 input state: 1
OK
ocm-m>
```

A quick check of the bytes in the listing shows that the bytes line up with the listing (see the on_base_param_t structure in the code). For example, the last two bytes are the number of multi-hop-capable and multi-hop-repeater-capable devices. Note that they match (see red below and above).

```
00:05E8: 00 09 00 b4 b4 b4 b4 b4 bc b4 bc 01 6f c1 38 87
00:05F8: 00 00 00 00 00 00 01 02 03 04 05 06 07 08 09 0a
00:0608: 0b 0c 0d 0e 0f 02 01
OK
```

1. How crowded is the radio traffic? An area with lots of traffic is going to result in far more lost messages and the need to handle lost messages will increase.
2. How important is it that messages are not lost? In some cases the consequences of a lost message will be quite severe. In other cases, it will not be a big problem. The answer to this question will determine whether a message that does not get acknowledged will be sent again.
3. How many devices are in the network? How often do most devices send messages? The answers to both of these questions will have a big effect on how crowded the radio traffic is. A key change in a network with two devices can easily propogate through the entire network quickly. A key change in a network with a thousand devices, most of which are asleep the majority of the time, will be much more complicated?
4. What are the devices' resources? Do all devices have the same resources? A simple client with 16K code space and 1K RAM and a transceiver that can only communicate at 38,400 baud cannot do what a more advanced device with 128K code space. 8K of RAM, and the ability to switch data rates can do. If all devices have the ability to function as multi-hop repeaters and switch data rates, life is much easier. If a simple device cannot switch data rates and must be able to communicate constantly with a device that can switch data rates, the device that can switch data rates will not be able to even though it can. The simple device will also be unable to handle multi-hop messages so that the range of devices that it can communicate will be smaller. The simple device will not be able to handle sophisticated timing messages so all network communication must take this into consideration. This may or may not be problematic, but it is something the programmers and network designers must consider. ONE-NET offers many advanced communication options. Simple devices with low resources will not be able to use all of them.
5. Do any devices sleep? How critical is battery life? ONE-NET is generally geared towards meeting the needs of low-resource battery-powered sensors which are asleep the vast majority of the time and wake up once an hour or once a day, report their status, and immediately go back to sleep. The prime consideration is to minimize battery consumption. This can have a drastic

effect on the rest of the network. One, the radio waves must be clear when these devices check in order to minimize lost messages and hence preserve battery power. Two, devices which are primarily sleeping cannot receive messages when they are asleep, so administration messages like key changes must consider this. This may or may not have an effect on the other devices. If you have an intercom, for example, that device will be initiating many stream messages. If a stream message is initiated during the same period that a sleeping device is about to wake up, the master may decide to disallow the stream message because that message will interfere with the sleeping device's status report. The network designer must decide which message is more vital or if it matters. There is no correct answer. It will depend on the needs of the network.

6. How important is security? How important is it that messages get sent and received immediately? One of the primary concerns of ONE-NET is the prevention of replay attacks. These are vital for applications such as locking and unlocking locks. It is generally not nearly as important for operations such as reporting humidity or turning up the volume on a stereo. One of the primary ways that ONE-NET prevents replay attacks is to make sure that no two legitimate commands are sent with the same payload. The main way this is done is to change the message ID after each command. When message IDs are about to run out, a device will request a key change and reset the Message IDs to a lower value. The key change must propagate to the entire network. During that propagation, potentially two devices might not be able to communicate with each other because one has the new key and the other does not. The network designer must decide how important replay attacks are and determine whether lost messages due to key changes outweigh this concern.
7. How far away are the devices from each other? How long are the packets that need to be sent? Long packets reduce the range of a wireless message. If devices are close to each other, this will not be a problem. If devices are not close to each other, one or more client repeaters may be needed to transmit the message. In addition, devices with low resource may not be able to handle long packets. Any devices which communicate with a device that cannot handle long packets needs to know this.
8. How important is portability? Is this a "contained" network? If a ONE-NET network is going to be filled entirely with devices from the same manufacturer, the network designer can make all sorts of data messages and parsing mechanisms that are not specified by the ONE-NET protocol and all

devices will know how to handle these messages. The goal of ONE-NET is generally to make the messages as universal as possible, so that a door/window sensor from one company can communicate with a wireless chime from another company and both can communicate with a master controller from a third company. To do that, all messages must comply to one of the ONE-NET defined message types and parsing mechanisms. There is a tradeoff. The ONE-NET specification allows for a significant variety of define messages, but there will still be times when a developer wants to add an internal message type undefined by ONE-NET in order to make the network run better. Doing so takes portability away. The network / product designer must take this into consideration.

9. What is the topology / "shape" of the network? Does almost all communication involve the master? Is the master as uninvolved as possible? If multi-hop communication is involved? If so, what devices are within range of what other devices and what devices can serve as repeaters? Is the network largely unchanging or are nodes moving in and out of range or constantly added and removed?
10. How intricate are the timing of the messages? Are messages obsolete if delayed or should they still be sent, even if late? What are the ramifications?
11. Is ONE-NET the right protocol? ONE-NET is not a good fit for all applications. It has a much slower data rate than other protocols. If you need lots of data transferred quickly and reliably, you may need to consider another protocol. Similarly, if your network is quite large and needs routing tables, you may need to consider another protocol. ONE-NET is excellent for networks with low power needs, long range needs, where the bulk of the messages are short messages.

5.3 "memload" command

The "memload" command loads data into memory. It takes ASCII hex byte pairs as input.

For example, let's change the source and destination units on the peer table. First, a listing (only the peer table is shown).

```
Peer table:
  001:0:002:1
  001:2:002:3
```

Now point the memory to the peer table. Relevant entries are in red below. Let's change it so the second entry goes from 5 to 6 instead of 2 to 3.

```
00:0718: b4 b3 00 01 b4 b3 02 03 b4 b4 0f 0f b4 b4 0f 0f
00:0728: b4 b4 0f 0f b4 b4 0f 0f b4 b4 0f 0f b4 b4 0f 0f
```

The blue area is what we want to change.

```
00:0718: b4 b3 00 01 b4 b3 02 03 b4 b4 0f 0f b4 b4 0f 0f
00:0728: b4 b4 0f 0f b4 b4 0f 0f b4 b4 0f 0f b4 b4 0f 0f
```

Re-set the memory, specifying an array index and offset. A quick memdump confirms we are pointed to the right spot.

```
ocm-m> memory:peer:1:2
Address = (00:071E) : Length = 2
OK
ocm-m> memdump
Address = (00:071E) : Length = 2
OK
00:071E: 02 03
OK
ocm-m>
```

Now load the new memory (0506). Do not add spaces. Then quickly follow up with memdump and list commands to verify. Only relevant sections are shown.

```
ocm-m> memload:0506
Copied 2 bytes.
OK
```

```
ocm-m> memdump

00:071E: 05 06
OK
ocm-m> list

Peer table:
  001:0:002:1
  001:5:002:6
```

5.4 Adjusting Timers

As mentioned, timer 7 is the invite timer. The ont_timer_t structure lists the tick counts remaining as a 4 byte unsigned integer (tick_t). On the Evaluation Board, there is one tick per millisecond, so the tick count and the time in milliseconds is equivalent. Point the memory to that…

```
ocm-m> memory:timer:7:1
Address = (00:0490) : Length = 4
OK


ocm-m> memdump
Timer 7:        active          573212

00:0490: 1c bf 08 00
OK
```

A quick inspection of the data shows that this chip stores bytes in little endian order. 0x0008BF1C equals 573,212, which is what is in the printout. Suppose we want this invite process to time out in another minute if it does not get a response. One minute equals 60,000 milliseconds, which is 0x0000EA60. Reversing that into little endian order, we want to load 0x60EA0000.

A quick "memload" command, followed by memdump shows the following.

```
ocm-m> memload:60EA0000
Copied 4 bytes.
OK
ocm-m> memdump
Timer 7:        active          57746

00:0490: 92 e1 00 00
```

The memdump command shows 57.746 seconds, which is two seconds less than a minute. The discrepancy can be explained by the fact that it took about that long to type the memdump command.

5.5 *“interval” command*

The “interval” command works like the memload command except that it takes an interval in milliseconds. From the memdump command.

```
Interval 0(Resp. Timeout):      50
Interval 1(Write Pause):        0
Interval 2(Invite Send Time):   250
Interval 3(Channel Scan Time):  10000
Interval 4(Invite Trans. Timeout):      10000
```

Suppose we want invitations sent out every second instead of every 250 milliseconds. Simply adjust the Invite Send Time interval to 1000 ms.

```
interval:2:1000
```

The next time you send an invite, the transmission LED will blink much more slowly.

5.5.1 “Write Pause” Interval

The “Write Pause” interval is the interval that ONE-NET waits prior to sending a packet after the one_net_write() function. When the write pause interval is greater than 0 and the verbosity level is greater than 3, ONE-NET devices will output the packet they are about to send out the transceiver to the serial cable. Higher verbosity levels will result in more detailed printouts.

```
ocm-m> interval:1:10
OK
ocm-m> verbose level:3
OK
ocm-m> invite:3333-3333
OK
ocm-m>

179134 sending 52 bytes:

00:09F9: 55 55 55 33 b4 bc 6a b4 b4 b4 b4 b4 b4 b4 bc b4
00:0A09: bc c5 c6 54 d3 9a 9c 94 6a 63 b5 c2 69 3a cc 39
00:0A19: 53 d4 b2 54 b4 56 d5 da 65 53 92 ca a9 63 ba 56
00:0A29: 39 cc 93 34
Pause done



179460 sending 52 bytes:

00:09F9: 55 55 55 33 b4 bc 6a b4 b4 b4 b4 b4 b4 b4 bc b4
00:0A09: bc c5 c6 54 d3 9a 9c 94 6a 63 b5 c2 69 3a cc 39
00:0A19: 53 d4 b2 54 b4 56 d5 da 65 53 92 ca a9 63 ba 56
00:0A29: 39 cc 93 34
Pause done
```

5.6 Options to define in config_options.h

DEBUGGING_TOOLS must be defined in config_options.h.

6 Packet Filtering

ONE-NET allows devices to “filter” packets. This is mainly done for debugging and sniffing purposes, particularly when testing multi-hop.

6.1 Filtering By Device

Use the “range test” command line option. When range testing is on, packets will be filtered. When off, range testing is has no effect. If a device is ignored, no messages from it will be responded to. Note that device IDs are the REPEATER Device ID. See examples below…

```
ocm-m> range test:display
Range Testing is off : # of In-Range Devices : 0
OK
ocm-m> range test:on       // all devices are ignored
OK
ocm-m> range test:add:002    // 002 is not ignored
OK
ocm-m> range test:add:004    // 002 and 004 are not ignored
OK
ocm-m> range test:display
Range Testing is on : # of In-Range Devices : 2
Encoded DID : B4B3 -- Raw DID: 0x002
Encoded DID : B4B5 -- Raw DID: 0x004
OK
ocm-m> range test:remove:004    // 004 is now ignored
OK
ocm-m> range test:display
Range Testing is on : # of In-Range Devices : 1
Encoded DID : B4B3 -- Raw DID: 0x002
OK
ocm-m> range test:clear     // all devices are ignored.
OK
ocm-m> range test:off       // filtering is off.  No devices ignored
OK
ocm-m> range test:display
Range Testing is off : # of In-Range Devices : 0
OK
ocm-m> range test:on
```

6.1.1 Options to define

RANGE_TESTING must be defined in config_options.h.
RANGE_TESTING_ARRAY_SIZE must be defined in one_net_port_const.h.

6.2 Filtering By PID

Use the “pid block” command line option. When pid block is on, packets will be filtered. When off, it has has no effect. If a device is ignored, no messages from it will be responded to. The PID values represent the the 6 least significant bits of the 12-bit raw PID. FF for the PID means “wildcard” or “select all”.

```
ocm-m> pid block:off    // turn PID blocking off
OK
ocm-m> pid block:on     // turn PID blocking on
OK
ocm-m> pid block:reject:sa    // accept only non-Stay-Awake packets
OK
ocm-m> pid block:accept:mh    // accept only Multi-Hop packets
OK
ocm-m> pid block:reject:01    // reject Single Data ACKs
OK
ocm-m> pid block:display     // display criteria

PID Blocking is on
PID 0x00:Accept
PID 0x01:Reject
PID 0x02:Accept
PID 0x03:Accept
PID 0x04:Accept
PID 0x05:Accept
PID 0x06:Accept
PID 0x07:Accept
PID 0x08:Accept
PID 0x09:Accept
PID 0x0A:Accept
PID 0x0B:Accept
PID 0x0C:Accept
PID 0x0D:Accept
PID 0x0E:Accept
Stay Awake:Reject
Multi-Hop:Accept
```

```
ocm-m> pid block:all:sa    // remove restriction on Stay-Awake packets.
OK

ocm-m> pid block:display
PID Blocking is on
PID 0x00:Accept
PID 0x01:Reject
PID 0x02:Accept
PID 0x03:Accept
PID 0x04:Accept
PID 0x05:Accept
PID 0x06:Accept
PID 0x07:Accept
PID 0x08:Accept
PID 0x09:Accept
PID 0x0A:Accept
PID 0x0B:Accept
PID 0x0C:Accept
PID 0x0D:Accept
PID 0x0E:Accept
Stay Awake:N/A
Multi-Hop:Accept
OK
ocm-m> pid block:reject:ff   // reject all
OK
ocm-m> pid block:display
PID Blocking is on
PID 0x00:Reject
PID 0x01:Reject
PID 0x02:Reject
PID 0x03:Reject
PID 0x04:Reject
PID 0x05:Reject
PID 0x06:Reject
PID 0x07:Reject
PID 0x08:Reject
PID 0x09:Reject
PID 0x0A:Reject
PID 0x0B:Reject
PID 0x0C:Reject
PID 0x0D:Reject
PID 0x0E:Reject
Stay Awake:N/A
Multi-Hop:Accept

ocm-m> pid block:accept:ff   // Remove all restrictions on the 6 least sig. bits
OK
```

6.2.1 Options to define

PID_BLOCK must be defined in config_options.h.

7 C++ Utilities Programs

Several command line C++ programs have been written to help learn ONE-NET. You can encode, decode, encrypt, decrypt, and parse packets, among other things. In particular, you can sniff packets using an Evaluation Board using a low verbosity level (or a high one). Sniff using a sniffer or use a write pause and copy the output into a file. The utilities are in the directory…

applications/desktop_sniffer/sniff_parse

Execute the Makefile. Most of the utilities have self-explanatory command line usage statements. For the sniffer, a sample file called sniff.txt has been provided. Execute the following command at the command line to see detailed packet breakdowns. The following is the Linux version. For windows, leave off the leading "./".

```
./sniff_parse 255 valid sniff.txt
```

You should see the following on the screen.

```
Timestamp: 1566852 ms

Valid Digits: True, Valid Msg CRC: True, Valid Decoding: True, Valid PID: True, Valid Payload
Decrypt: True, Valid Payload CRC: True, Valid: True

Number of Encoded Bytes: 62, Encoded Bytes:

55 55 55 33 B4 BC B3 B4 B3 B4 B4 B4 B4 B4 BC B4 BC 34 C3 A6 A6 D5 9A 66

C5 AC D5 9C 5A 39 62 36 A3 5A 6C D5 AC A3 94 DA B6 96 D4 AC CC B6 65 D2

C9 D9 C9 C2 34 99 C5 CC B2 9A 35 55 66 C9

Header: 0x55555533, Rptr Did: Encoded (0xB4BC) Raw (0x001) , Msg Crc: Encoded (0xB3) Raw,Shifted
(0x08) Calculated (0x08) CRCs match.

Dst Did: Encoded (0xB4B3) Raw (0x002) , Nid: Encoded (0xB4B4B4B4B4BC) Raw (0x000000001) , Src
Did: Encoded (0xB4BC) Raw (0x001)

Pid: Encoded (0x34C3) Raw (0x040A)  : 12 Raw Bits : 010000001010)

(Raw PID Bits 11 - 8(# XTEA blocks):0100: 4)

(Raw PID Bit 7(MH Bit):0: 0x0: Multi-Hop? No)

(Raw PID Bit 6(SA Bit):0: 0x0: Stay-Awake? No)

(Raw PID Bits 5 - 0(Packet Type):001010: 0x0A: Packet Type -- ONE_NET_RAW_STREAM_DATA)

Multi-Hop : false

Number of Encoded Payload Bytes: 43, Encoded Payload:

A6 A6 D5 9A 66 C5 AC D5 9C 5A 39 62 36 A3 5A 6C D5 AC A3 94 DA B6 96 D4

AC CC B6 65 D2 C9 D9 C9 C2 34 99 C5 CC B2 9A 35 55 66 C9

Number of Key Bytes: 16 : Number of Rounds : 8, Key: 00010203-04050607-08090A0B-0C0D0E0F

Number of Encrypted Payload Bytes(with technique): 33, Encrypted Payload:

79 EF 2B D8 C6 7C A6 35 77 59 A8 F1 F1 96 A8 EC 6B B8 64 91 B4 FC DF 4D

3D 0B 4C 24 7A D4 93 63 40

Number of Decrypted Payload Bytes(with technique): 33, Decrypted Payload:

4E 06 DF C0 00 03 11 00 17 C3 31 00 17 C3 32 00 17 C3 33 00 17 C3 34 00

17 C3 35 00 17 C3 36 00 40

Payload Crc: Payload (0x4E) Calculated (0x4E) CRCs match.  Msg Id: 0x06D

Elapsed Time: 785 ms -- Response Needed: true -- Data :
0017C3310017C3320017C3330017C3340017C3350017C33600
```