PLC Chipset SG Programmers Guide

MKG-xxxxx Ver. 1.0

May, 2013

Confidential and Proprietary - Qualcomm Atheros, Inc.

Qualcomm Atheros, Inc.
1700 Technology Drive
San Jose, CA 95110-1383
U.S.A.

Revision History

Revision	Date	Description
Ver. 1.0	April, 2013	Initial version

Contents

1 Acronyms and Abbreviations 5
2 Architecture Overview 6
2.1 Power line Frequency 6
2.2 SG System architecture 6
3 UART Data transparent TX/RX 7
3.1 UART Data transmit 7
3.2 UART Data receive 8
3.3 Power Meter Address and MAC address Mapping 8
4 Communication protocol 9
4.1 Protocol 9
4.2 Ethernet MME packet header 9
4.3 Get power meter data request (VS_UART_CMD_Req MME) 9
4.4 Get power meter data response (VS_UART_CMD_Cnf) 10
4.5 Get topology request and response (VS_AVLN_TOPO_Req VS_AVLN_TOPO_Cnf) 11
4.6 Network information request (VS_NW_INFO_STATS_Req) 14
4.7 Network information response (VS_NW_INFO_STATS_Cnf) 15
4.8 Get software version request (VS_SW_VER_Req) 16
4.9 Get software version response (VS_SW_VER_Cnf) 17
4.10 Software reset request (VS_RS_DEV_Req) 17
4.11 Software reset response (VS_RS_DEV_Cnf) 17
4.12 MME timeout 18
Appendix : Example for Concentrator raw socket programming 19

Table and Figure

Figure 2-1： QCA broadband PLC Smart Grid system block diagram 6

Figure 3-1： UART data transparently transmit 7

Figure 3-2： UART data transparently receive 8

Figure 4-1： MME packet captured by wireshark 10

Figure 5-2： Raw UART Data sent 10

Table 2-1： Acronyms and Abbreviations 5

Table 4-1： MME Packet Header 9

Table 4-2： VS_UART_CMD_Req 9

Table 4-3： VS_UART_CMD_Cnf 10

Table 4-4： VS_AVLN_TOPO_Req 11

Table 4-5： VS_AVLN_TOPO_Cnf 11

Table 4-6： VAR field for Get tree command 12

Table 4-7： VAR field for Get trace command 12

Table 4-8： STA’s Info 13

Table 4-9： VS_NW_INFO_STATS_Req 14

Table 4-10： VS_NW_INFO_STATS_Cnf 16

Table 4-11： VS_SW_VER_Req 16

Table 4-12： VS_SW_VER_Cnf 17

Table 4-13： VS_RS_DEV_Req 17

Table 4-14： VS_RS_DEV_Cnf 17

1 Acronyms and Abbreviations

Acronyms	Definition
Power Meter	One phase Smart meter and three phase Smart meter
CCO module	CCO module is placed into concentrator. It included 3 QCA6411, 1 5-ports-switch.
STA module	STA module is placed into smart meter. It included 1 QCA7000.
CCO	Central Coordinator
STA	Station
NSTA	Normal STA
PSTA	Proxy STA
HSTA	Hidden STA
AVLN	HomePlug AV Logical Network
Relay	CCO cannot communicate with 1 HSTA, but via PSTA with relay, it can communicate with each other.
n Relay	Multi- relay
MME	MAC Management Message
PB	PHY Block
TEI	Terminal Equipment Identifier
QCA6411	Lynx chip used for CCO
QCA7000	Lynx chip used for STA
DA	Destination Address，Destination MAC address
SA	Source Address，Source MAC address。

Table 1-1：Acronyms and Abbreviations

2 Architecture Overview

2.1 Power line Frequency

To comply with the HomePlug GP specification, the Powerline frequency must be 50Hz±3.0% or 60 Hz±3.5%. Powerline frequency outside of this range will cause the firmware to reboot the device.

2.2 SG System architecture

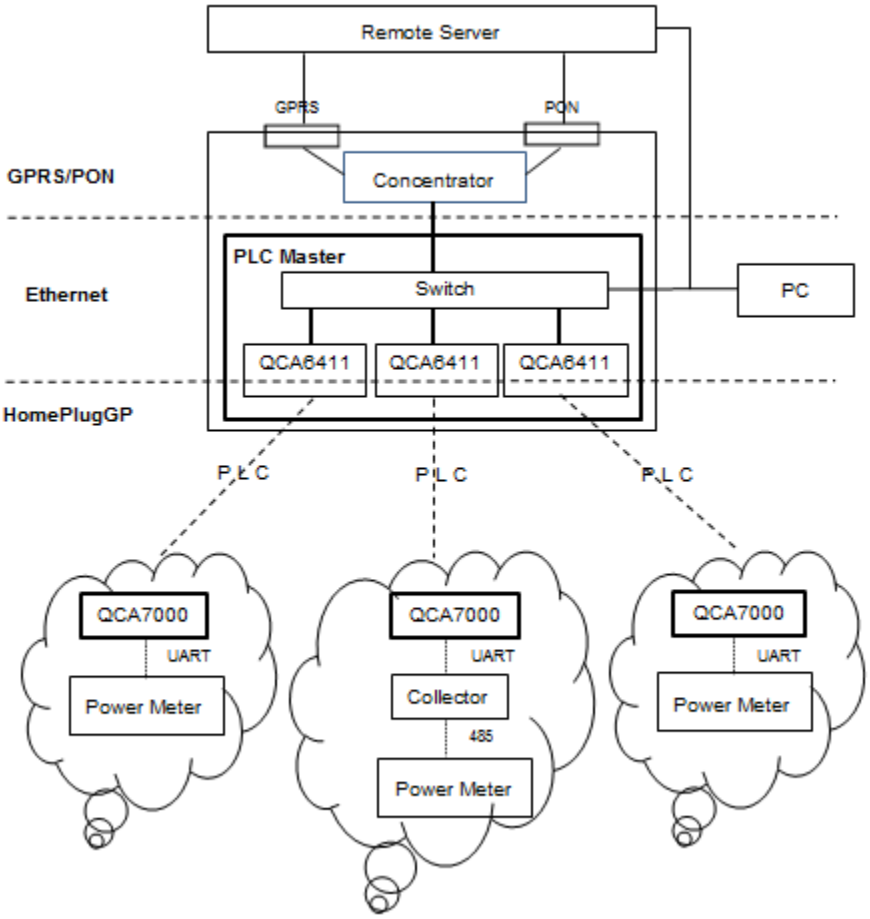

Figure 2-1：QCA broadband PLC Smart Grid system block diagram

CCO module was plugged into concentrator, which had 3 QCA6411 corresponding to 3 phase power line, per QCA6411 use different Network password to indicate 3 different AVLAN. Per VLAN support 253 STAs.

STA module was plugged into power meter, which deployed with 1 QCA7000.STA module fetch power meter data via UART @2400 baud rate. UART baud rate can be configured with AVITAR tools.(@1200 @9600 not recommend to use high baud rate)

3 QCA6411 connect with switch and communicate with concentrator via Ethernet port.

3 UART Data transparent TX/RX

3.1 UART Data transmit

Payload was raw UART data user wants to send to power meter. Maxim UART data length is within 256 Bytes.

This transmit is initiated by Concentrator, so user need to fill in MME packet, refer chapter 4 for sending MME Data.

UART data will transparently send to power meter, so the UART data should be valid and can be recognized by power meter. Here Payload is UART RAW data.

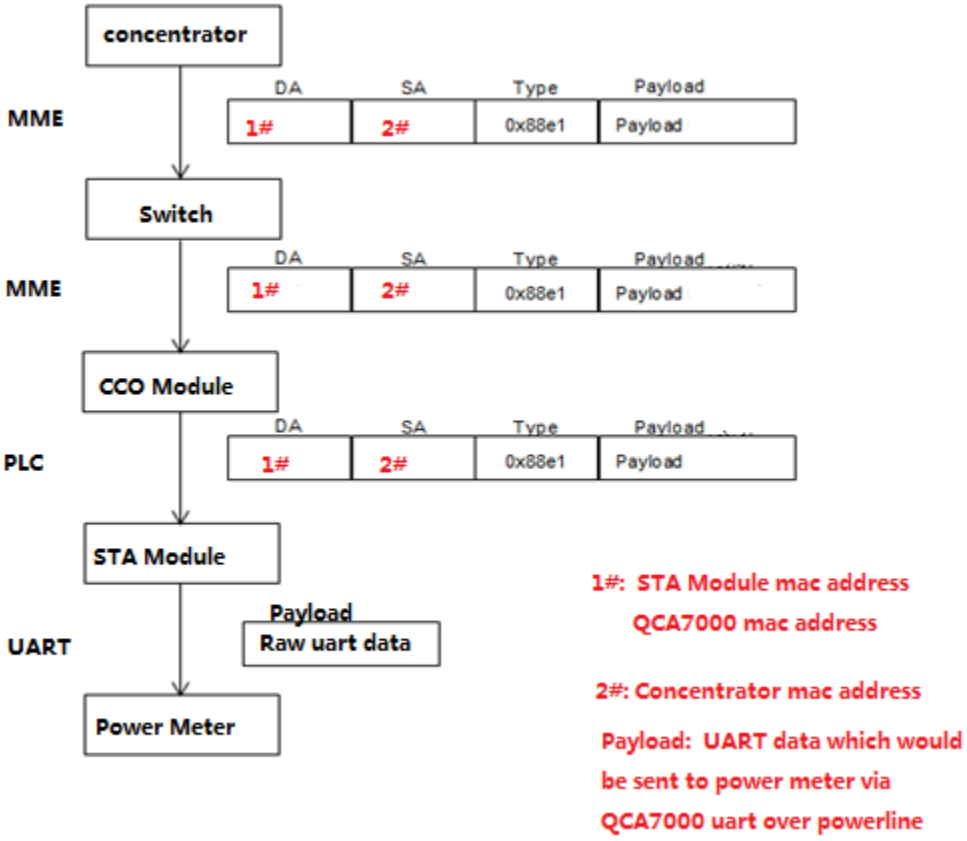

Figure 3-1：UART data transparently transmit

3.2 UART Data receive

Payload was raw UART data which power meter response. Maxim UART data length is within 256 Bytes. UART data will be sent to concentrator transparently. Received MME packet, concentrator can recognize which STA Module response. Further UART data analysis will be handled by concentrator.

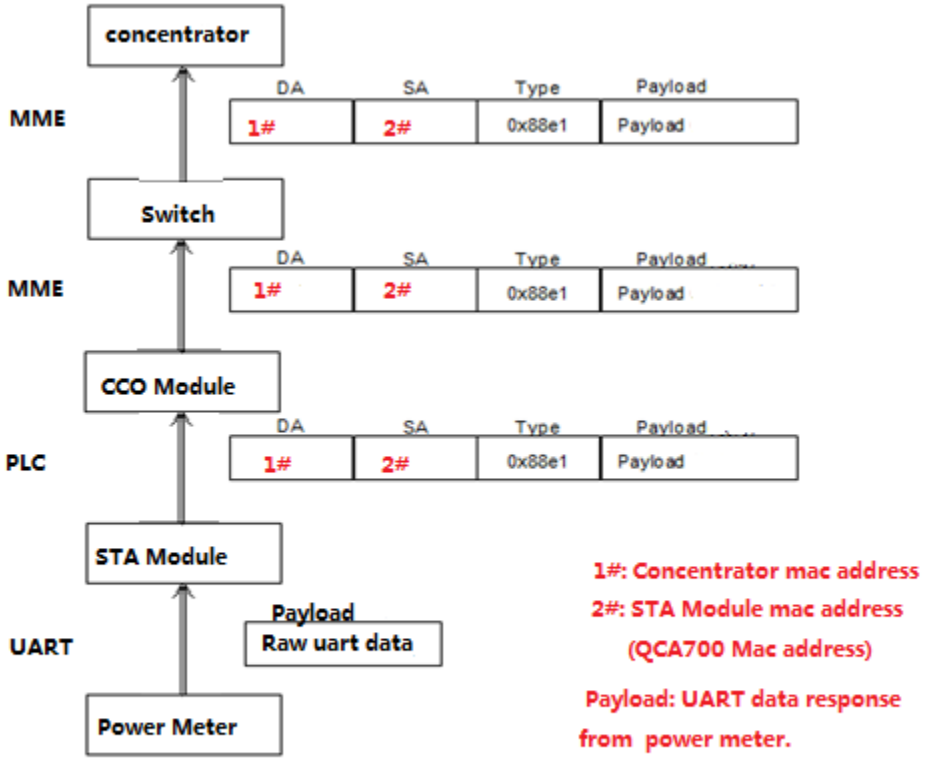

Figure 3-2：UART data transparently receive

3.3 Power Meter Address and MAC address Mapping

Per STA module has own mac address (QCA7000 MAC Address), which was stored in NVM. This mac address can be configured by AVITAR tools. The mapping relationship between QCA7000 module MAC address and Meter ID should be maintained by Concentrator program. Power meter address and mac address mapping can be realized by following steps:

1. Concentrator sends VS_NW_INFO_STATS_Req MME to fetch how many STA modules are online.
2. Concentrator initially sends unicast MME frame to online STA module. Payload (UART Data) should be data to fetch the power meter address or other packet which power meter can recognize.
3. Concentrator analysis the received payload, and extract the power meter ID.
4. Maintain the mapping with STA module MAC address with power meter ID.

4 Communication protocol

4.1 Protocol

- Concentrator communicates with CCO Module via MME.
- CCO communicate with remote server via private protocol, such as 372.1 etc.
- CCO Module communicates with STA Module via HomePlugAV protocol.

  The user just focus on protocol (MME) between Concentrator with CCO module, low level communication is invisible to user.

4.2 Ethernet MME packet header

Offset	Size	Field Name	Description
0x0000	6	ODA	Original Destination Address
0x0006	6	OSA	Original Source Address
0x000C	2	MTYPE	Ethertype (0x88, 0xE1)
0x000E	1	MMV	MME Version (0)
0x000F	2	MMTYPE	MME Type
0x0011	3	OUI	Intellon OUI (0x00, 0xb0, 0x52)

Table 4-1：MME Packet Header

Different Ethernet MME is raw mac frame which shared same packet heard, refer table 4-1.

4.3 Get power meter data request (VS_UART_CMD_Req MME)

Offset	Size	Field Name	Description
0x0000	6	ODA	Original Destination Address
0x0006	6	OSA	Original Source Address
0x000C	2	MTYPE	Ethertype (0x88, 0xE1)
0x000E	1	MMV	MME Version (0)
0x000F	2	MMTYPE	0xA400 (Request)
0x0011	3	OUI	Intellon OUI (0x00, 0xb0, 0x52)
0x0014	2	LENGTH	UART raw data length
0x0016	Variable	DATA	UART raw data

Table 4-2：VS_UART_CMD_Req

- ODA: STA module mac address
- OSA: Concentrator mac address
- MTYPE: 0x88, 0xE1 (constant)
- MMV: 0 (constant)
- MMTYPE: 0xA400(constant)
- OUI: 0x00, 0xB0, 0x52(constant)
- LENGTH: Length of DATA, size in bytes
- DATA: UART command(raw data)

Use wire shark tools to monitor the data you send, the captured data should be like Figure 5-1.

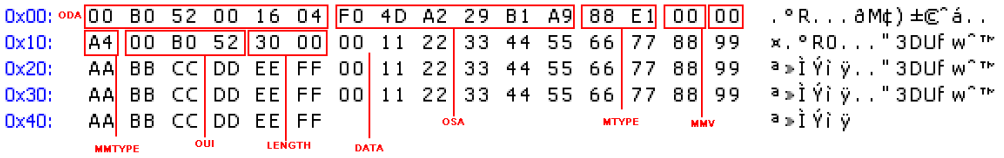

Figure 4-1：MME packet captured by wireshark

```
00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC
DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
```

Figure 4-2：Raw UART Data sent

4.4 Get power meter data response (VS_UART_CMD_Cnf)

Offset	Size	Field Name	Description
0x0000	6	ODA	Original Destination Address
0x0006	6	OSA	Original Source Address
0x000C	2	MTYPE	Ethertype (0x88, 0xE1)
0x000E	1	MMV	MME Version (0)
0x000F	2	MMTYPE	0xA401 (Confirm)
0x0011	3	OUI	Intellon OUI (0x00, 0xb0, 0x52)
0x0014	6	RSVD	Reserved
0x001A	2	LENGTH	UART raw data length
0x001C	Variable	DATA	UART raw data

Table 4-3：VS_UART_CMD_Cnf

- ODA: Concentrator mac address
- OSA: STA module mac address
- MTYPE: 0x88, 0xE1 (constant)
- MMTYPE: 0xA401(constant)
- MMV: 0 (constant)
- OUI: 0x00, 0xB0, 0x52(constant)
- LENGTH: Length of DATA, size in bytes
- DATA: UART response(raw data)

This mac frame is generated by STA module, so concentrator need to further analysis the data.

4.5 Get topology request and response (VS_AVLN_TOPO_Req VS_AVLN_TOPO_Cnf)

Offset	Size	Field Name	Description
0x0000	6	ODA	Original Destination Address
0x0006	6	OSA	Original Source Address
0x000C	2	MTYPE	Ethertype (0x88, 0xE1)
0x000E	1	MMV	MME Version (0)
0x000F	2	MMTYPE	0xA308 (Request)
0x0011	3	OUI	Intellon OUI (0x00, 0xb0, 0x52)
0x0012	1	COMMAND	0x01: get tree 0x02: trace 0x03-0xff: reserved
0x0013	n	VAR	Base on command type

Table 4-4：VS_AVLN_TOPO_Req

Offset	Size	Field Name	Description
0x0000	6	ODA	Original Destination Address
0x0006	6	OSA	Original Source Address
0x000C	2	MTYPE	Ethertype (0x88, 0xE1)
0x000E	1	MMV	MME Version (0)
0x000F	2	MMTYPE	0xA309 (Confirm)
0x0011	3	OUI	Intellon OUI (0x00, 0xb0, 0x52)
0x0014	1	COMMAND	0x01: get tree 0x02: trace 0x03-0xff: reserved
0x0015	1	RESPONSE	LSB 0th bit: show whether the response is successful or not. “1” is fail, “0” is successful. MSB 7th bit: show whether current MME has more MME followed. “1” is yes, “0” means “no” and current is the last MME. Other bits: reserved
0x0016	n	VAR	Base on command type

Table 4-5：VS_AVLN_TOPO_Cnf

This MME is sent by concentrator, it shows all the directly connected STA information under the given STA. Further, it can be used to show the whole topology of one AVLN.

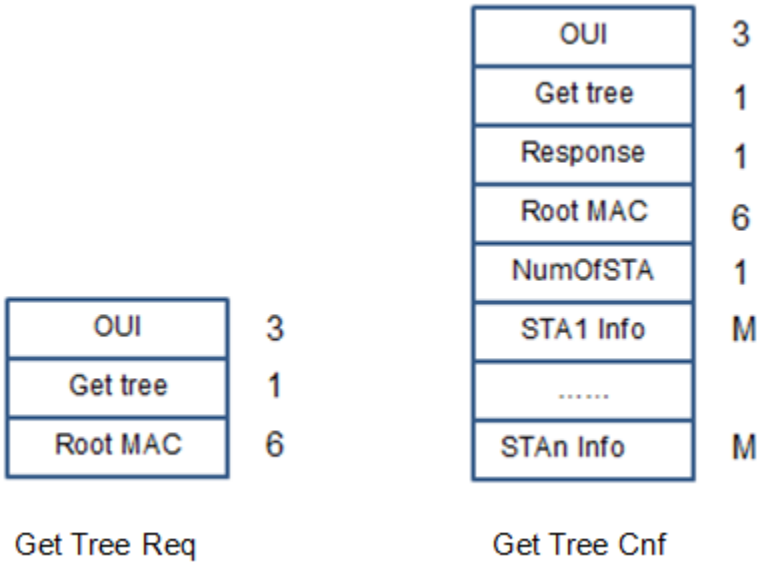

Table 4-6：VAR field for Get tree command

- Response: only for Cnf MME. If the response is successful or not. “1” is fail, “0” is successful
- Root MAC: the MAC of chosen STA to query
- NumOfSTA: the number of STA directly under the chosen STA
- STAn Info: the Nth STA’s Info. Check table 4-8.

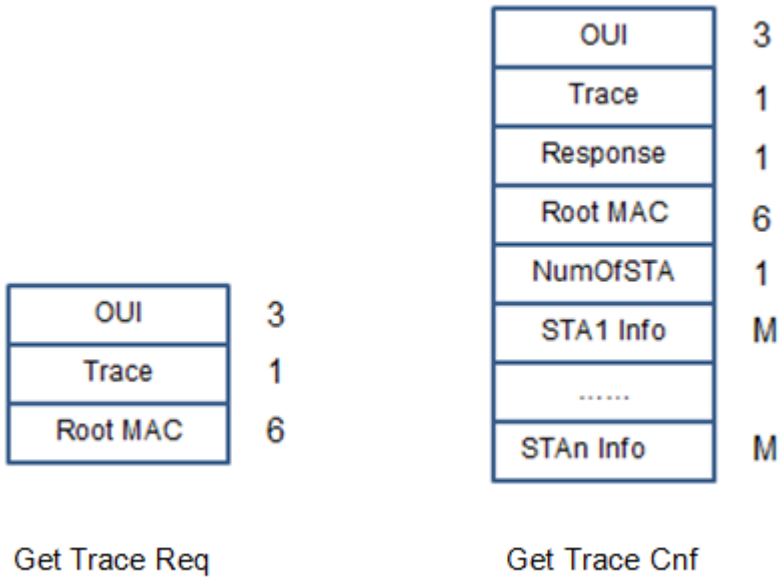

Table 4-7：VAR field for Get trace command

- Root MAC: the chosen STA's MAC
- NumOfSTA: the number of STA in the path from chosen STA to CCO
- STAn Info: the Nth STA's Info. Check table 4-8.

MAC	6
TEI	1
numOfSTA	1
Status	1

Table 4-8：STA's Info

- MAC: STA MAC
- TEI: STA TEI
- NumOfSTA: the number of STA directly under this STA
- Status: the status of this STA. currently "1" means authenticated, "0" is associated

For Trace subcommand, numOfSTA and Status can be ignored

Following could be used as guide to develop application feature at concentrator to get topology related information.

In order to get the topology of CCo, App should follow below steps:

1. Send a get tree command to CCo, "root MAC" is CCo's MAC
2. CCo will return the list of STAs directly connected to it.
3. App check each STA's info, if any STA has at least one STA under it (numOfSTA is non-zero in **Error! Reference source not found.**), App should send a similar get tree command to CCo with "root MAC" set to STA's mac instead. (NOTE: STAs having other STA under it are the non-leaf node in the tree topology, STAs not having other STA under it are the leaf node in the tree topology)
4. Loop 3 until complete sending request MME to CCo for querying all the non-leaf STA (loop all the non-leaf STAs by sending a get tree command)
5. App construct the tree topology based on the information above and show it out.

To get the data path (relay path) for a STA, App should follow below steps:

1. Send a trace command to CCo, "root MAC" is the STA's MAC
2. CCo will return a response with the STA list where all STAs on the path between STA and CCo are listed.
3. App show it out.

4.6 Network information request (VS_NW_INFO_STATS_Req)

Offset	Size	Field Name	Description
0x0000	6	ODA	Original Destination Address
0x0006	6	OSA	Original Source Address
0x000C	2	MTYPE	Ethertype (0x88, 0xE1)
0x000E	1	MMV	MME Version (0)
0x000F	2	MMTYPE	0xA074 (Request)
0x0011	3	OUI	Intellon OUI (0x00, 0xb0, 0x52)
0x0014	1	FIRST_TEI	First TEI requested. If FIRST_TEI is Zero the NUMSTAS field will be valid and up to 96 sets of Station Information will be returned. Otherwise the NUMSTAS field will list the number of sets of Station Information returned and all the existent sets of Station Information will be return at or after the FIRST_TEI (up to 96).

Table 4-9： VS_NW_INFO_STATS_Req

This MME is sent by concentrator to get the network information, which devices registered in the network, and get the MAC address of each device, if some expected devices not exist in the network, concentrator cannot communicate with these devices.

Notes: If FIRST_TEI is set to 0, NUMSTAS in the response stands for total stations in the network, if the number is bigger than 96, users should send the second or maybe third request to get the information(max station number in a network is 253), since the max size of Ethernet frame is 1514 octets.

The first request, FIRST_TEI = 0,

The second request, FIRST_TEI = 98,

The third request, FIRST_TEI = 194,

Station TEI starts from 2, the MAX TEI is 254.

4.7 Network information response (VS_NW_INFO_STATS_Cnf)

Offset	Size	Field Name	Description
0x0000	6	ODA	Original Destination Address
0x0006	6	OSA	Original Source Address
0x000C	2	MTYPE	Ethertype (0x88, 0xE1)
0x000E	1	MMV	MME Version (0)
0x000F	2	MMTYPE	0xA075 (Confirm)
0x0011	3	OUI	Intellon OUI (0x00, 0xb0, 0x52)
0x0014	1	FIRST_TEI	First TEI from Request.
0x0015	1	IN_AVLN	Station is a Member of an AVLN
0x0016	7	NID	Network Identifier The least significant 54 bits of this field contains the NID of the AVLN. The remaining 2 bits are set to 0b00.
0x001D	1	SNID	Short Network Identifier The last significant 4 bits of this field contains the Short Network Identifier. The remaining 4 bits are set to 0x0
0x001E	1	TEI	Terminal Equipment Identifier of the STA in the AVLN
0x001F	1	STATIONROLE	Role of the station in the AVLN 0x00 = STA 0x01 = Proxy Coordinator 0x02 = CCo 0x03 – 0xFF = reserved
0x0020	6	CCO_MACADDR	MAC Address of the CCo of the network.
0x0026	1	ACCESS	Access Network 0x00 = This NID corresponds to an in-home network 0x01 = This NID corresponds to an Access Network 0x02 - 0xFF = reserved
0x0027	1	NumCordNWs	Number of Neighbor Networks that are coordinating with the AVLN 0x00 = none (Un-Coordinated mode) 0x01 = one Coordinating network, and so on
0x0028	1	CCO_TEI	Terminal Equipment Identifier of the CCo in the AVLN
0x0029	1	NUMSTAS	Number of AV STAs in the AVLN = L 0x00 = None, 0x01 = One, and so on.
0x002A	6	DA[0]	MAC Address of the STA – 0
0x0030	1	TEI[0]	TEI of the STA – 0
0x0031	6	1stBDA[0]	MAC Address of the first Node bridged by STA – 0

0x0037	1	AvgPHYDR_TX[0]	Average PHY Data Rate in Mega Bits per second from STA to DA[0]. 0x00 = unreachable/unknown 0x01 = 1 Mbps, and so on
0x0038	1	AvgPHYDR_RX[0]	Average PHY Data Rate in Mega Bits per second from DA[0] to STA. 0x00 = unreachable/unknown 0x01 = 1 Mbps, and so on
* * *			
	6	DA[L-1]	MAC Address of STA – (L-1)
	1	TEI[L-1]	TEI of the STA – (L-1)
	6	1stBDA[L-1]	MAC Address of the first Node bridged by STA – (L-1)
	1	AvgPHYDR_TX[L-1]	Average PHY Data Rate in Mega Bits per second from STA to DA[0]. 0x00 = unreachable/unknown 0x01 = 1 Mbps, and so on
	1	AvgPHYDR_RX[L-1]	Average PHY Data Rate in Mega Bits per second from DA[0] to STA. 0x00 = unreachable/unknown 0x01 = 1 Mbps, and so on

Table 4-10： VS_NW_INFO_STATS_Cnf

4.8 Get software version request (VS_SW_VER_Req)

Offset	Size	Field Name	Description
0x0000	6	ODA	Original Destination Address
0x0006	6	OSA	Original Source Address
0x000C	2	MTYPE	Ethertype (0x88, 0xE1)
0x000E	1	MMV	MME Version (=0)
0x000F	2	MMTYPE	0xA000 (Request)
0x0011	3	OUI	Intellon OUI (0x00, 0xb0, 0x52)

Table 4-11： VS_SW_VER_Req

This MME is sent by concentrator to get software version of devices, the destination could be QCA6411 or QCA7000.

4.9 Get software version response (VS_SW_VER_Cnf)

Offset	Size	Field Name	Description
0x0000	6	ODA	Original Destination Address
0x0006	6	OSA	Original Source Address
0x000C	2	MTYPE	Ethertype (0x88, 0xE1)
0x000E	1	MMV	MME Version (0)
0x000F	2	MMTYPE	0xA001 (Confirm)
0x0011	3	OUI	Intellon OUI (0x00, 0xb0, 0x52)
0x0014	1	MSTATUS	MME Status (0x00=Success)
0x0015	1	MDEVICEID	Device ID
0x0016	1	MVERLENGTH	Length of Version String (including null)
0x0017	128	MVERSION	Version String
0x0097	5	RSVD	

Table 4-12： VS_SW_VER_Cnf

4.10 Software reset request (VS_RS_DEV_Req)

Offset	Size	Field Name	Description
0x0000	6	ODA	Original Destination Address
0x0006	6	OSA	Original Source Address
0x000C	2	MTYPE	Ethertype (0x88, 0xE1)
0x000E	1	MMV	MME Version (0)
0x000F	2	MMTYPE	0xA01C (Request)
0x0011	3	OUI	Intellon OUI (0x00, 0xb0, 0x52)

Table 4-13： VS_RS_DEV_Req

This MME is sent by concentrator to reset target device.

4.11 Software reset response (VS_RS_DEV_Cnf)

Offset	Size	Field Name	Description
0x0000	6	ODA	Original Destination Address
0x0006	6	OSA	Original Source Address
0x000C	2	MTYPE	Ethertype (0x88, 0xE1)
0x000E	1	MMV	MME Version (0)
0x000F	2	MMTYPE	0xA01D (Confirm)
0x0011	3	OUI	Intellon OUI (0x00, 0xb0, 0x52)
0x0014	1	MSTATUS	MME Status (0x00=Success, 0x01 = Failure)

Table 4-14： VS_RS_DEV_Cnf

4.12 MME timeout

The ODA of MME should be existed in the network information. If not, there will be no response. The time out is defined by user; it can be 3 or 5 seconds.

Appendix : Example for Concentrator raw socket programming

Filename: plc.h

```
#ifndef __PLC_H__
#define __PLC_H__
#include <linux/if_ether.h>

typedef unsigned short uint16;
typedef unsigned char byte;
typedef unsigned char uint8;

#define CMD_REQ_HDR_SZ          ( sizeof(struct cmd_req_hdr) )
#define CMD_IND_HDR_SZ          ( sizeof(struct cmd_ind_hdr) )

/* Default settings of cmd_req_hdr */
#define REQ_MTYPE               0x88e1
#define REQ_MMV                 0x00
#define REQ_MMTYPE              0xA400
#define REQ_OUI0                0x00
#define REQ_OUI1                0xB0
#define REQ_OUI2                0x52

/*mme type*/
#define MMTYPE_CMD_IND          0xa402

/*Ether packet type*/
#define ETH_P_HPAV              0x88e1

/*struct type */
struct cmd_req_hdr
{
   struct ethhdr eth_hdr;
   uint8 mmv;
   uint16 mmtype;
   char oui[3];
   uint16 length;
   byte data[0];
} __attribute__((packed));

struct cmd_cnf_hdr
{
   struct ethhdr eth_hdr;
   uint8 mmv;
   uint16 mmtype;
   char oui[3];
   uint16 length;
   byte data[0];
} __attribute__((packed));
```

```
struct cmd_ind_hdr
{
   struct ethhdr eth_hdr;
   uint8 mmv;
   uint16 mmtype;
   char oui[3];
   byte address[ETH_ALEN];
   uint16 length;
   byte data[0];
} __attribute__((packed));


extern void cmd_req_hdr_init( struct cmd_req_hdr* p_req_hdr, const uint8* const dest_mac,
                              const uint8* const src_mac, uint16 len);
#endif
```

Filename: plc.c

```
#include <string.h>
#include "plc.h"

void cmd_req_hdr_init( struct cmd_req_hdr* p_req_hdr, const uint8* const dest_mac, const uint8* const src_mac, uint16 len)
{
   memcpy( p_req_hdr->eth_hdr.h_dest, dest_mac, ETH_ALEN );
   memcpy( p_req_hdr->eth_hdr.h_source, src_mac, ETH_ALEN );
   p_req_hdr->eth_hdr.h_proto = htons( REQ_MTYPE );
   p_req_hdr->mmv = REQ_MMV;
   p_req_hdr->mmtype = REQ_MMTYPE;
   p_req_hdr->oui[0] = REQ_OUI0;
   p_req_hdr->oui[1] = REQ_OUI1;
   p_req_hdr->oui[2] = REQ_OUI2;
   /*p_req_hdr->length = htons( len );*/
   p_req_hdr->length = len;
}
```

Filename: common.h

```
extern int str2hex( const char* str, char byte[], int *psize );
extern int find_char( char str[], int len, char ch );
```

Filename: common.c

```
#include <stddef.h>

int str2hex( const char* str, char byte[], int *psize )
{
   char ch;    int i = 0;
   if( str == NULL )  return -1;
   while ( (ch = *str++) != '\0' )
   {
      if (ch >= 'a' && ch <= 'f')              ch = ch - 'a' + 0x0a;
```

```
      else if (ch >= 'A' && ch <= 'F')          ch = ch - 'A' + 0x0a;
      else if (ch >= '0' && ch <= '9')         ch = ch - '0';
      else  continue;

      if ( i % 2 == 0 ) byte[ i++ / 2 ] = ch << 4;
      else byte[ i++ / 2 ] += ch;
   }

   if ( i % 2 == 0 ) *psize = i / 2;
   else return -1;
   return 0;
}

/* Count char */
int find_char( char str[], int len, char ch )
{
   int i = 0;
   for (i = 0; i < len; i++) if ( str[i] == ch ) return i;
   return -1;
}
```

Filename: eth_plc.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netdb.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/in.h>
#include <netpacket/packet.h>
#include <net/ethernet.h>
#include <net/if.h>
#include <arpa/inet.h>
#include <errno.h>
#include <time.h>
#include "plc.h"
#include "config.h"

#define RCV_BUF_SIZE         1024 * 1                    /* Receive buffer size */
#define SND_BUF_SIZE         1024 * 1                    /* Send buffer size */
#define MAX_PKT_LEN         256
#define RCV_TO                 1

const char zero_mac[ETH_ALEN] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
struct eth_pkt
{
            struct ether_header eth_head;
            char eth_data[0];
}__attribute__ ((__packed__));
```

```
/* This is a list of interface name prefixes which are `bad' in the sense
 * that they don't refer to interfaces of external type on which we are
 * likely to want to listen. We also compare candidate interfaces to lo. */
static char *bad_interface_names[] =
{
        "lo:",
        "lo",
        "stf",                  /* pseudo-device 6to4 tunnel interface */
        "gif",                  /* psuedo-device generic tunnel interface */
        "dummy",
        "vmnet",
        NULL                    /* last entry must be NULL */
};


/* Receive buffer size */
static int rcv_buf_szie = RCV_BUF_SIZE;
static char rcv_buff[RCV_BUF_SIZE] = { 0 };

/*Send buffer size*/
static int snd_buf_szie = SND_BUF_SIZE;
static char snd_buff[SND_BUF_SIZE] = { 0 };

/* Interface name */
static char if_name[10] = "eth0";

/* Host MAC */
static char host_mac[ETH_ALEN] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
static char dest_mac[ETH_ALEN] = { 0xff, 0xff, 0xff, 0xff, 0xff, 0xff };

static void reverse_char_array( char eth_mac[], int len )
{
        int i;
        char tmp;

        for (i = 0; i < len / 2; i++)
        {
                tmp = eth_mac[i];
                eth_mac[i] = eth_mac[len - 1 - i];
                eth_mac[len - 1 - i] = tmp;
        }
}

static void eth_show_mac(const int type, const char eth_mac[])
{
   int i = 0;
   if (0 == type) printf("SMAC=[");
   else printf("DMAC=[");
   for(i = 0; i < ETH_ALEN - 1; i++) printf("%02x:", *((unsigned char *)&(eth_mac[i])));
   printf("%02x] ", *((unsigned char *)&(eth_mac[i])));
}
```

```
/* Set interface as promiscuous mode */
static int set_promisc( const char *ifname, int fd, int flags)
{
   int ret = -1;
   struct ifreq st_ifr;
   /* Get interface flags */
   strcpy( st_ifr.ifr_name, ifname);
   ret = ioctl(fd, SIOCGIFFLAGS, &st_ifr );
   if (ret < 0)
   {
      perror("[Error]Get Interface Flags");
      return -1;
   }

   if (flags == 0)
   {
            /* reset promiscuous mode */
            st_ifr.ifr_flags &= ~IFF_PROMISC;
   }
   /* Set promiscuous mode */
   st_ifr.ifr_flags |= IFF_PROMISC;
   ret = ioctl(fd, SIOCSIFFLAGS, &st_ifr);

   if (ret < 0)
   {
      perror("[Error]Set Interface Flags");
      return -1;
   }
   return 0;
}

int plc_init_socket( void )
{
   int ret = -1;
   int fd = -1;
   struct ifreq st_ifr;
   struct sockaddr_ll st_local = {0};
   struct timeval rcv_timeout = {0, 10000};                    /* 100 minisecond  */
   /* Create a socket */
   fd = socket( PF_PACKET, SOCK_RAW, htons( ETH_P_ALL ));
   if (fd < 0)
   {
      perror("[Error]Initinate L2 raw socket");
      return -1;
   }
    /* Set if name */
   set_ifname( if_name );
   /* Set the interface as promiscuous mode */
   set_promisc( if_name, fd, 1 );
   /* Set the timeout value of receiving, the default value is 0, which means will never time out */
   ret = setsockopt( fd, SOL_SOCKET, SO_RCVTIMEO, (char*)&rcv_timeout, sizeof(struct timeval) );
```

```
   if (ret < 0)
   {
      perror("[Error]Set socket receive timeout");
      close(fd);
      return -1;
   }
   /* Get index of interface */
  strcpy( st_ifr.ifr_name, if_name );
  ret = ioctl(fd, SIOCGIFINDEX, &st_ifr);
 if (ret < 0)
 {
            perror("[Error]Ioctl operation");
            close(fd);
            return -1;
 }

 /* Bind interface */
 st_local.sll_family = PF_PACKET;
 st_local.sll_ifindex = st_ifr.ifr_ifindex;
/*st_local.sll_protocol = htons( ETH_P_ALL );*/
 st_local.sll_protocol = htons( ETH_P_HPAV );

 ret = bind(fd, (struct sockaddr *)&st_local, sizeof(st_local) );
 if (ret < 0)
 {
            perror("[Error]Bind the interface");
            close(fd);
            return -1;
 }

 if ( memcmp( host_mac, zero_mac, ETH_ALEN ) == 0)
 {
            /* Get MAC address */
            ret = ioctl( fd, SIOCGIFHWADDR, &st_ifr );
            if (ret < 0)
            {
                        perror("[Error]Get the MAC address");
                        close(fd);
                        return -1;
            }
            memcpy( host_mac, st_ifr.ifr_hwaddr.sa_data, sizeof(host_mac) );
 }
  return fd;
}

int plc_exit_socket( fd )
{
   return close( fd );
}
```

```
int eth_recv( const int fd,  char** pp_packet)
{
        int pkt_len = -1;

        /* Receive frame */
        memset( rcv_buff, 0, RCV_BUF_SIZE );
        pkt_len = recvfrom( fd, rcv_buff, rcv_buf_szie, 0, NULL, NULL );
        if (pkt_len < 0)
        {
                return -1;
        }

        *pp_packet = rcv_buff;

        #ifdef DB_RCV_ETH_RAW
           int i;
           printf( "\nReceive raw ether frame:\n");
           for (i = 0; i < pkt_len; i++)
           {
              printf( "%02X ", (unsigned char)(*pp_packet)[i] );
           }
           printf( "\n" );
        #endif

        return pkt_len;
}

int eth_plc_recv( const int fd, char* const plc_packet )
{
   int i;
   int pkt_len = -1;
   struct cmd_ind_hdr* p_packet = NULL;

   i = 0;
   /*while (i++ <= RCV_RETRY_CNT)*/
   time_t start_time;
   time_t current_time;

   start_time = time( (time_t*)NULL );
   if (start_time < 0)
   {
      return -1;
   }

   do
   {
      current_time = time( (time_t*)NULL );
      if (current_time < 0)
      {
         return -1;
      }
```

```
      pkt_len = eth_recv( fd, (char**)&p_packet );
      if (pkt_len < 0)
      {
         continue;
      }

      if ( memcmp( dest_mac, p_packet->address, ETH_ALEN ) == 0 && p_packet->mmtype == MMTYPE_CMD_IND )
      {
         if (pkt_len > MAX_PKT_LEN)
         {
            pkt_len = MAX_PKT_LEN;
         }

         pkt_len -= CMD_IND_HDR_SZ;
         memcpy( plc_packet, p_packet->data, pkt_len );
         return pkt_len;
      }
   }
   while (current_time - start_time < RCV_TO);

   return -1;
}

int eth_send( const int fd, const char* const packet, const int len )
{
            int ret = -1;
            int pkt_len = -1;
            int plc_addr_pos = 0;
            struct eth_pkt* eth_packet;
            /*char dest_mac[ETH_ALEN];*/

            if (packet == NULL) return -1;

            pkt_len = len;
            eth_packet = (struct eth_pkt*)malloc( pkt_len );

            if (eth_packet == NULL) return -1;

            memcpy( eth_packet, packet, len );
            /* Send packet */
            pkt_len = sendto( fd, eth_packet, pkt_len, 0, NULL, 0 );
            if (pkt_len < 0)
            {
                        free( eth_packet );
                        return -1;
            }
            free( eth_packet );
            return pkt_len;
}
```

```
int eth_plc_send( const int fd, const char* const plc_packet, const int len )
{
        int ret = -1;
        int pkt_len = -1;
        int plc_addr_pos = 0;
        struct cmd_req_hdr* req_header;

        if (plc_packet == NULL) return -1;

        pkt_len = len + CMD_REQ_HDR_SZ;
        req_header = (struct cmd_req_hdr*)malloc( pkt_len );

        if (req_header == NULL) return -1;

        /* Find the address starting position */
        while (plc_packet[ plc_addr_pos++ ] != 0x68)
        {
                if (plc_addr_pos >= len)
                {
                        free( req_header );
                        return -1;
                }
        }

        memcpy( dest_mac, (char*)plc_packet + plc_addr_pos, ETH_ALEN );
        /*reverse_char_array( dest_mac, ETH_ALEN );*/

        cmd_req_hdr_init( req_header, dest_mac, host_mac, len );
        memcpy( req_header->data, plc_packet, len );

        #ifdef DB_SND_ETH_RAW
          int i;
          printf( "\nSend raw ether frame:\n");
          for (i = 0; i < pkt_len; i++)
          {
            printf( "%02X ", (unsigned char)(((char*)req_header)[i])) ;
          }
          printf( "\n" );
        #endif

        /* Send packet */
        pkt_len = sendto( fd, req_header, pkt_len, 0, NULL, 0 );
        if (pkt_len < 0)
        {
                free( req_header );
                return -1;
        }

        free( req_header );
        return pkt_len;
}
```

```
static int is_bad_interface_name(char *interface_name)
{
	char **p;

	for (p = bad_interface_names; *p; ++p)
		if (strncmp(interface_name, *p, strlen(*p)) == 0) return 1;
	return 0;
}

/* This finds the first interface which is up and is not the loopback
 * interface or one of the interface types listed in bad_interface_names. */
int set_ifname( char interface_name[] )
{
	struct if_nameindex * nameindex;
	int i;

	nameindex = if_nameindex();
	if(nameindex == NULL)
	{
		return -1;
	}
	/* Try to find a interface named if_name */
	i = 0;
	while(nameindex[i].if_index != 0)
	{
		if (strcmp(nameindex[i].if_name, "lo") != 0 && !is_bad_interface_name(nameindex[i].if_name))
		{
			if (strcmp( interface_name, nameindex[i].if_name ) == 0)
			{
				if_freenameindex(nameindex);
				return 0;
			}
		}
		i++;
	}
	/* If no interface named if_name is found, copy the first interface name found to if_name */
	i = 0;
	while(nameindex[i].if_index != 0)
	{
		if (strcmp(nameindex[i].if_name, "lo") != 0 && !is_bad_interface_name(nameindex[i].if_name))
		{
			strcpy( interface_name, nameindex[i].if_name );
			if_freenameindex(nameindex);
			return 0;
		}
		i++;
	}
	return -1;
}
```

```
void plc_set_ifname( char interface_name[] )
{
        strcpy( if_name, interface_name );
}

void plc_set_host_mac( char mac_addr[] )
{
        int i;

        for (i = 0; i < ETH_ALEN; i++)
        {
           host_mac[i] = mac_addr[i];
        }
}
```

Filename: config.h

```
#ifdef DEBUG
  #define DB_SND_ETH_RAW
  #define DB_RCV_ETH_RAW
  #define DB_TEMP
#endif
```

Filename: main.c

```
#include <stdio.h>
#include <string.h>
#include "common.h"

#define COM_RETRY_CNT           3
/*#define RCV_CNT                       32*/

const static char host_mac_addr[6] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };

int main( int argc, char* argv[] )
{
        char recv_to_flag = 0;
        char input_flag = 1;
        /*char send_buf[32] = { 0 };*/
        /*char input_buf[100];*/
        char send_buf[1024] = { 0 };
        char input_buf[1024];
        int com_len;
        int str_len;

        char const send_com[32] = { 0xfe, 0xfe, 0xfe, 0xfe, 0x68, 0x32, 0x00, 0x00, 0x00,
                                    0x00, 0x00, 0x68, 0x13, 0x00, 0x15, 0x16 };
        char receive_buf[300];
        int rcv_len = 0;
        int fd;
        int i;
        char *p;
```

```
plc_set_ifname( "eth0" );
plc_set_host_mac( host_mac_addr );
fd = plc_init_socket();
printf( "\n\n" );
printf("/********QCA7000 Remote meter module test********/");
printf( "\n" );
printf("FE FE FE 68 32 00 00 00 00 00 68 11 04 33 34 34 35 E7 16");
printf( "\n" );
printf("FE FE FE 68 32 00 00 00 00 00 68 11 04 33 34 35 35 E8 16");
while (1)
{

        printf( "\n\n" );
        printf("-----------------------------------------------------------------");

            while( 1 )
            {
               printf( "\n" );
               printf("Input: ");
               gets( input_buf );

              if (str2hex( input_buf, send_buf, &com_len ))
              {
                 printf( "\nInput error!\n");
              }
              else
              {
                 break;
              }
}

 if (eth_plc_send( fd, send_buf, com_len ) < 0 )
 {
       printf( "\neth_send error\n" );
       continue;
  }
        /* Retry COM_RETRY_CNT times */
        for (i = 0; i < COM_RETRY_CNT; i++)
        {
                rcv_len = eth_plc_recv( fd, receive_buf );

                 if (rcv_len > 0)
                 {
                                str_len = find_char( receive_buf, rcv_len,  0x16 );
                                if (str_len > 0) break;
                                else   i--;
                  }
        }

        if (i == COM_RETRY_CNT) printf( "\nReceive time out!" );
        else
        {
```

```
                        printf( "\nReceive %d data: ", str_len + 1 );
                        for (i = 0; i < str_len + 1; i++) printf( "%02X ", (unsigned char)receive_buf[i] );
                    }
            }
            plc_exit_socket( fd );
}
```

Filename: makefile

```
TARGET = eth_plc_7000
OBJS = main.c common.c eth_plc.c plc.c

#CC = /opt/nxp/gcc-4.3.2-glibc-2.7/bin/arm-vfp-linux-gnu-gcc
CC = gcc

all: $(TARGET)

$(TARGET): $(OBJS)
        @$(CC) -I. -o $@ $(OBJS)
        @echo "Compile completed!"
        @#cp $(TARGET) ~/nfs_dir/for_lpc3250/test

clean:
        @rm -vf $(TARGET) *.o *~
```