Proxy Based Back Channel Flow

Wei-hsing.Huang@SPISim.com

Ver. 20191118

Overview:

- Assumptions
- Motivation
- Features
- Example flow
- Notes

- Example codes:
  http://www.spisim.com/support/ticket/IBISATM/ProxyBackChannelTxRx.zip

Assumptions:

TX/RX AMI model developer knows RX/TX AMI “binary” model’s capabilities and training protocols etc. (Proprietary API level is fine)

Motivation:

- Avoid using file based data exchange:
  - File IO is not efficient
  - Error prone due to e.g. possible IO buffering/delay

- No IBIS-AMI spec. changes needed:
  - Work with existing IBIS-AMI spec.
  - Work on existing or even order EDA tool

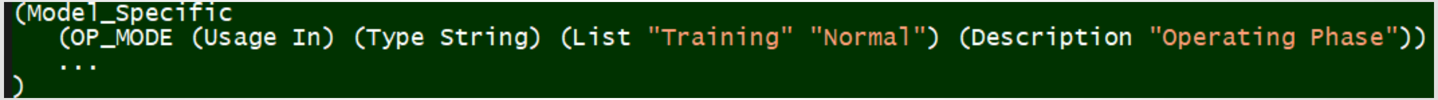
- More efficient and flexible:
  - Direct data exchange/training between Tx and Rx .dlls/.sos
  - Freedom to define any training required proprietary API functions
  - Easier debugging/development process

Features:

- File IO twice only:
  - Once for TX and RX at the very beginning
  - Once for TX and RX when training is done
- “Proxy” based “coupling”:
  - RX (.dll/.so) loads TX (.dll/.so)
  - Subsequent iterative/proprietary function calls between these two only
    - Total transparent to EDA tool/simulator (it does not know this is happening)
- Make use of existing mechanism:
  - OS System’s username, temp folder etc using standard “getenv” etc
  - IBIS-AMI spec’s DLL_ID and DLL_PATH

Example Flow: OP. Mode

- Two op-modes:
  - Normal operation (non-training) mode
    - Tx/Rx need to have settings ready (e.g. default values or config. file)
  - Training mode
    - For training and generate settings (config. file) when done
  - Training protocol defined by TX or RX binary model provider can specify a common AMI parameter for this purpose
    - e.g.:



```
(Model_Specific
   (OP_MODE (Usage In) (Type String) (List "Training" "Normal") (Description "Operating Phase"))
   ...
)
```

Normal Mode:

- TX AMI and RX AMI:
  - Read “config. setting“ file from where .dll/.so is located
  - If not found, use default values or report fatal errors.
  - Initialize EQ using these values

- EDA tool:
  - Perform normal channel simulation just like most of the cases
  - AMI_Init->(one or more AMI_GetWave)->AMI_Close
  - Nothing special in this mode!

Training Mode: TX

- TX AMI:
    - Delete existing “config. file“ if found
    - Become a “Pass-Through”
        - So that RX AMI will see un-equalized channel response later

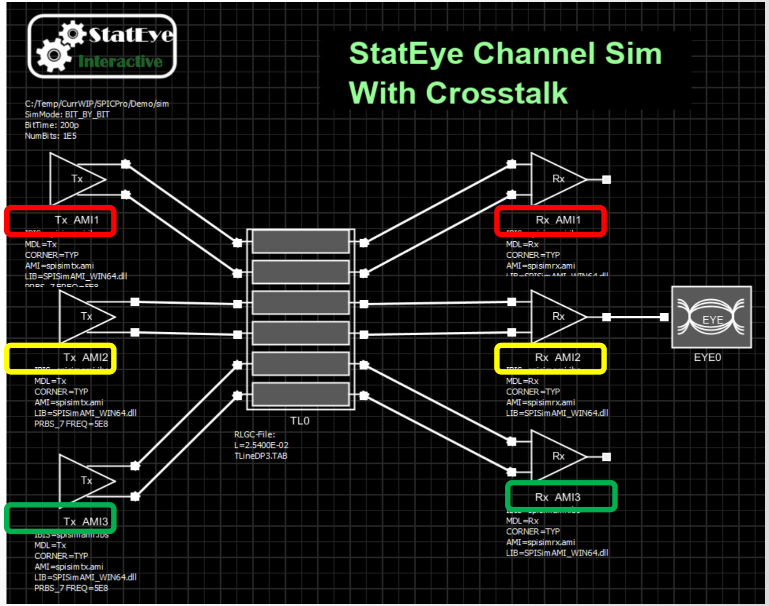
    - Generate a file (First File-IO):
        - Obtain its own info using DLL_ID and DLL_PATH
        - Obtain process ID info using e.g. GetCurrentProcessId()
        - Obtain user name and a common path using e.g “getenv”
            - Using OS environ. variables such as “username”, “temp”, “LocalAppData”
            - These values are common to all processes on this PC/OS
        - Store these info. to a file specified on next slide:
            - DLL_PATH/DLL_ID.dll/.so (i.e. path to this TX .dll/.so)
            - ami parameters received from EDA tool

Training Mode: TX

- Write to this file (colored text are variables):
  - E.g. TEMP/USERNAME_PROCID_DLLID_TIMESTAMP.txt
  - E.g. /tmp/whuang4_ 40228_RX_AMI1_2019103118000000.txt
  - Explanations:
    - TEMP: so that RX know where to find this file
    - USERNAME: for differentiation in case this is a multi-user server
    - PROCID: process ID, for differentiation in case multi-treaded or running >1 instance of EDA tools
    - DLLID & TIMESTAMP:
      - for differentiation in case when multiple TX-RX pairs are involved
      - for differentiation in case there are old/left-over file (e.g. timing tolerance < 5 sec.)
      - see next slides for "Heuristic" search algorithm
  - RX AMI should be able to find this file by itself!
    - RX knows: TEMP/USERNAME_PROCID_~~DLLID~~_TIMESTAMP.txt
    - If there is only one such file (i.e. one pair TX-RX), DLLID is irrelevant.

Multi TX-RX pairs scenarios:

- Use TimeStamp for heuristic search
  - Tx-Rx pairs can be identified if EDA tool call them in this order:
    - Tx$_i$, Rx$_i$, Tx$_j$, Rx$_{j,}$ Tx$_k$, Rx$_{k\ldots}$ or
    - Tx$_i$, Tx$_j$, Tx$_k$, Rx$_i$, Rx$_j$, Rx$_k$
  - These ordering should cover most of the cases!
  - Using “oldest” timestamp, Rx will find corresponding Tx
  - Once found, Rx will delete corresponding Tx’s file.



C:/Temp/CurrWIP/SPICPro/Demo/sim
SimMode: BIT_BY_BIT
BitTime: 200p
NumBits: 1E5

Tx_AMI1
MDL=Tx
CORNER=TYP
AMI=spisimtx.ami
LIB=SPISimAMI_WIN64.dll

Tx_AMI2
MDL=Tx
CORNER=TYP
AMI=spisimtx.ami
LIB=SPISimAMI_WIN64.dll
PRBS_7 FREQ=5E8

Tx_AMI3
IBIS=spisimami.ibs
MDL=Tx
CORNER=TYP
AMI=spisimtx.ami
LIB=SPISimAMI_WIN64.dll
PRBS_7 FREQ=5E8

TL0
RLGC-File:
L=2.5400E-02
TLineDP3.TAB

Rx_AMI1
MDL=Rx
CORNER=TYP
AMI=spisimrx.ami

Rx_AMI2
MDL=Rx
CORNER=TYP
AMI=spisimrx.ami
LIB=SPISimAMI_WIN64.dll

Rx_AMI3
MDL=Rx
CORNER=TYP
AMI=spisimrx.ami
LIB=SPISimAMI_WIN64.dll

Training Mode : RX

- RX AMI:
    - Delete RX’s existing “config. file“ if existed
    - Identify the file upstream TX has created during its AMI_Init
        - Fatal error if not found.
    - Parse that file and obtain these info.:
        - Path to the TX .dll/.so
        - ami parameters that TX saw initially
        - delete or rename this file (to support multi TX/RX pairs scenario)
    - Load the TX AMI .dll/.so (proxy pattern)

Training Mode: RX

- At this point:
  - RX has un-equalized channel response (1st TX was a pass-through)
  - TX (2nd instance) has been loaded by this RX
  - RX has ami parameters TX should see (pass that to TX!)
  - RX has RX's ami parameters from EDA tool
  - RX and TX know proprietary APIs methods available to them two

- Start training:
  - Iteration between TX and RX
    - Developer decides this should happen in RX's AMI_Init or AMI_GetWave
  - TX can train RX or vice versa right here
    - They are "dancing" together, only themselves need know who is leading…
    - It's the 2nd instance TX which RX is training or being trained.

Training Mode: RX

- When training is done:
  - RX save optimized config. settings to a file where RX.dll is located
  - RX tell TX to save its optimized settings where TX.dll is located
  - RX release TX .dll/.so (2nd instance) it has loaded
- EDA proceed as normal operation:
  - RX release memory when its AMI_Close is called.
  - “Pass-through” TX (1st instance) release memory in its AMI_Close
  - EDA tool does not know what has just happened between TX and RX!
- Training mode finished!
  - Now TX/RX can operate in “Normal” mode.

Notes:

- Versioning TX/RX .dll
  - So that compatible version can be trained together.
- Be careful about "static" variables:
  - They will be seen in the same process, across different .dlls
- Repeater(s) are present in the channel?
  - This flow should still work if they are LTI
  - This flow may **NOT** work if one or more of them are NLTV (non-LTI)
    - In this case, channel response RX received is not useful even though 1st TX is a pass-through.
    - Only EDA tool can arrange proper calling order along the channel

Notes: Training Modes

- TX .dll/.so has 3 operation modes:
  1. Normal mode
  2. Training mode & Pass-through
     - This is default settings in training mode
     - When TX is loaded by EDA tool, it is in this mode (1st Tx Instance)
  3. Training mode & Optimization
     - This will happen in API defined optimization functions
     - When Tx is loaded by Rx (2nd Tx Instance), Tx will be in this mode

Notes: Mixed 32-bit/64-bit dll?

- 32/64 dll can’t be mixed without advanced handling

  See:
  - https://social.msdn.microsoft.com/Forums/en-US/06176268-79b1-4b2b-a981-eba89b578949/mixing-32-and-64-bit-code?forum=netfx64bit
  - https://stackoverflow.com/questions/9292345/mixing-32bit-and-64bit-managed-assemblies

- In such case, PROCID will be different
- This will still work for single-threaded case:
  - TEMP/USERNAME_~~PROCID~~_~~DLLID~~_TIMESTAMP.txt

SPISIM

EDA Expertise in Signal, Power Integrity & Simulation