Vision SDK

Network Tools

User Guide

Document Revision v1.06

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards ought to be provided by the customer so as to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations and notices. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is neither responsible nor liable for any such use.

Resale of TI's products or services with *statements different from or beyond the parameters* stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products.
www.ti.com/sc/docs/stdterms.htm

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

TABLE OF CONTENTS

1 Introduction

This document describes the various networking tools that are available on PC side to interface with Vision SDK running on target (TDA2x or TDA3x EVM).

Following tools are available

- Network Control tool to send user defined commands and parameters to the target. Target will do a user defined action on receiving the command and can optionally return results to the PC.
- Network TX tool to send MJPEG compressed frames and RAW/YUV frames from PC to EVM. This tool can be used to feed pre-recorded test data to algorithms on target side
- Network RX tool to receive MJPEG compressed frames, RAW/YUV frames, meta-data buffers from EVM to PC. This tool can be used to send algorithm results to PC

IMPORTANT NOTE: These tools can be used only when BIOS based NDK/NSP runs on target (TDA2x EVM/MonsterCam or TDA3x EVM)

IMPORTANT NOTE: Currently the PC side tools are tested on Windows PC

1.1 Building the PC side tools

- Install GCC compiler for Windows (ex, http://www.codeblocks.org/)
- Install GNU Make for Windows (ex, “gmake” is available as part XDC install at $(xdc_PATH)/gmake)
- Install bash shell in Windows via tool like https://msysgit.github.io/ or Cygwin
- Edit RULES.MK at vision_sdk\tools\network_tools\build to point to installed path of GCC compiler (CGTOOLS_PATH=xxx) and installed path of GNU make (MAKE=xxx)
- Open bash shell prompt and goto vision_sdk\tools\network_tools\build
- Invoke GNU make to build the tools (“gmake” if XDC path is in your system path)
- Binaries are generated at vision_sdk\tools\network_tools\bin

1.2 Finding IP address of the target EVM

- All the networking tools used TCP/IP to exchange data with EVM. NDK needs to be enabled on EVM to support TCP/IP communication.
- All the networking tools need to know the IP address of the EVM in order to exchange information with it.
- By default NDK is enabled in Vision SDK
  - To make sure NDK is enabled, open command prompt at path “$(VISION_SDK_INSTALL)\vision_sdk” and type “gmake config”
  - Make sure “NDK_PROC_TO_USE” is NOT “none”
  - By default for TDA2x, NDK_PROC_TO_USE=a15_0
  - By default for TDA3x, NDK_PROC_TO_USE=ipu1_1
  - You can edit “Rules.make” (in “vision_sdk”) to change the CPU on which NDK runs. NDK can run on a15_0 or ipu1_0 or ipu1_1
  - When transferring large data it is recommended to run a15_0 (TDA2x) to get better network throughput

- By default IP address is set to DHCP mode
    - This can be changed to static IP by editing below file \vision_sdk\src\main_app\<soc>\cfg\NDK_config.cfg
    - Change at lines

      ```
      var enableStaticIP        = 1;
      if (enableStaticIP)
      {
          /* Settings for static IP configuration */
          Ip.address = "192.168.1.200";
          Ip.mask = "255.255.255.0";
          Ip.gatewayIpAddr = "192.168.1.1";
          Ip.ifIdx = 1;
      }
      ```

- Build and boot the Vision SDK on EVM as mentioned in Vision SDK user guide. Make sure network cable is connected. If DHCP mode is selected make sure the network can give a IP address to the EVM via DHCP.
- After boot note the IP address as mentioned in the main menu

  ```
  [IPU1-0]     23.294372 s:  Current System Settings,
  [IPU1-0]     23.294403 s:  ========================
  [IPU1-0]     23.294464 s:  Display Type   : HDMI 1920x1080 @ 60fps
  [IPU1-0]     23.294494 s:  Capture Source : Sensor OV10635 1280x720  @
  30fps - VIP, YUV422
  [IPU1-0]     23.294555 s:  My IP address  : 172.24.190.226
  ```

- This IP address will be used as input when running the PC side tools
- Ping the EVM from PC side to make sure its accessible on the network
    - Ex, $ ping 172.24.190.226
    - Make sure you see reply from EVM
    - If ping is not able to succeed then there is some issue in your network connectivity
- **IMPORTANT NOTE: Make sure "ping" is successful before trying any of the network tools**

2 Network Control Tool

This tool can be used to send user defined commands with parameters from PC side to the target. The target will respond with appropriate results. Users can extend this tool define their own set of commands, parameters and responses.

2.1 Tool Summary

Tool name	network_ctrl.exe
Tool source code (PC side)	vision_sdk\tools\network_tools\network_ctrl
Tool source code (target side)	vision_sdk\examples\tda2xx\src\links\network_ctrl

2.2 Tool Usage

- Make sure you know the IP address of the EVM
- Invoke the tool as shown below, it will print the supported options
  - $ network_ctrl.exe

```
#
# network_ctrl --ipaddr <ipaddr> [--port <server port>] --cmd <command string> <command parameters>
#
# (c) Texas Instruments 2014
#
# Supported commands,
# -------------------
# echo <string to echo>
# mem_rd <memory address in hex> <size of memory to read in units of 32-bit words>
# mem_wr <memory address in hex> <value to be written in units of 32-bit words>
# mem_save <memory address in hex> <size of memory to read in bytes> <filename in which data is saved>
# iss_raw_save <filename in which data is saved>
# iss_yuv_save <filename in which data is saved>
# iss_send_dcc_file <dcc file name to be sent>
# iss_save_dcc_file <Sensor ID> <dcc file name to be sent>
#       Supported sensors IDs are 140, 10640, 132 and 224
# iss_clear_dcc_qspi_mem <Sensor ID>
#       Supported sensors IDs are 140, 10640, 132 and 224
# iss_write_sensor_reg <chan num> <RegAddr> <RegVal>
# iss_read_sensor_reg <chan num> <RegAddr>
# iss_read_2a_params
# iss_write_2a_params <AE Mode> {Digital Gain} {Analog Gain} {Exposure Time} <AWB Mode> {Red Gain} {Green Gain} {Blue Gain} {Color Temparature}
# stereo_calib_image_save <filename prefix in which data is saved>
# stereo_set_params <numDisparities> <disparityStepSize> <disparitySearchDir>
<disparitySupportWinWidth> <disparitySupportWinHeight> <leftRightCheckEna>
<censusWinWidth> <censusWinHeight> <censusWinHorzStep>
<censusWinVertStep> <pp_colormap_index>
```

```
[EXAMPLE] stereo_set_params 128 4 0 11 11 0 9 9 2 2 0
```

```
#stereo_set_dynamic_params <pp_cost_max_thresh> <pp_conf_min_thresh>
<pp_texture_lumalothresh> <pp_texture_lumahithresh> <pp_texture_thresh>
<pp_leftright_thresh> <pp_maxdisp_dissimilarity> <pp_minconf_nseg_thresh>
```

```
#stereo_set_dynamic_params <pp_cost_max_thresh> <pp_conf_min_thresh>
<pp_texture_lumalothresh> <pp_texture_lumahithresh> <pp_texture_thresh>
<pp_leftright_thresh> <pp_maxdisp_dissimilarity> <pp_minconf_nseg_thresh>
```

```
[EXAMPLE] stereo_set_dynamic_params 95 98 0 100 85 255 2 2
```

- The “port” option need not be specified
- By default when NDK is enabled on target, a task is started on target side to listen to commands from this tool

```
#stereo_calib_lut_to_qspi <rectMapRight_int_converted.bin>
<rectMapLeft_int_converted.bin>
```

2.2.1 Supported commands

Current commands mentioned below are supported

Command	Description
echo	Takes a string as input and on successful execution, this string is printed on target side console
mem_rd	Reads specified memory contents from target and prints its value on PC side. Useful for dumping HW registers when JTAG is not connected
mem_wr	Writes a value to user defined memory location on target. Useful for writing a HW register when JTAG is not connected
mem_save	Reads specified memory contents from target and write to a file on PC. Useful to dump a chunk of memory like buffer data to a file on PC
mem_load	Reads data from a file on PC side and writes to specified memory location on target.
qspi_wr	Writes SBL/TESOC/AppImage into QSPI memory. It accepts Image name and address where image to be written. SBL need to flash at 0x0 and AppImage_BE at 0x80000. For TDA3xx TESOC image need to be written at 0x60000
sys_reset	EVM will receive command for COLD RESET
iss_raw_save	Dumps a frame of ISS raw data to a file on PC. Applicable only when ISS use-case with ISS capture link is running on target. Applicable to TDA3x ONLY.

iss_yuv_save	Dumps ISS simcop output frame in yuv format to a file on PC. Applicable only when ISS use-case with ISS simcop link is running on target. Applicable to TDA3x ONLY.
iss_send_dcc_file	Send DCC tuning file from PC to target and apply the settings to ISS HW. It applies the settings to the ISS HW immediately, so old settings will get overwritten with the new settings. But this will not save the new settings, so when the use case is rerun, it will rerun with the old settings. To save the new settings, use iss_save_dcc_file command which will save the settings in the QSPI or save the settings in the sensor specific driver (refer to the BSP driver user guide to get more information). Applicable only when ISS use-case with ISS capture link and ISS ISP link are running on target. Applicable to TDA3x ONLY.
iss_save_dcc_file	Send DCC tuning file from PC to target and target saves DCC tuning file to QSPI. Currently it is supported only for AR0140, OV10640 and AR0132 sensors. The QSPI offset for each sensor is fixed. If this is changed in network tool, it requires change in the target application also. Applicable only when ISS use-case with ISS capture link and ISS ISP link are running on target. Applicable to TDA3x ONLY.
Iss_clear_dcc_qspi_mem	Erases contents of the QSPI for the given sensor id. DCC profile is stored in the QSPI at a fixed location for the sensors AR0140, OV10640, AR0132 and IMX224. This API is used to erase this memory. Applicable only when ISS use-case with ISS capture link and ISS ISP link are running on target. Applicable to TDA3x ONLY.
Iss_write_sensor_reg	Writes the 16bit value to the given sensor registers. Channel id is used to select the sensor. For single sensor usecase, channel id must be set to 0. For multiple sensor usecase, appropriate channel id should be used. Used by the DCC (Dynamic Camera Configuration Tool) to write to sensor registers. Applicable only when ISS use-case with ISS capture link is running on target. Applicable to TDA3x ONLY.
Iss_read_sensor_reg	Reads a value of the given sensor register and prints it on PC side. Channel id is used to select the sensor. For single sensor usecase, channel id must be set to 0. For multiple sensor usecase, appropriate channel id should be used. Used by the DCC (Dynamic Camera Configuration Tool) to read sensor registers. Applicable only when ISS use-case with ISS capture link is running on target. Applicable to TDA3x ONLY.
iss_read_2a_params	Used to read AEWB algorithms current output parameters. This includes AEWB algorithm mode, analog/digital gains, exposure time and color temperature. Applicable only when ISS use-case with ISS capture link is

	running on target. Applicable to TDA3x ONLY.
iss_write_2a_params	Used to set the AEWB mode to manual and set the output parameters to fixed value. This includes AEWB algorithm mode, analog/digital gains, exposure time and color temperature. Applicable only when ISS use-case with ISS capture link is running on target. Applicable to TDA3x ONLY.
stereo_calib_image_save	This command saves a frame each from left and right sensor into pgm files. The final filename for the left image will be composed by the strings ‘left_’+’ <filename_prefix> + ‘.pgm’. Similarly, the filename for the right image will be composed by the strings ‘right_’+’ <filename_prefix> + ‘.pgm’ It is used during stereo calibration process.
stereo_set_params	This command sets stereo parameters. It should be sent before any stereo usecase is started. All the parameters are configured together and there is no means of changing specific items. It takes following parameters : <numDisparities> <disparityStepSize> <disparitySearchDir> <disparitySupportWinWidth> <disparitySupportWinHeight> <leftRightCheckEna> <censusWinWidth> <censusWinHeight> <censusWinHorzStep> <censusWinVertStep> <pp_colormap_index>whose default values are 128 4 0 11 11 0 9 9 2 2 0 respectively. Please refer to the StereoVision_DSP_UserGuide.pdf inside the ti_components\algorithms_codecs\REL.200.V.ST.C66X.00.XX.XX.XX\modules\ti_stereovision\docs directory, for details on each parameter.
stereo_set_dynamic_params	This command sets stereo parameters on the fly. It can be sent after any stereo use-case is started. All the parameters are configured together and there is no means of changing specific items. It takes following parameters : <pp_cost_max_thresh> <pp_conf_min_thresh> <pp_texture_lumalothresh> <pp_texture_lumahithresh> <pp_texture_thresh> <pp_leftright_thresh> <pp_maxdisp_dissimilarity> <pp_minconf_nseg_thresh>whose default values are 95 98 0 100 85 255 2 2 respectively. Please refer to the *StereoVision_DSP_UserGuide.pdf* inside the *ti_components\algorithms_codecs\REL.200.V.ST.C66X.00.XX.XX.XX\modules\ti_stereovision\docs* directory, for details on each parameter.
stereo_calib_lut_to_qspi	This commands writes into NAND flash memory the lookup tables generated by the camera automatic calibration tool for use by the remap algorithm running on EVE.

It takes following parameters:

<rectMapRight_int_converted.bin>
<rectMapLeft_int_converted.bin>

These are files generated by the camera automatic calibration tool. Please see the *VisionSDK_MultiSensorFusionStereoCalibrationGuide.pdf* for further details.

On the PC side, the following output should be displayed:

```
# NETWORK: Connected to Server (172.24.190.212:5000)!!!
# FILE: Reading file [rectMapRight_int_converted.bin] ...
Done. [1136472 bytes]
# FILE: Reading file [rectMapLeft_int_converted.bin] ...
Done. [1136472 bytes]
# Writing the QSPI tags
# Command stereo_calib_lut_to_qspi: Sent 2272960 bytes
# Command stereo_calib_lut_to_qspi: Received reponse
(status = 0, prmSize = 0)
```

Whereas on the target's side, the console window should show something similar to:

```
[HOST  ]    119.827209 s:  NETWORK_CTRL: Received command
[stereo_calib_lut_to_qspi], with 2100032 bytes of
parameters
[HOST  ]    119.903656 s:  NETWORK_CTRL: qSpiOffset =
30408704, size = 2100032
[HOST  ]    119.903686 s:  QSPI Init Started
[HOST  ]    119.903808 s:  MID - 1
[HOST  ]    119.903808 s:  DID - 18
[HOST  ]    119.903808 s:  QSPI Init Completed Sucessfully
[HOST  ]    119.903839 s:  QSPI Write Started, please
wait!
[HOST  ]    120.003295 s:  QSPI Erase Block Started,
please wait!
[HOST  ]    123.427154 s:  QSPI Erase Block Completed
Sucessfully
[HOST  ]    126.839233 s:  QSPI Write Completed
Sucessfully
[HOST  ]    126.840728 s:  NETWORK_CTRL: Sent response for
command [stereo_calib_lut_to_qspi], with 0 bytes of
parameters
```

Note that due to the size of the tables, it will take a long time before seeing the text output 'QSPI Write Completed'.

2.3 Extending the tool

NOTE: Users needs to read this section only if they plan to extend the tool by adding their own commands and command handlers, others can ignore this section.

This tool can be extended by users to add more commands from PC side and "plugin" different actions on target side when the command is received.

TIP: The easiest way to extend the tool is see an existing command implementation and customize it

- On PC side do the below
  - Create your own file for the command, ex, network_ctrl_handle_echo.c (vision_sdk\tools\network_tools\network_ctrl\src)
  - Write a function that will send the command and receive the results, ex, handleEcho
  - A command is string of characters
  - Use below API to send the command, "params" point to the user specific parameters of "size" bytes
    - SendCommand(char *command, void *params, int size);
    - Command line parameters are available inside the user handler in the below structure fields
      - gNetworkCtrl_obj.params[x]
  - Use below API to receive results, "prmSize" is size of result parameters
    - RecvResponse(char *command, UInt32 *prmSize);
  - If "prmSize" is not zero then read the parameters using below API into user pointer
    - RecvResponseParams(char *command, UInt8 *pPrm, UInt32 prmSize);
  - Register this handler using below API in function Init()
    - void RegisterHandler(char *command, void (*handler)(), int numParams)
  - Now build and run the tool with the new command and command specific parameters
- On target side do the below
  - Create your own file for handling the command, ex, network_ctrl_handle_echo.c (vision_sdk\examples\tda2xx\src\links\network_ctrl)
    - Void NetworkCtrl_cmdHandlerEcho(char *cmd, UInt32 prmSize)
  - This handler is called, when the "cmd" is received, "prmSize" is size of parameters that accompany this command
  - Use below API to read the received parameters
    - Int32 NetworkCtrl_readParams(UInt8 *pPrm, UInt32 prmSize)
  - Based on the parameters handle the command
  - Use below API to send reply for the command
    - Int32 NetworkCtrl_writeParams(UInt8 *pPrm, UInt32 prmSize, UInt32 returnStatus)
  - Register the handler in NetworkCtrl_init() (vision_sdk\examples\tda2xx\src\links\network_ctrl\network_ctrl_tsk.c) using below API
    - Int32 NetworkCtrl_registerHandler(char *cmd, NetworkCtrl_Handler handler)

- Ex, NetworkCtrl_registerHandler("echo", NetworkCtrl_cmdHandlerEcho);

- Once handler are registerd on PC side and target side, recompile and run the binaries
- On PC side call the tool with the new command and parameters as input
  - Ex, **network_ctrl --ipaddr 192.168.1.200 --cmd echo “hello, world !”**

2.4 Communication Protocol

NOTE: Users needs to read this section only if they plan to implement their own tool on PC side to interface with the target side, others can ignore this section.

The tool uses TCP/IP as underlying communication protocol. On top of this a thin application layer protocol is added as described below.

This protocol needs to be used in case other tools are made on PC which need to interface to the target side.

See also vision_sdk\tools\network_tools\common\inc\networkCtrl_if.h

NOTE: All fields are specified in little-endian order

2.4.1 Command protocol

Fields	Bytes	Description
Header TAG (0x1234ABCD)	4	HEADER TAG to make sure this is not a spurious data on the network port
Command	64	Command to be sent. Represented as a string of NULL terminated characters
returnValue	4	Set to 0 when sending command, filled by response
Flags	4	Flags for specific status control
prmSize	4	Size of parameters that follow. Can be set to 0 when no parameters are present
Params	“prmSize”	Stream of bytes representing parameters, MUST be equal to “prmSize” that is specified earlier

2.4.2 Response protocol

The response protocol format is exactly same as command protocol with the following differences

- “Command” is set to same value as the command that was received
- “returnValue” is set based result of command execution. Value itself depends on command. Typically value of 0 indicates successful command execution
- “Flags” is set with value of 0x00000001 to indicate that this response is ACK to earlier command
- “prmSize” is set to parameter of results, i.e parameter sent as response are different from parameters sent as command
- “Params” is the response / result parameters

3 Network TX Tool

Network TX tool is used to send MJPEG compressed frames and RAW/YUV frames from PC to EVM. This tool can be used to feed pre-recorded test data to algorithms on target side.

IMPORTANT NOTE:

- On TDA3x, though MPJEG frames can be sent to TDA3x, there is no MJPEG decode on TDA3x to decode the frames. Hence on TDA3x, typically one would use RAW/YUV frame for network TX

IMPORTANT NOTE:

- When sending RAW/YUV frames
  - When NDK runs on M4 CPU, one can achieve a data rate of about 2 MB/s (16Mbps)
  - When NDK runs on A15 CPU (TDA2x ONLY), one can achieve a data rate of about 60 MB/s (480Mbps)
  - Make sure to select the appropriate CPU to run the NDK depending on the SoC that is used

3.1 Tool Summary

Tool name	network_tx.exe
Tool source code (PC side)	vision_sdk\tools\network_tools\network_tx
Tool source code (target side)	vision_sdk\src\links_common\nullSrc
Example usage (target side)	vision_sdk\examples\tda2xx\src\usecases\network_rx_tx chains_networkRxDecDisplay*.* chains_networkRxDisplay*.*

3.2 Tool Usage

- Make sure you know the IP address of the EVM (see 1.2 Finding IP address of the target EVM)
- Create a Vision SDK use-case on target side
  - Specify the source of data as "NullSrc" link.
    - **IMPORANT:** Make sure "NullSrc" is on the same CPU that NDK is enabled
  - Set NullSrcLink_CreateParams.dataRxMode as NULLSRC_LINK_DATA_RX_MODE_NETWORK
  - Set NullSrcLink_CreateParams.networkServerPort, if not specified default value is used.
    - This value needs to match the "—port" value specified in PC tool
    - Typically when only single "NullSrc" link exists in use-case no need to specify this value
    - When more than one "NullSrc" link is present in use-case then each "NullSrc" link MUST have unique networkServerPort. In this case, "network_tx.exe" is invoked multiple times on PC with matching "—port" value

    - Set other "NullSrc" create parameters like data type, MJPEG or RAW/YUV and frame resolution, like width, height
    - Set NullSrcLink_CreateParams.timerPeriodMilliSecs to specify the rate as which data should be sent, ex, set to 33 for approx. 30fps receive frame-rate
- Some examples which use the "NullSrc" link to receive data over network can be found at below path
    - vision_sdk\examples\tda2xx\src\usecases\network_rx_tx
        - chains_networkRxDecDisplay*.*
        - chains_networkRxDisplay*.*
- Compile and run the use-case as usual. Once the use-case is running on the target side …
- Invoke the tool as shown below, it will print the supported options

```
$ network_tx
# ERROR: IP Address of server MUST be specified
# ERROR: Atleast one input file MUST be specified


#
# network_tx --ipaddr <ipaddr> [--port <server port>] --files <CH0 file> <CH1 file> ...
#
# (c) Texas Instruments 2014
#
```

- The "port" option when not specified default port is used.
- Multiple "channels" of data can be sent by specifying multiple files.
- No need to specify the data type, frame resolution etc on the PC side, this is specified on the target side
- When use-case runs on the target, target will request frames from PC side, PC side will read from the input files and send the data over network
- "network_tx.exe" can be invoked multiple times, each invocation feeding different "NullSrc" links in the use-case on target
- Once the input file reaches "end of file", input is read again from start of the file, i.e it will continuously stream the data until either the use-case stops or tool exit by doing "Ctrl-C"
- For 'Network + stereo + Display' use-case, <Ch0 file> corresponds to the left camera and <Ch1 file> corresponds to the right camera.

4 Network RX Tool

Network RX tool is used to receive MJPEG compressed frames and RAW/YUV/Meta data frames from EVM to PC. This tool can be used to save data from algorithms on PC side.

IMPORTANT NOTE:

- On TDA3x, MPJEG frames can not sent to PC, since there is no MJPEG encoder on TDA3x

IMPORTANT NOTE:

- When sending RAW/YUV frames
  - When NDK runs on M4 CPU, one can achieve a data rate of about 2 MB/s (16Mbps)
  - When NDK runs on A15 CPU (TDA2x ONLY), one can achieve a data rate of about 60 MB/s (480Mbps)
  - Make sure to select the appropriate CPU to run the NDK depending on the SoC that is used

4.1 Tool Summary

Tool name	network_rx.exe
Tool source code (PC side)	vision_sdk\tools\network_tools\network_rx
Tool source code (target side)	vision_sdk\src\links_common\null
Example usage (target side)	vision_sdk\examples\tda2xx\src\usecases\network_rx_tx chains_networkTxEncDisplay*.* chains_networkTxDisplay*.*

4.2 Tool Usage

- Make sure you know the IP address of the EVM (see 1.2 Finding IP address of the target EVM)
- Create a Vision SDK use-case on target side
  - Connect the output of the algorithm or link for which data needs to be saved to PC to a "Null" link
    - **IMPORANT:** Make sure "NullSrc" is on the same CPU that NDK is enabled
  - Set NullLink_CreateParams. dumpDataType as NULL_LINK_COPY_TYPE_NETWORK
  - Set NullLink_CreateParams.networkServerPort, if not specified default value is used.
    - This value needs to match the "—port" value specified in PC tool
    - Typically when only single Null link exists in use-case no need to specify this value
    - When more than one "Null" link is present in use-case then each "Null" link MUST have unique networkServerPort. In this case, "network_rx.exe" is invoked multiple times on PC with matching "—port" value
  - Set other "Null" create parameters as usal

- Some examples which use the “Null” link to receive data over network can be found at below path
  - vision_sdk\examples\tda2xx\src\usecases\network_rx_tx
    - chains_networkTxEncDisplay*.*
    - chains_networkTxDisplay*.*
- Compile and run the use-case as usual. Once the use-case is running on the target side …
- Invoke the tool as shown below, it will print the supported options

```
# ERROR: IP Address of server MUST be specified
# ERROR: Atleast one output file MUST be specified

#
# network_rx --ipaddr <ipaddr> [--port <server port>] --files <CH0 file> <CH1 file>
#
# (c) Texas Intruments 2014
#The “port” option when not specified default port is used.
```

- Multiple “channels” of data can be saved by specifying multiple files.
- No need to specify the data type, frame resolution etc on the PC side, PC side will just save whatever is sent by target side
- When use-case runs on the target, target will send frames to PC side via the “Null” link, PC side will write the received frames to files specified by the user
- “network_rx.exe” can be invoked multiple time, each invocation receiving data from different “Null” links in the use-case on target
- Once the use-case on target is stooped, the tool on PC exits and saved file can be viewed/analysed as required. Alternatively “Ctrl-C” can be used to stop the tool and close the file which is being written