Ambiq Micro Data Transfer Profile (AMDTP) Example Project

Date	Revision History	Reviser
2017-09-14	V0.1 draft created	Mike Li

Service Declaration

The service UUID of Ambiq Micro DTP (Data Transfer Protocol) service is defined as below 00002760-08C2-11E1-9073-0E8AC72E1011.
Note:
Base UUID of Bluetooth SIG is 00000000-0000-1000-8000-00805F9B34FB.
https://www.bluetooth.com/specifications/assigned-numbers/service-discovery

Service Characteristics Definitions

Rx : 00002760-08C2-11E1-9073-0E8AC72E0011
Tx : 00002760-08C2-11E1-9073-0E8AC72E0012
ACK/Control : 00002760-08C2-11E1-9073-0E8AC72E0013

Characteristic	Requirements	Mandatory Properties	Security Permissions	Description
Characteristic Rx	M	Write	None	Data from client
Characteristic Rx User Description	O	Read	None	Value read by client
Characteristic Tx	M	Notify	None	Value notification to client
Characteristic Tx Client Characteristic Configuration descriptor	M	Read/Write	None	Value notification configuration
Characteristic ACK	M	Write/Notify	None	ACK/Control to client
Characteristic ACK Client Characteristic Configuration descriptor	M	Read/Write	None	ACK notification configuration

Characteristics

The following characteristics are defined in the AM DTP Service. Only one instance of each characteristic is permitted within this service.

Characteristic Name	Mandatory Properties	Security Permission
Characteristic Rx	Write Command	None
Characteristic Tx	Notify	None
Characteristic ACK	Write Command/Notify	None

Characteristic Descriptors

Characteristic User Description

This characteristic descriptor defines the AM DTP version with read permission property.

Client Characteristic Configuration Descriptor

The notification characteristic will start to notify if the value of the CCCD (Client Characteristic Configuration Descriptor) is set to 0x0001 by client. The send data characteristic will stop notifying if the value of the CCCD is set to 0x0000 by client.

Service Behaviors

1. Either server or client may initiate data transfer.
2. Client enables notification of Tx over its CCCD upon connection establishment.
3. Client enables notification of ACK over its CCCD upon connection establishment.
4. Server to Client transmission (ACK mechanism enabled) :
    a. Server starts data transmission by sending data packet to the client via notification (Characteristic Tx).
    b. Client response with acknowledgement by writing to ACK characteristic.
    c. Upon the ACK, Server behaviors are as following
        i. AMDTP_STATUS_SUCCESS
            Checksum is good. Server sends next packet.
        ii. AMDTP_STATUS_CRC_ERROR
            Checksum is bad. Server resend current packet.
        iii. AMDTP_STATUS_TIMEOUT
            Packet sent timeout. Server resend current packet.

5. Client to Server transmission (ACK mechanism enabled) :
    a. Client starts data transmission by sending data data packet to the server via writing to the RX characteristic (Characteristic Rx).
    b. Server response with acknowledgement by ACK notification (Characteristic ACK).
    c. Upon the ACK, Client behaviors are as following
        i. AMDTP_STATUS_SUCCESS
           Checksum is good. Client sends next packet.
        ii. AMDTP_STATUS_CRC_ERROR
           Checksum is bad. Client resend current packet.
        iii. AMDTP_STATUS_TIMEOUT
           Packet sent timeout. Client resend current packet.

AMDTP Packet Definition

AMDTP Packet Format

Length : 2 bytes (data + checksum)
Header : 2 bytes
Data : 0 ~ 512 bytes
Checksum (CRC32) : 4 bytes (Header and length are excluded, only data part is calculated)

Length	Header	Data	Checksum (CRC32)
2 bytes	2 bytes	0 ~ 512 bytes	4 bytes

<u>AMDTP Header Format</u>

bit15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Pkt type				Serial Number				Enc	Ack	RFU					

Serial Number : Packet serial number
Enc : Encryption enabled
Ack : Ack mechanism enabled

Packet Types

typedef enum eAmdtpPktType
{
    AMDTP_PKT_TYPE_UNKNOWN,
    AMDTP_ PKT_TYPE _DATA,
    AMDTP_ PKT_TYPE _ACK,
    AMDTP_PKT_TYPE_CONTROL,

```
        AMDTP_ PKT_TYPE_MAX
}eAmdtpPktType_t;
```

Data Packet

Prefix : 2 bytes length + 2 bytes header
Data : 0 ~ 512 bytes
Checksum : 4 bytes

ACK Packet

Prefix : 2 bytes length + 2 bytes header
Status : 1 byte
Checksum : 4 bytes

CONTROL Packet

Prefix : 2 bytes length + 2 bytes header
Status : 1 byte
Serial Number : 1 byte
Checksum : 4 bytes

AMDTP Status Code

```
typedef enum
{
        AMDTP_STATUS_SUCCESS,
        AMDTP_STATUS_CRC_ERROR,
        AMDTP_STATUS_INVALID_METADATA_INFO,
        AMDTP_STATUS_INVALID_PKT_LENGTH,
        AMDTP_STATUS_INSUFFICIENT_BUFFER,
        AMDTP_STATUS_UNKNOWN_ERROR,
      AMDTP_STATUS_BUSY,
    AMDTP_STATUS_NOTIFY_DISABLED,
    AMDTP_STATUS_TX_NOT_READY,
    AMDTP_STATUS_RESEND_REPLY,
    AMDTP_STATUS_RECEIVE_CONTINUE,
    AMDTP_STATUS_RECEIVE_DONE,
  AMDTP_STATUS_MAX
}eAmdtpStatus;
```

AMDTP Fragmentation and Reassemble

The maximum transmit unit size in ATT layer can be different from various products which introduced a limitation to the maximum payload size of a notification packet in BLE. In order to overcome above limitation, we have implemented an AMDTP packet fragmentation and reassemble mechanism in AMDTP service. The user can configure the maximum AMDTP packet size for fitting different applications. When transmitting, an AMDTP packet will be fragmented into maximum link layer MTU size and will be sent from the length field to the CRC filed in an AMDTP packet. The receiver side will check the whole AMDTP packet is received based on the AMDTP packet length information and does a CRC check for AMDTP packet correctness.

AMDTP Data Deliver Reliability

An ACK mechanism is added into AMDTP profile level for the data deliver reliability. Figure 1 AMDTP Packet Transfer Flowchart shows the communications between sender and receiver.

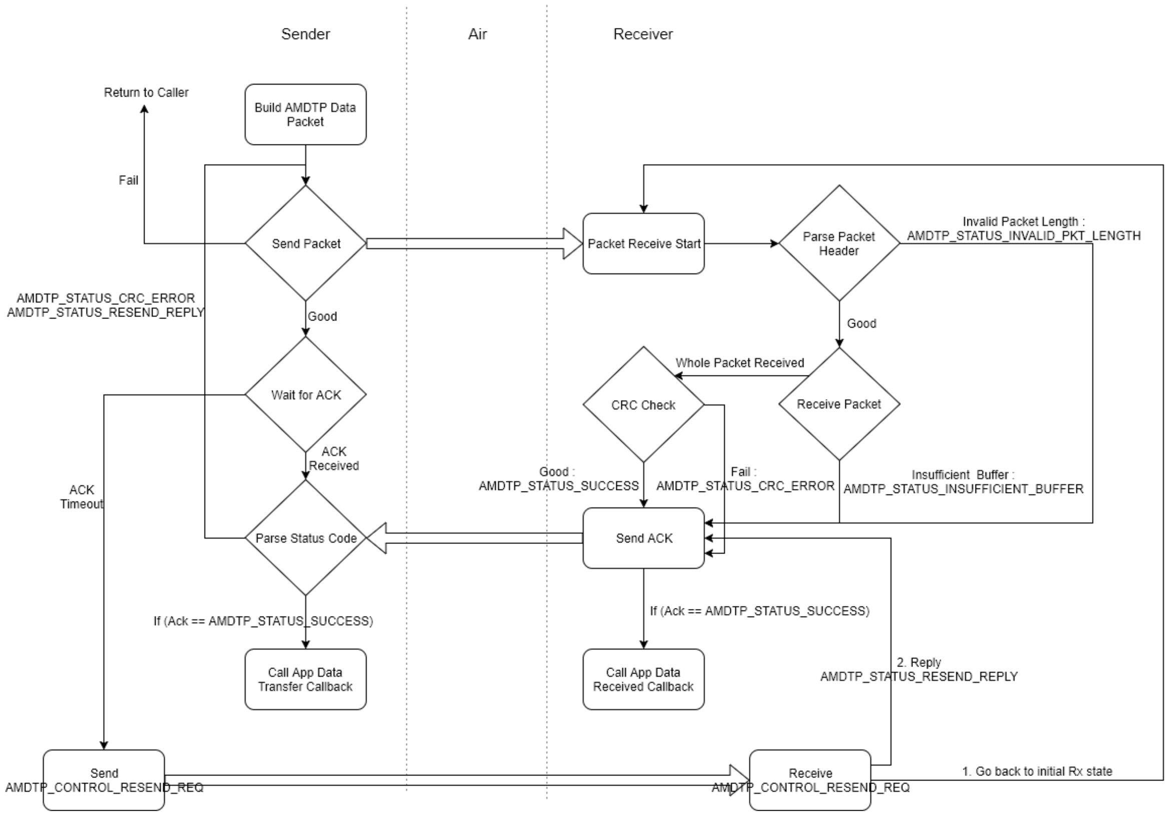

Sender
Air
Receiver
Return to Caller
Build AMDTP Data Packet
Fail
Send Packet
Packet Receive Start
Parse Packet Header
Invalid Packet Length : AMDTP_STATUS_INVALID_PKT_LENGTH
AMDTP_STATUS_CRC_ERROR
AMDTP_STATUS_RESEND_REPLY
Good
Good
Whole Packet Received
Wait for ACK
CRC Check
Receive Packet
ACK Received
ACK Timeout
Good : AMDTP_STATUS_SUCCESS
Fail : AMDTP_STATUS_CRC_ERROR
Insufficient Buffer : AMDTP_STATUS_INSUFFICIENT_BUFFER
Parse Status Code
Send ACK
If (Ack == AMDTP_STATUS_SUCCESS)
If (Ack == AMDTP_STATUS_SUCCESS)
Call App Data Transfer Callback
Call App Data Received Callback
2. Reply AMDTP_STATUS_RESEND_REPLY
Send AMDTP_CONTROL_RESEND_REQ
Receive AMDTP_CONTROL_RESEND_REQ
1. Go back to initial Rx state

FIGURE 1 AMDTP PACKET TRANSFER FLOWCHART

AMDTP Integration with Applications

Below are the procedures to add AMDTP profile into an example that uses Cordio BLE stack.

1. Add below files into project

   amdtp_main.c
   amdtp_common.c
   amdtps_main.c
   svc_amdtp.c

2. Add below paths into "include" folder

   sdk_root/ambiq_ble/apps/amdtps
   sdk_root/ambiq_ble/profiles/amdtpcommon
   sdk_root/ambiq_ble/profiles/amdtps
   sdk_root/ambiq_ble/services

3. In application handler initialization function (e.g. "*AmdtpHandlerInit()*"), call below function to initialize AMDTP server. Two callback functions "*amdtpDtpRecvCback()*" and "*amdtpDtpTransCback()*" need to be created in the project

   *amdtps_init(handlerId, (AmdtpsCfg_t *) &amdtpAmdtpsCfg, amdtpDtpRecvCback, amdtpDtpTransCback);*

4. Add "AMDTPS_TX_CH_CCC_HDL" and "AMDTPS_ACK_CH_CCC_HDL" to CCC set

5. Call function "*amdtps_start()*" or "*amdtps_stop()*" when "AMDTP_AMDTPS_TX_CCC_IDX" value changed

6. Call function "*amdtps_proc_msg()*" in the message process function for below messages

   AMDTP_TIMER_IND
   ATTS_HANDLE_VALUE_CNF
   DM_CONN_OPEN_IND
   DM_CONN_CLOSE_IND
   DM_CONN_UPDATE_IND

7. Call below functions when add the characteristics

   SvcAmdtpsCbackRegister(NULL, amdtps_write_cback);

SvcAmdtpsAddGroup();

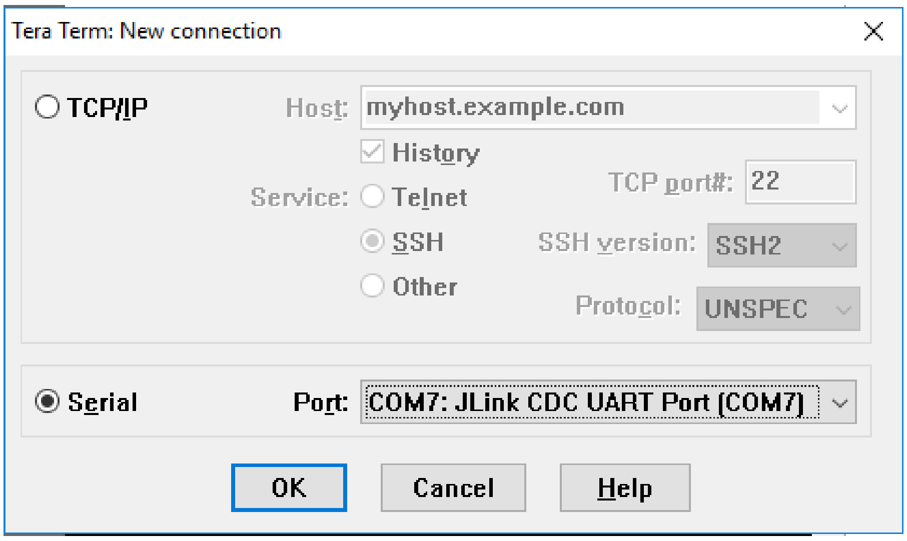
Hands on AMDTP example

AMDTP Server

1. Program AMDTPS project into the development board
2. Reset the board and it will start to advertise automatically
3. Debug output is going through SWO

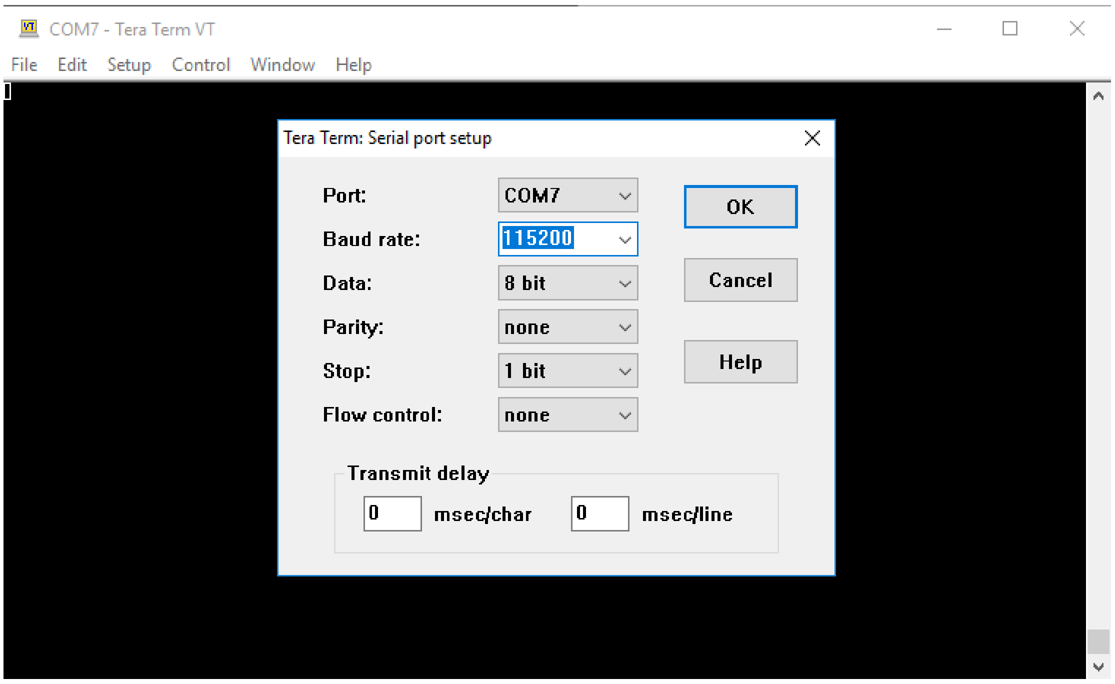
AMDTP Client

1. Program AMDTPC project into the development board
2. Debug output is going through SWO
3. Start a COM terminal tool (e.g. Tera Term) and connects to the board



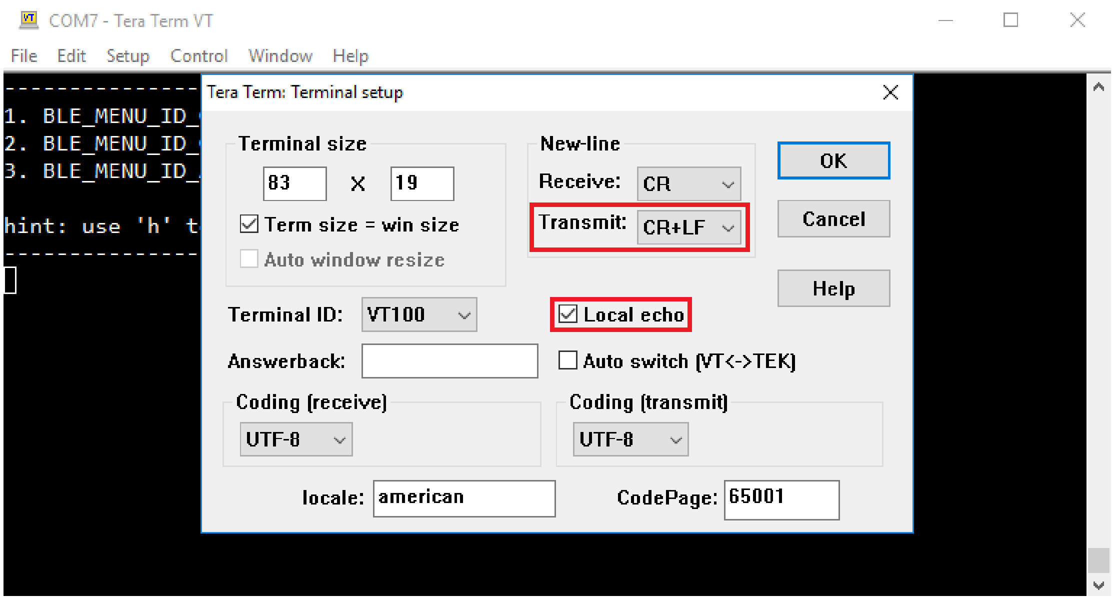
Host: myhost.example.com
TCP port#: 22
SSH version: SSH2
Protocol: UNSPEC
Serial [✓] Port: COM7: JLink CDC UART Port (COM7)

4. In "Setup" → "Serial port", change "Baud rate" to 115200 as below then click "OK"


COM7 - Tera Term VT
File Edit Setup Control Window Help
Tera Term: Serial port setup
OK
Cancel
Help

Port: COM7
Baud rate: 115200
Data: 8 bit
Parity: none
Stop: 1 bit
Flow control: none
Transmit delay
0 msec/char
0 msec/line

5. In "Setup" → "Terminal", change "Transmit" in "New-line" tab to "CR+LF" and enable "Local echo" then click "OK"

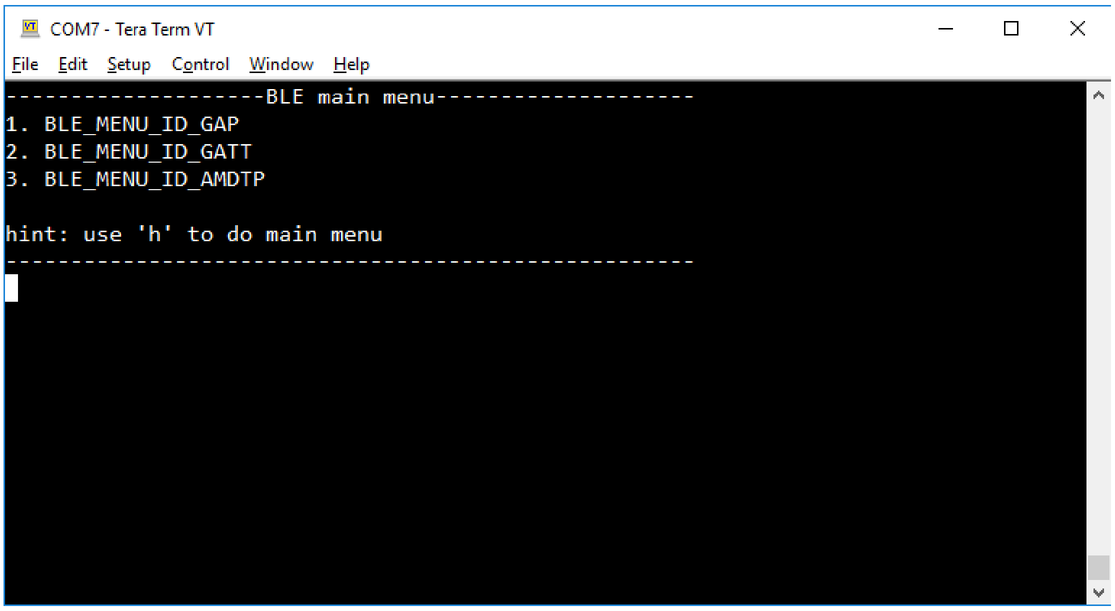

COM7 - Tera Term VT
File Edit Setup Control Window Help
Tera Term: Terminal setup
OK
Cancel
Help

```
---------------
1. BLE_MENU_ID_
2. BLE_MENU_ID_
3. BLE_MENU_ID_

hint: use 'h' t
---------------
```

Terminal size: 83 x 19
☑ Term size = win size
☐ Auto window resize
New-line
Receive: CR
Transmit: CR+LF
Terminal ID: VT100
Answerback:
☑ Local echo
☐ Auto switch (VT<->TEK)
Coding (receive): UTF-8
Coding (transmit): UTF-8
locale: american
CodePage: 65001

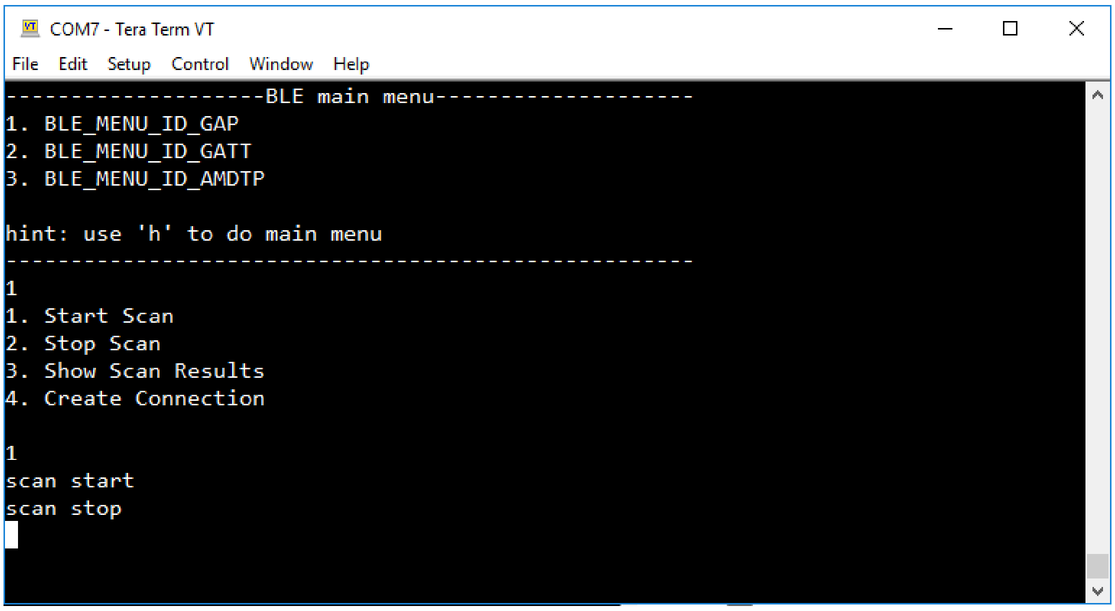
6. After resetting the board, we should be able to see below output in terminal



```
--------------------BLE main menu--------------------
1. BLE_MENU_ID_GAP
2. BLE_MENU_ID_GATT
3. BLE_MENU_ID_AMDTP

hint: use 'h' to do main menu
-----------------------------------------------------
```

7. To create connection with AMDTPS, input 1 and press "Enter" from keyboard to go into "BLE_MENU_ID_GAP". Input 1 again to "Start Scan" and wait for 5 seconds for scan complete (hint: we can observe AMDTP client activities from the J-link SWO output while operating).

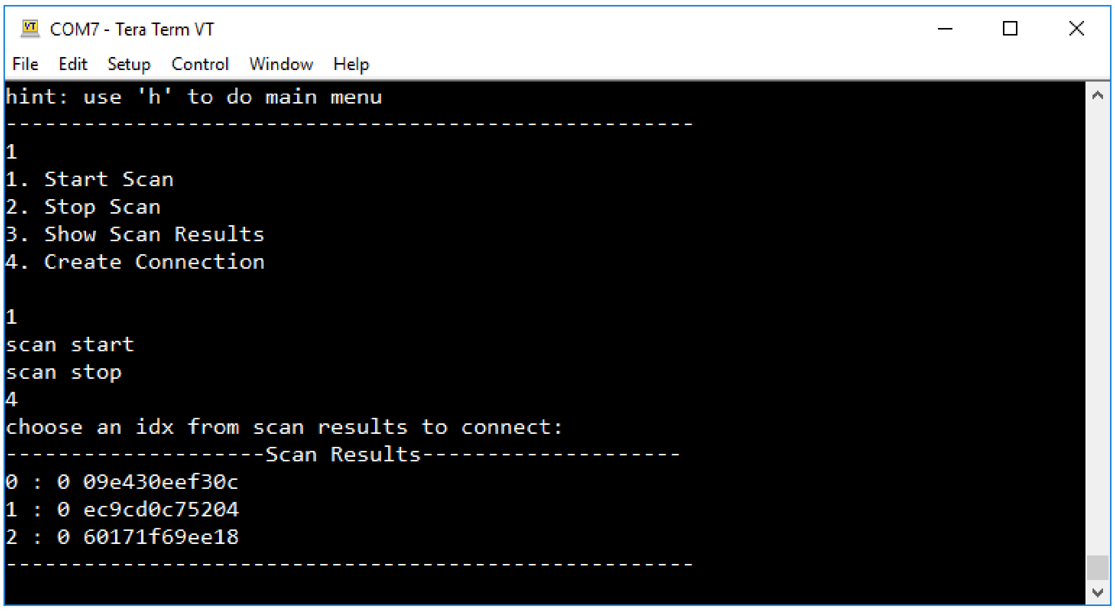


```
--------------------BLE main menu--------------------
1. BLE_MENU_ID_GAP
2. BLE_MENU_ID_GATT
3. BLE_MENU_ID_AMDTP

hint: use 'h' to do main menu
-----------------------------------------------------
1
1. Start Scan
2. Stop Scan
3. Show Scan Results
4. Create Connection

1
scan start
scan stop
```

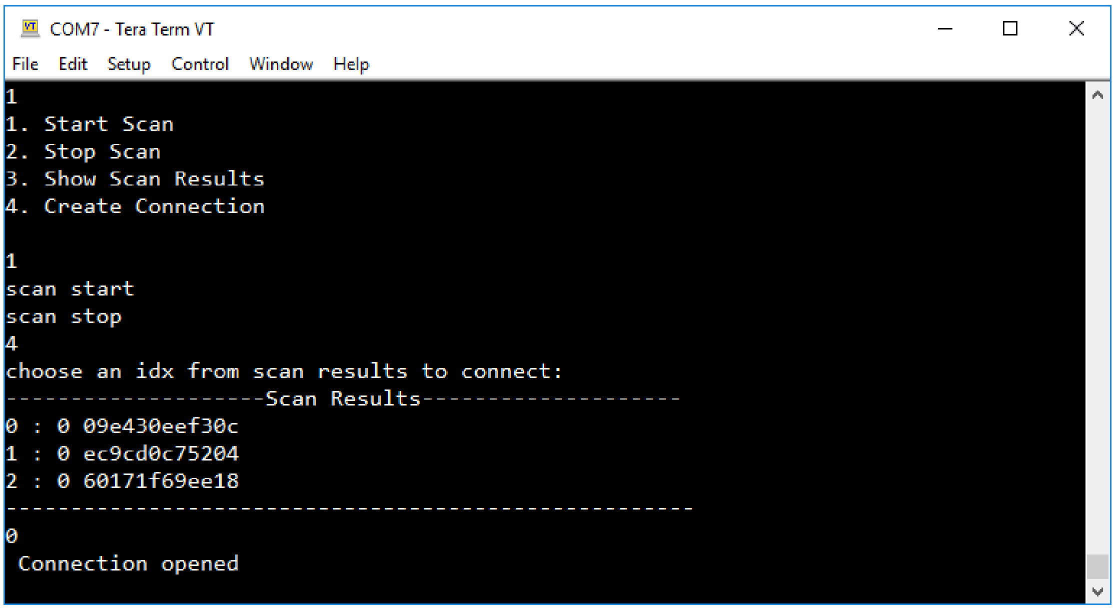
8. After scan complete, input 4 to "Create Connection". Scan results will be popped as below figure. The first number in the list is the index and followed by BD address type and BD address.



```
hint: use 'h' to do main menu
-----------------------------------------------------
1
1. Start Scan
2. Stop Scan
3. Show Scan Results
4. Create Connection

1
scan start
scan stop
4
choose an idx from scan results to connect:
--------------------Scan Results--------------------
0 : 0 09e430eef30c
1 : 0 ec9cd0c75204
2 : 0 60171f69ee18
-----------------------------------------------------
```

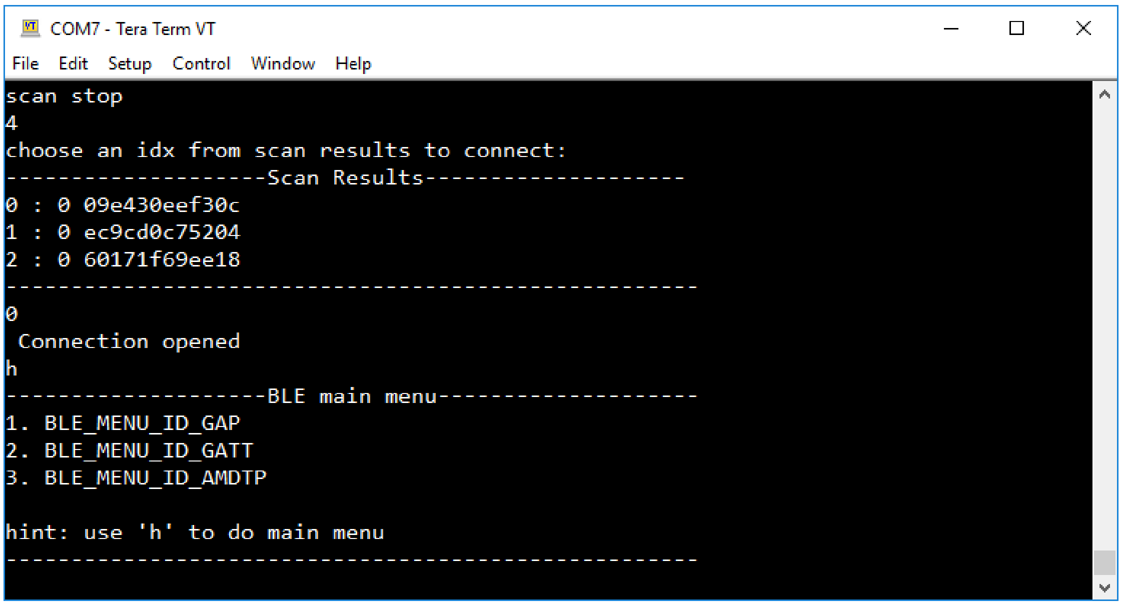
9. Input the target index that we would like to connect to. A "Connection opened" message will show up after connecting to target device.



```
1
1. Start Scan
2. Stop Scan
3. Show Scan Results
4. Create Connection

1
scan start
scan stop
4
choose an idx from scan results to connect:
--------------------Scan Results--------------------
0 : 0 09e430eef30c
1 : 0 ec9cd0c75204
2 : 0 60171f69ee18
-----------------------------------------------------
0
 Connection opened
```

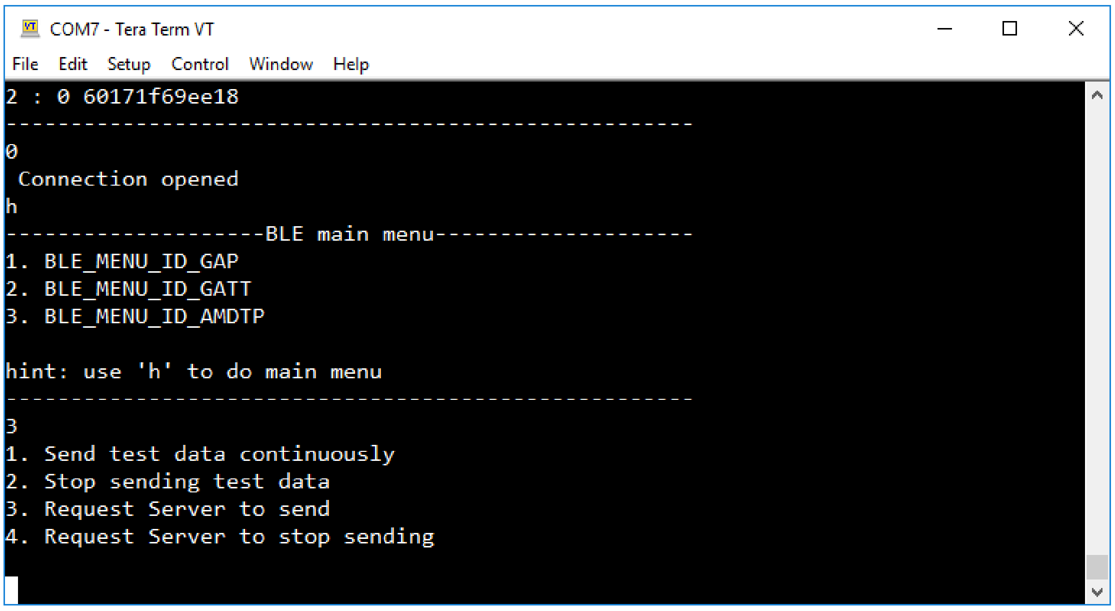
10. Input "h" to go back to root menu



```
scan stop
4
choose an idx from scan results to connect:
--------------------Scan Results--------------------
0 : 0 09e430eef30c
1 : 0 ec9cd0c75204
2 : 0 60171f69ee18
-----------------------------------------------------
0
 Connection opened
h
--------------------BLE main menu--------------------
1. BLE_MENU_ID_GAP
2. BLE_MENU_ID_GATT
3. BLE_MENU_ID_AMDTP

hint: use 'h' to do main menu
-----------------------------------------------------
```

11. Input 3 to go into "BLE_MENU_ID_AMDTP" and it provides 4 commands to interact with AMDTP Server



```
2 : 0 60171f69ee18
----------------------------------------------------
0
 Connection opened
h
--------------------BLE main menu--------------------
1. BLE_MENU_ID_GAP
2. BLE_MENU_ID_GATT
3. BLE_MENU_ID_AMDTP

hint: use 'h' to do main menu
----------------------------------------------------
3
1. Send test data continuously
2. Stop sending test data
3. Request Server to send
4. Request Server to stop sending
```