Quick Start Guide:
Flexidag Ethernet Data flow

1 Overview

This article is to describe that how to set-up Flexidag Ethernet Data flow by using HostAPI. HostAPI is a protocol help user to exchange data between PC and CV2x. One of its usage is to feed image from PC to CV2x, run flexidag iterations on CV2x and send output results back to PC side. We provide two programs, *ut2*(PC) and *cvflow_comm*(CV2x), to demo this flow.

Figure 1 shows the protocol stack of Flexidag Ethernet data flow.

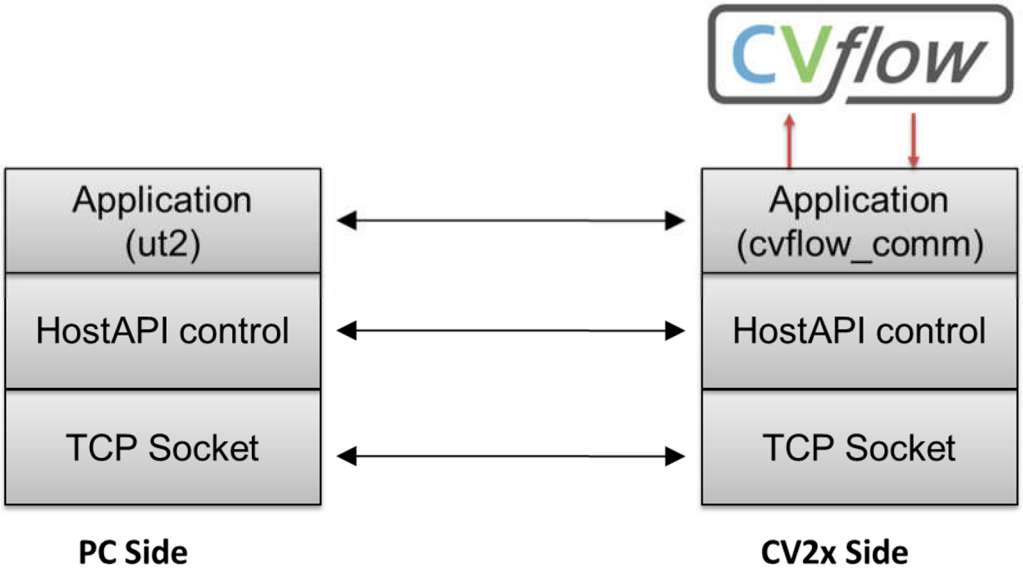

Figure 1

2 Example

Figure 2 shows the usage of HostAPI. User needs to prepare a Linux PC and one piece of CV2x development board. Connect these two devices by Ethernet cable. And then follow the steps below to set up and run HostAPI. The following takes the flexidag, MobileNet+SSD, as an example.

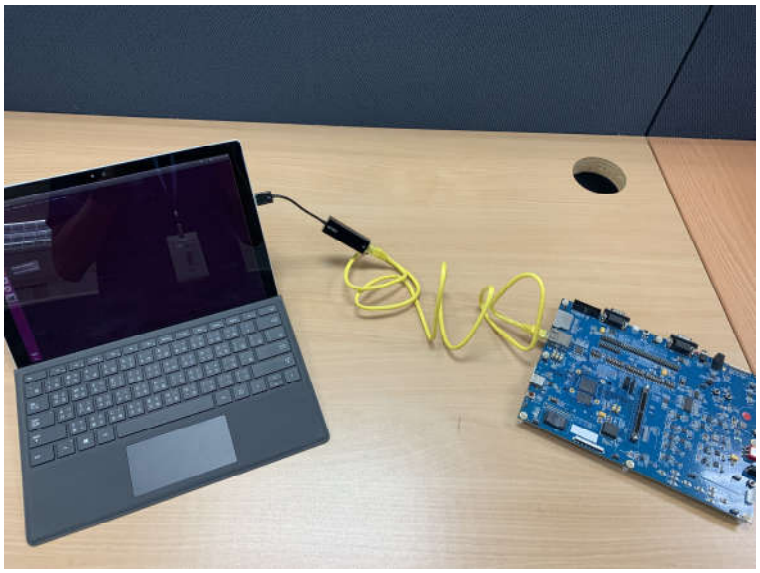

Figure 2

2.1 STEP1: Check Ethernet Configuration

Assume IP address is '169.254.197.83' on PC side. Board side is '169.254.197.80'.

PC side:

Issue the command to setup MTU,

sudo ifconfig eno1 mtu 4000

'eno1' is network interface of PC side. And then type

ifconfig

Check if the 'addr' and 'MTU' has been set to the appropriate value.

```
ambuser@ambuser-OptiPlex-9020:~$ ifconfig
eno1      Link encap:Ethernet  HWaddr f8:b1:56:c5:61:d8
          inet addr:169.254.197.83  Bcast:169.254.255.255  Mask:255.255.0.0
          inet6 addr: fe80::5b9d:9bee:e480:657f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:4000  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:42 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:5660 (5.6 KB)
          Interrupt:20 Memory:f7c00000-f7c20000
```

Board side:

Key the following commands on Linux console (UART0). To enable Ethernet PHY,

modprobe ambarella_eth

To config Board side IP address,

ifconfig eth0 169.254.197.80

```
/ # modprobe ambarella_eth
[    5.857696] pps_core: LinuxPPS API ver. 1 registered
[    5.862661] pps_core: Software ver. 5.3.6 - Copyright 2005-2007 Rodolfo Giometti <giometti@linux.it>
ifconfig eth0 169.254.197.80[    5.876978] PTP clock support registered
[    5.943675] libphy: Ambarella MDIO Bus: probed
[    5.954840] ambarella-eth e000e000.ethernet: Enhance Ethernet PHY[0]: 0x001cc915!
[    5.963083] ambarella-eth e000e000.ethernet: ambarella ptp clock register.
[    5.969966] ambarella-eth e000e000.ethernet: MAC Address[da:50:1e:d1:10:01].
/ # ifconfig eth0 169.254.197.80
[    6.713554] net eth0: adv: sym 0, asym: 0
/ #
```

PC side:

Check network connection,

ping 169.254.197.80

```
ambuser@ambuser-OptiPlex-9020:~$ ping 169.254.197.80
PING 169.254.197.80 (169.254.197.80) 56(84) bytes of data.
64 bytes from 169.254.197.80: icmp_seq=1 ttl=64 time=0.271 ms
64 bytes from 169.254.197.80: icmp_seq=2 ttl=64 time=0.151 ms
64 bytes from 169.254.197.80: icmp_seq=3 ttl=64 time=0.147 ms
64 bytes from 169.254.197.80: icmp_seq=4 ttl=64 time=0.140 ms
64 bytes from 169.254.197.80: icmp_seq=5 ttl=64 time=0.139 ms
64 bytes from 169.254.197.80: icmp_seq=6 ttl=64 time=0.146 ms
64 bytes from 169.254.197.80: icmp_seq=7 ttl=64 time=0.121 ms
```

2.2 STEP2: Test TCP Throughput

Board Side:

To run in server mode,

iperf -s -i 1 -w 1024k

```
/ # iperf -s -i 1 -w 1024k
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size:  448 KByte (WARNING: requested 1000 KByte)
------------------------------------------------------------
```

PC Side:

Run in client mode and connect to Board,

iperf -c 169.254.197.80 -i 1 -w 1024k

```
ambuser@ambuser-OptiPlex-9020:~$ iperf -c 169.254.197.80 -i 1 -w 1024k
------------------------------------------------------------
Client connecting to 169.254.197.80, TCP port 5001
TCP window size:  416 KByte (WARNING: requested 1.00 MByte)
------------------------------------------------------------
[  3] local 169.254.197.83 port 34554 connected with 169.254.197.80 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec   112 MBytes   935 Mbits/sec
[  3]  1.0- 2.0 sec   111 MBytes   934 Mbits/sec
[  3]  2.0- 3.0 sec   111 MBytes   934 Mbits/sec
[  3]  3.0- 4.0 sec   111 MBytes   934 Mbits/sec
[  3]  4.0- 5.0 sec   111 MBytes   934 Mbits/sec
[  3]  5.0- 6.0 sec   110 MBytes   926 Mbits/sec
[  3]  6.0- 7.0 sec   111 MBytes   934 Mbits/sec
[  3]  7.0- 8.0 sec   111 MBytes   934 Mbits/sec
[  3]  8.0- 9.0 sec   112 MBytes   935 Mbits/sec
```

Then you will see the TX throughput is about 800 ~ 900Mbps. After this test, press ‘Ctrl+c’ on Board side to cancel test.

2.3 STEP3: Build Flexidag

We take MobileNet-SSD as an example. Please refer to the section ‘How to Generate Flexibin with Customized Linux Application’ on CnnTestbedUserGuide V2.0.0 or newer. Once you have built the flexidag; copy it, e.g. the flexidag folder is “01000_mnet_ssd_adas_cf_flexidag_raw”, to SD card.

2.4 STEP4: Run Flexidag Ethernet Data flow

Board side:

Insert SD card and types the following command to run *cvflow_comm*.

To start up flexidag scheduler,

flexidag_schdr -s

```
/ # flexidag_schdr -s
  ATT PA region : [0x37000000 -- 0x80000000]
  ATT CA region : [0x37000000 -- 0x80000000]
ambacv visorc binary layout:
        VP:   [0x37400000 ++ 0x0008da34]

[   18.105450] Set bin path /lib/firmware/
[   18.112538]  VP:   [0x37400000 ++ 0x0008da34]
  ATT PA region : [0x37000000 -- 0x80000000]
  ATT CA region : [[   18.116971] ambacv: schdr cleanup finished
0x37000000 -- 0x80000000]
Ambacv kernel module version: 5.0
Host[   18.127557] ioctl_cavalry_enable() : Cavalry enabled for 0 slots
[   18.138449] Waiting for 1 sub-schedulers

[   18.143700] Explicitly clearing VISORC area from cache [0x37600000 - 0x3acdffff]
[   18.156019] start visorc
[   18.193603] ================ Log Buffer Info ================
[   18.199169]    VP:CVTASK: @ [0x3ac97f00--0x3acdfeb7] (294840)
[   18.204902]    VP: SCHED: @ [0x3ac4ff00--0x3ac97eb7] (294840)
[   18.210634]    VP:  PERF: @ [0x3ac07f00--0x3ac4feff] (294912)
[   18.216365]  ARM0:CVTASK: @ [0x3aaf7500--0x3ab3f4ff] (294912)
[   18.222096]  ARM0: SCHED: @ [0x3aaee500--0x3aaf74ff] (36864)
[   18.227740]  ARM0:  PERF: @ [0x3aace500--0x3aaee4ff] (131072)
[   18.233486] IDSP vp-queue: control-block=3ABF7E80 base=3ABF7F00, size=128
[   18.240266] IDSP async-queue base=3ABF6E80
[   18.246053] [AMBACV]: softreset 10
############################################################
                  memory partition
############################################################
       ORC cached: [0x37A73D00, 0x37AC3D00]
     ORC uncached: [0x37AC3D00, 0x37AD3D00]
              ARM: [0x3AA0BA80, 0x3AA1BA80]
     Sysflow name: [0x3AACBC00++0x00000000]
   cvtable header: [0x3AACBB80++0x00000000]
Receive startup_log update message
Snapshot current log buffer to startup_flush.log
```

To change working directory to flexidag you built,

cd /tmp/SD0/01000_mnet_ssd_adas_cf_flexidag_raw

To run APP,

./cvflow_comm

```
/tmp/SD0/01000_mnet_ssd_adas_cf_flexidag_raw # ./cvflow_comm
  ATT PA region : [0x16000000 -- 0x80000000]
  ATT CA region : [[  141.537116] [CVFLOW] : Restoring kernel sysconfig (ffffff8008c46000 -> 000000000
0791530)
0x16000000 -- 0x80000000]
schdrmsg_rx_entry : self-created
[ArmUtil_Mutex|DBG] Success to create mutex (ArmMemCtrlMut_00)
[ArmUtil_MemPool|DBG] ArmMemPool_Create(): PoolId = 0, IsCached = 1
[ArmUtil_MemPool|DBG]                      StartAddr=0x22000000, Size=96MB
[Shell|DBG] Please Enter Shell Command:
```

To load flexidag binary file into memory,

create 0 single flexibin/flexibin0.bin

```
[Shell|DBG] Please Enter Shell Command:
create 0 single flexibin/flexibin0.bin
[CtCvAlgoWrapper|DBG] LoadFlexibin: flexibin/flexibin0.bin
[ArmUtil_Mutex|DBG] Success to create mutex (ArmMemCtrlMut_01)
[ArmUtil_MemPool|DBG] ArmMemPool_Create(): PoolId = 1, IsCached = 1
[ArmUtil_MemPool|DBG]                      StartAddr=0x22000000, Size=64MB
####################################################
   flexidag[  0] sysflow name table
####################################################
      0: uuid=0         INPUT, 0, MNET_SSD_ADAS_FLEX_RAW_AG_INP_0, $##FLEXIDAG
      1: uuid=2         MNET_SSD_ADAS_FLEX_RAW_AG_SUPER, mnet_ssd_adas_flex_raw_ag, mnet_ssd_adas_flex
_raw_ag_orc, mnet_ssd_adas_flex_raw_ag
      2: uuid=4         OUTPUT, 0, MNET_SSD_ADAS_FLEX_RAW_AG_OUT_0, $##FLEXIDAG
      3: uuid=6         OUTPUT, 1, MNET_SSD_ADAS_FLEX_RAW_AG_OUT_1, $##FLEXIDAG
```

To start Ethernet data flow,

eth_start

```
[Shell|DBG] Please Enter Shell Command:
eth_start
[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)
```

PC side:

Copy the source code from below location to your Linux PC.

$SDK/ambalink/pkg/ambacv/arm_framework/app/cvflow_comm/tools/PC_APP

And then type the following commands to build client program.

Change working directory to ‘host’ folder of PC_APP,

cd $PATH/PC_APP/host

Build HostAPI dynamic library,

make

Change working directory to ‘app/ut2’,

cd $PATH/PC_APP/host/app/ut2

Edit file ‘settings.txt’ on $PATH/PC_APP/host/app/ut2 as below

```
INPUT0 ./data_pad32.bin
OUTPUT0 ./golden_mbox_loc.out
OUTPUT1 ./golden_mbox_conf_flatten.out
```

‘settings.txt’ defines a few keys to control APP flow.

- INPUT0 key is the input data of MobileNet-SSD
- OUTPUT0 key is the golden data-0 of MobileNet-SSD
- OUTPUT1 key is the golden data-1 of MobileNet-SSD

** If you don’t have the gold data or not want to compare the received data with the golden data; just delete these two keys, OUTPUT0/OUTPUT1.*

Build ut2 program,

make

Change working directory to the build folder,

cd $PATH/PC_APP/build

Perform *ut2* to connect to CV2x.

./ut2 169.254.197.80

And then *ut2* sends the same input data 'data_pad32.bin' to CV2x continually. The *cvflow_comm* of CV2x runs flexidag with this data and sends the test results back to PC. Then *ut2* compares these data with golden data. PC console will show

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

RX: Connection okay.
TX: Connection okay.

[UT2] Key: INPUT0, Value: ./data_pad32.bin
[UT2] file ./data_pad32.bin size 288000
[UT2] NN in num 1
[UT2] Key: OUTPUT0, Value: ./golden_mbox_loc.out
[UT2] file ./golden_mbox_loc.out size 7680
[UT2] Key: OUTPUT1, Value: ./golden_mbox_conf_flatten.out
[UT2] file ./golden_mbox_conf_flatten.out size 26848
[UT2] NN out_golden num 2
[TX] Seq: 0, TimeStamp:26760
[TX] Seq: 1, TimeStamp:26861
[TX] Seq: 2, TimeStamp:26977
[TX] Seq: 3, TimeStamp:27058
[RX] Seq: 0, TimeStamp:26760
```

Board side:

UART0 console will show

```
[ArmUtil_Mutex|DBG] Success to create mutex (XPCB_Mut)
[ArmUtil_ETH|DBG] ArmEth_Init() channel 0 connected!!
[ArmUtil_Mutex|DBG] Success to create mutex (XPCB_Mut)
[ArmUtil_ETH|DBG]
[ArmUtil_ETH|DBG] TX: Connection okay.
[ArmUtil_ETH|DBG] RX: Connection okay.
[ArmUtil_ETH|DBG]
[ArmUtil_MsgQ|DBG] Success to create MsgQueue (/EthMsgQ)
[ArmUtil_Mutex|DBG] Success to create mutex (EthCtrlMut)
[CtCvAlgoWrapper|DBG] Register callback for Slot 0
[CtCvAlgoWrapper|DBG] Register callback to idx 0
[CtCvAlgoWrapper|DBG] Register callback for Slot 0
[CtCvAlgoWrapper|DBG] Register callback to idx 1
[CCF|DBG]
[CCF|DBG] ------------------------------------
[CCF|DBG] [FlexiDag]Name :              SingleNN_0
[CCF|DBG] [FlexiDag]RetCode :           0
[CCF|DBG] [FlexiDag]Processing time :   7520us
[CCF|DBG] ------------------------------------
[CCF|DBG]
[CCF|DBG] ------------------------------------
[CCF|DBG] [FlexiDag]Name :              SingleNN_0
[CCF|DBG] [FlexiDag]RetCode :           0
[CCF|DBG] [FlexiDag]Processing time :   14752us
[CCF|DBG] ------------------------------------
[CCF|DBG]
[CCF|DBG] ------------------------------------
[CCF|DBG] [FlexiDag]Name :              SingleNN_0
[CCF|DBG] [FlexiDag]RetCode :           0
[CCF|DBG] [FlexiDag]Processing time :   21952us
[CCF|DBG] ------------------------------------
```