Getting Started with ‘HostAPI’

HostAPI is a simple protocol to help communication of CV2x and PC through Ethernet. Here is the example for exchanging data between CV2x and PC.

CV2x Side

1. Init socket server by ArmEth_Init()
2. Do connection throughput test by ArmEth_TxRxTest()
3. Send two data to PC by ArmEth_Send()
   - Data0: Size is 1000 bytes, and each byte has content ‘0x12’
   - Data1: Size is 1000 bytes, and each byte has content ‘0x34’
4. Receive data from PC
   (1) Check the number and size of receiving data by ArmEth_GetSize()
   (2) Receive data by ArmEth_Recv()

server.c

```
#include "ArmErrCode.h"
#include "ArmLog.h"
#include "ArmEth.h"
#include "cvapi_ambacv_flexidag.h"

#define ARM_LOG_SERVER "SERVER"

int main(int argc, char **argv)
{
  UINT32 Rval = ARM_OK;
  UINT32 Ch;

  /* 1. Init Ethernet and get channel id */
  Rval = ArmEth_Init(&Ch);

  /* 2. Throughput test */
  {
    flexidag_memblk_t TestBuf = {0};
```

```
    UINT32 Size = 10 * 1024 * 1024;

    if (ARM_OK == Rval) {
        Rval = AmbaCV_UtilityCmaMemAlloc(Size, 1, &TestBuf);
    }

    if (ARM_OK == Rval) {
        Rval = ArmEth_TxRxTest(Ch, TestBuf.pBuffer, Size);
        (void) AmbaCV_UtilityCmaMemFree(&TestBuf);
    }
}

/* 3. TX test */
if (ARM_OK == Rval) {
    ARM_ETH_SIZE_INFO_s SizeInfo = {0};
    ARM_ETH_DATA_INFO_s DataInfo = {0};
    UINT32 i;
    flexidag_memblk_t TxBuf[ARM_ETH_MAX_IO];

    /* Fill size info */
    SizeInfo.Num = 2;
    for (i = 0; i < SizeInfo.Num; i++) {
        SizeInfo.Size[i] = 1000;
        Rval = AmbaCV_UtilityCmaMemAlloc(SizeInfo.Size[i], 1, &TxBuf[i]);
        if (ARM_OK != Rval) {
            ArmLog_ERR(ARM_LOG_SERVER, "## AmbaCV_UtilityCmaMemAlloc fail", 0U, 0U);
            break;
        }
    }

    if (ARM_OK == Rval) {
        (void) memset((void *)TxBuf[0].pBuffer, 0x12, SizeInfo.Size[0]);
        (void) memset((void *)TxBuf[1].pBuffer, 0x34, SizeInfo.Size[1]);
        /* Fill data info */
        //DataInfo.SeqNum, will assign it in API
        DataInfo.TimeStamp = 1234567;
        DataInfo.pBuf[0] = TxBuf[0].pBuffer;
        DataInfo.pBuf[1] = TxBuf[1].pBuffer;
```

```
        /* TX */
        Rval = ArmEth_Send(Ch, &SizeInfo, &DataInfo);
        /* free mem */
        for (i = 0; i < SizeInfo.Num; i++) {
            (void) AmbaCV_UtilityCmaMemFree(&TxBuf[i]);
        }
    }
}

/* 4. RX test */
if (ARM_OK == Rval) {
    ARM_ETH_SIZE_INFO_s SizeInfo = {0};
    ARM_ETH_DATA_INFO_s DataInfo = {0};
    UINT32 i;
    flexidag_memblk_t RxBuf[ARM_ETH_MAX_IO];
    flexidag_memblk_t RxBufGolden[ARM_ETH_MAX_IO];

    /* Get size info */
    Rval = ArmEth_GetSize(Ch, &SizeInfo);
    if (ARM_OK == Rval) {
        for (i = 0; i < SizeInfo.Num; i++) {
            printf("[RX]Size-%u: %u\n", i, SizeInfo.Size[i]);
            Rval = AmbaCV_UtilityCmaMemAlloc(SizeInfo.Size[i], 1, &RxBuf[i]);
            if (ARM_OK != Rval) {
                ArmLog_ERR(ARM_LOG_SERVER, "## AmbaCV_UtilityCmaMemAlloc fail", 0U, 0U);
                break;
            }
            /* Assign RX buffer address */
            DataInfo.pBuf[i] = RxBuf[i].pBuffer;

            /* Golden buffer */
            Rval = AmbaCV_UtilityCmaMemAlloc(SizeInfo.Size[i], 1, &RxBufGolden[i]);
            if (ARM_OK != Rval) {
                ArmLog_ERR(ARM_LOG_SERVER, "## AmbaCV_UtilityCmaMemAlloc fail", 0U, 0U);
                break;
            }
        }
    }
```

```
    /* Set Golden data */
    (void) memset((void *)RxBufGolden[0].pBuffer, 0x56, SizeInfo.Size[0]);
    (void) memset((void *)RxBufGolden[1].pBuffer, 0x78, SizeInfo.Size[1]);

    if (ARM_OK == Rval) {
      /* RX */
      Rval = ArmEth_Recv(Ch, &SizeInfo, &DataInfo);
      /* Compare data */
      for (i = 0; i < SizeInfo.Num; i++) {
        INT32 Ret;
        Ret = memcmp((void *)DataInfo.pBuf[i], (void *)RxBufGolden[i].pBuffer, SizeInfo.Size[i]);
        if (Ret != 0) {
          printf("[RX][ERR] data-%u is incorrect!!\n", i);
        } else {
          printf("[RX][OK] data-%u is correct!!\n", i);
        }
      }
      printf("[RX] seq %u timestamp %lu\n", DataInfo.SeqNum, DataInfo.TimeStamp);
      /* free mem */
      for (i = 0; i < SizeInfo.Num; i++) {
        (void) AmbaCV_UtilityCmaMemFree(&RxBuf[i]);
        (void) AmbaCV_UtilityCmaMemFree(&RxBufGolden[i]);
      }
    }
  }

  return 0;
}
```

PC Side

1. Connect to socket server by AmbaEth_Init()
2. Do connection throughput test by AmbaEth_TxRxTest()
3. Receive data from CV2x
   (1) Check the number and size of receiving data by AmbaEth_GetSize()
   (2) Receive data by AmbaEth_Recv()
4. Send two data to CV2x by AmbaEth_Send()

- Data0: Size is 1000 bytes, and each byte has content ‘0x56’
- Data1: Size is 1000 bytes, and each byte has content ‘0x78’

client.c

```
#include <stdio.h>                    /* for printf() */
#include <stdlib.h>                   /* for malloc() */
#include <string.h>                   /* for memset(), memcmp() */
#include "AmbaEth_api.h"              /* Amba Ethernet API */

#define CLIENT_STRING_ERR             "[CLIENT][ERR]"
#define CLIENT_STRING                 "[CLIENT]"

int32_t main(int32_t argc, char **argv)
{
    int32_t ret = 0;
    uint32_t ch;

    do {
        if (2 != argc) {
            printf("%s %s input parameter(%d) is not 1\n", CLIENT_STRING_ERR, __func__, argc - 1);
            break;
        }

        /* 1. Connect to CV2x */
        ret = AmbaEth_Init(argv[1], &ch);
        if (ret == AMBA_ETH_ERR_NG) {
            break;
        }

        /* 2. Throughput test */
        {
            void *test_buf;
            uint32_t size = 10 * 1024 * 1024;

            test_buf = malloc(size);
            ret = AmbaEth_TxRxTest(ch, (char *)test_buf, size);
            free(test_buf);
            if (ret == AMBA_ETH_ERR_NG) {
```

```
        break;
    }
}

/* 3. Uint test CV2x->PC */
{
    AMBA_ETH_SIZE_INFO_s size_info = {0};
    AMBA_ETH_DATA_INFO_s data_info = {0};
    uint32_t i;
    void *p_rxBuf[AMBA_ETH_MAX_IO];
    void *p_rxBufGolden[AMBA_ETH_MAX_IO];

    /* Get size info */
    ret = AmbaEth_GetSize(ch, &size_info);
    if (ret == AMBA_ETH_ERR_NG) {
        break;
    }

    for (i = 0; i < size_info.num; i++) {
        printf("%s[RX]Size-%u: %u\n", CLIENT_STRING, i, size_info.size[i]);
        p_rxBuf[i] = malloc(size_info.size[i]);

        /* Assign RX buffer address */
        data_info.pBuf[i] = (char *)p_rxBuf[i];

        /* Golden buffer */
        p_rxBufGolden[i] = malloc(size_info.size[i]);
    }

    /* Set Golden data */
    (void) memset(p_rxBufGolden[0], 0x12, size_info.size[0]);
    (void) memset(p_rxBufGolden[1], 0x34, size_info.size[1]);

    /* RX */
    ret = AmbaEth_Recv(ch, &size_info, &data_info);
    if (ret == AMBA_ETH_ERR_NG) {
        break;
    }
```

```
        /* Compare data */
        for (i = 0; i < size_info.num; i++) {
            ret = memcmp((void *)data_info.pBuf[i], p_rxBufGolden[i], size_info.size[i]);
            if (ret != 0) {
                printf("%s[RX] data-%u is incorrect!!\n", CLIENT_STRING_ERR,i);
                break; // Terimated for()
            } else {
                printf("%s[RX] data-%u is correct!!\n", CLIENT_STRING, i);
            }
        }
        printf("%s[RX] seq %u timestamp %lu\n", CLIENT_STRING, data_info.seqNum,
data_info.timeStamp);
        /* free mem */
        for (i = 0; i < size_info.num; i++) {
            free(p_rxBuf[i]);
            free(p_rxBufGolden[i]);
        }

        if (ret == AMBA_ETH_ERR_NG) {
            break;// Terimated main()
        }
    }

    /* 4. Unit test PC->CV2x */
    {
        AMBA_ETH_SIZE_INFO_s size_info = {0};
        AMBA_ETH_DATA_INFO_s data_info = {0};
        uint32_t i;
        void *p_txBuf[AMBA_ETH_MAX_IO];

        /* Fill size info */
        size_info.num = 2;
        for (i = 0; i < size_info.num; i++) {
            size_info.size[i] = 1000;
            p_txBuf[i] = malloc(size_info.size[i]);

        }
```

```
        (void) memset(p_txBuf[0], 0x56, size_info.size[0]);
        (void) memset(p_txBuf[1], 0x78, size_info.size[1]);
        /* Fill data info */
        //DataInfo.SeqNum, will assign it in API
        data_info.timeStamp = 7654321;
        data_info.pBuf[0] = p_txBuf[0];
        data_info.pBuf[1] = p_txBuf[1];
        /* TX */
        ret = AmbaEth_Send(ch, &size_info, &data_info);
        if (ret == AMBA_ETH_ERR_NG) {
            break;
        }
        /* free mem */
        for (i = 0; i < size_info.num; i++) {
            free(p_txBuf[i]);
        }
    }

  } while(0);

  return ret;
}
```

Execution

CV2x Side

```
modprobe ambarella_eth
ifconfig eth0 169.254.197.80
flexidag_schdr -s
cd /tmp/SD0/
./server
```

PC Side

```
./client 169.254.197.80
```

Output

CV2x Side

```
[ArmUtil_ETH|DBG] XportInit()
[ArmUtil_Mutex|DBG] Success to create mutex (ETH Lock)
[ArmUtil_MsgQ|DBG] Success to create MsgQueue (/ETH Queue)
[ArmUtil_Task|DBG] Success to create Task (TCP Server)
[ArmUtil_ETH|DBG] ServerEntry() port 8888
[ArmUtil_ETH|DBG] Listening on port 8888 ...

[ArmUtil_Mutex|DBG] Success to create mutex (XPCB_Mut)
[ArmUtil_Mutex|DBG] Success to create mutex (XPCB_Mut)
[ArmUtil_Mutex|DBG] Success to create mutex (XPCB_Mut)
[ArmUtil_ETH|DBG] ArmEth_Init() channel 0 connected!!
[ArmUtil_ETH|DBG]
[ArmUtil_ETH|DBG] TX: Connection okay.
[ArmUtil_ETH|DBG] RX: Connection okay.
[ArmUtil_ETH|DBG]
[RX]Size-0: 1000
[RX]Size-1: 1000
[RX][OK] data-0 is correct!!
[RX][OK] data-1 is correct!!
[RX] seq 0 timestamp 7654321
```

PC Side

```
Connect to RX ...
Connect to TX ...
Channel 0 is connected.

RX: Connection okay.
TX: Connection okay.

[CLIENT][RX]Size-0: 1000
[CLIENT][RX]Size-1: 1000
[CLIENT][RX] data-0 is correct!!
[CLIENT][RX] data-1 is correct!!
[CLIENT][RX] seq 0 timestamp 1234567
```