2024 IEEE INTERNATIONAL SYMPOSIUM ON ELECTROMAGNETIC COMPATIBILITY, SIGNAL & POWER INTEGRITY

WWW.EMC2024.ORG

#IEEE_ESP24

Hybrid IBIS Summit with IEEE EMC+SIPI 2024
Phoenix, Arizona, USA
August 9, 2024

AMI DLL Hook: A Novel IBIS-AMI Simulation Debugging Method for Model Users

Chuanyu Li, Alaeddin A Aydiner, Sleiman Bou-Sleiman, Xinjun Zhang

Chuanyu Li

Chuanyu is a signal integrity engineer at Intel.

He received his B.S. degree and M.S. degree in electrical engineering from Harbin Institute of Technology, Harbin, China, in 2014 and 2016, respectively, working on electromagnetics simulation and power electronics. He also studied as an exchange student in National Taiwan University of Science and Technology, Taipei, in 2012-2013.

He has been working as a signal and power integrity engineer since his graduation in 2016. After joining Intel in 2022, he has been focusing on signal integrity. His current research interests include die-to-die connection protocols and buffer modeling.

Sleiman Bou-Sleiman

Sleiman is a Senior Staff engineer and Analog and Mixed Signal IP Architect at Intel.

Sleiman received his B.E. in Computer and Communication Engineering from the American University of Beirut, Lebanon in 2005, M.Sc. in Electrical Engineering from the Swedish Royal Institute of Technology (KTH) in 2007 and Ph.D. in Electrical Engineering from The Ohio State University in 2011. His research dealt with PLL frequency synthesis as well as robustness enhancement techniques and efficient built-in-testing for RFICs. Since joining Intel, Sleiman has been working on high-speed wireline SerDes transceiver architectures.

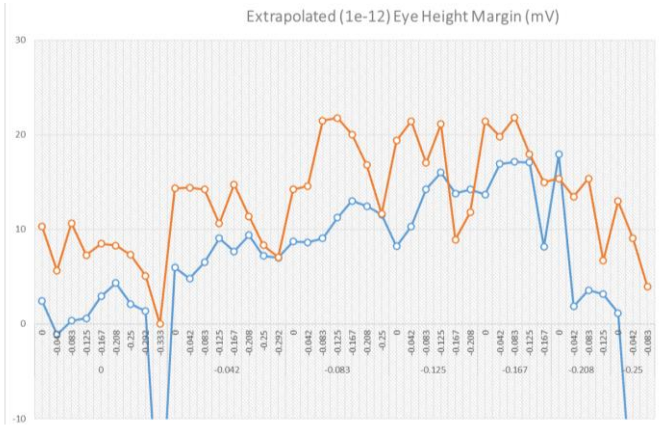
Sleiman has authored and co-authored a number of journal and conference papers, patents, book chapter, and a book on RF SoC Built-in-Self-Test and digital self-calibration. He is also technical reviewer for a number of journals and Transactions, an expert evaluator for the EU's European Innovation Council, and serves on the steering committee of the IEEE Midwest Symposium on Circuits and Systems (MWSCAS).

Agenda

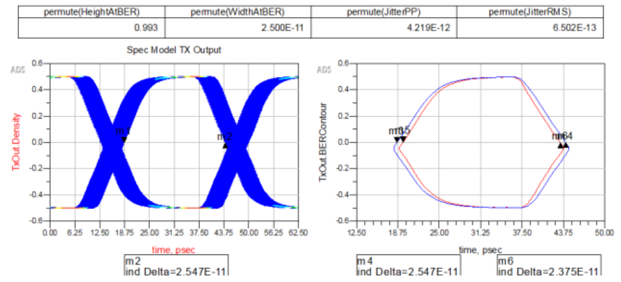
- Motivation
- Brief mechanism of IBIS-AMI simulation and hook debugging
- Implementation of the AMI DLL hook
- Practice of AMI DLL hook in SerDes AMI model alternative EDA enabling project
- Conclusion and future work

Motivation

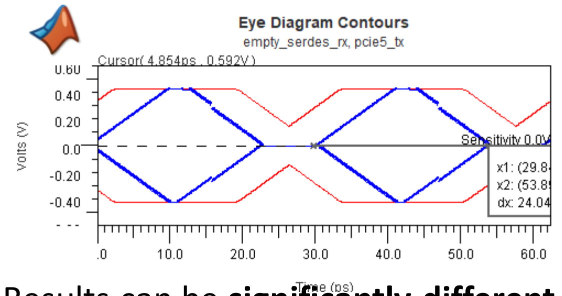
- I/O Buffer Information Specification (IBIS) Algorithmic Modeling Interface (AMI) model is now widely accepted in industry for SerDes signal integrity simulation.
- When running simulation with **IBIS-AMI** models, the following problems are common:
  - **Trend** of IBIS-AMI simulation is **not well-matched** to silicon simulation.
  - Simulation results are **different in different EDA software**.
- As a model user, it is difficult to debug the simulation because:
  - No debugging features defined in IBIS spec yet.
  - Try-and-run is inefficient, time-consuming, and is not guaranteed to solve the problem.
  - Few models or EDA provide enough debug dumps.
- **A general debugging method is needed for IBIS-AMI simulations!**



Trend of AMI simulation doesn't match silicon simulation well



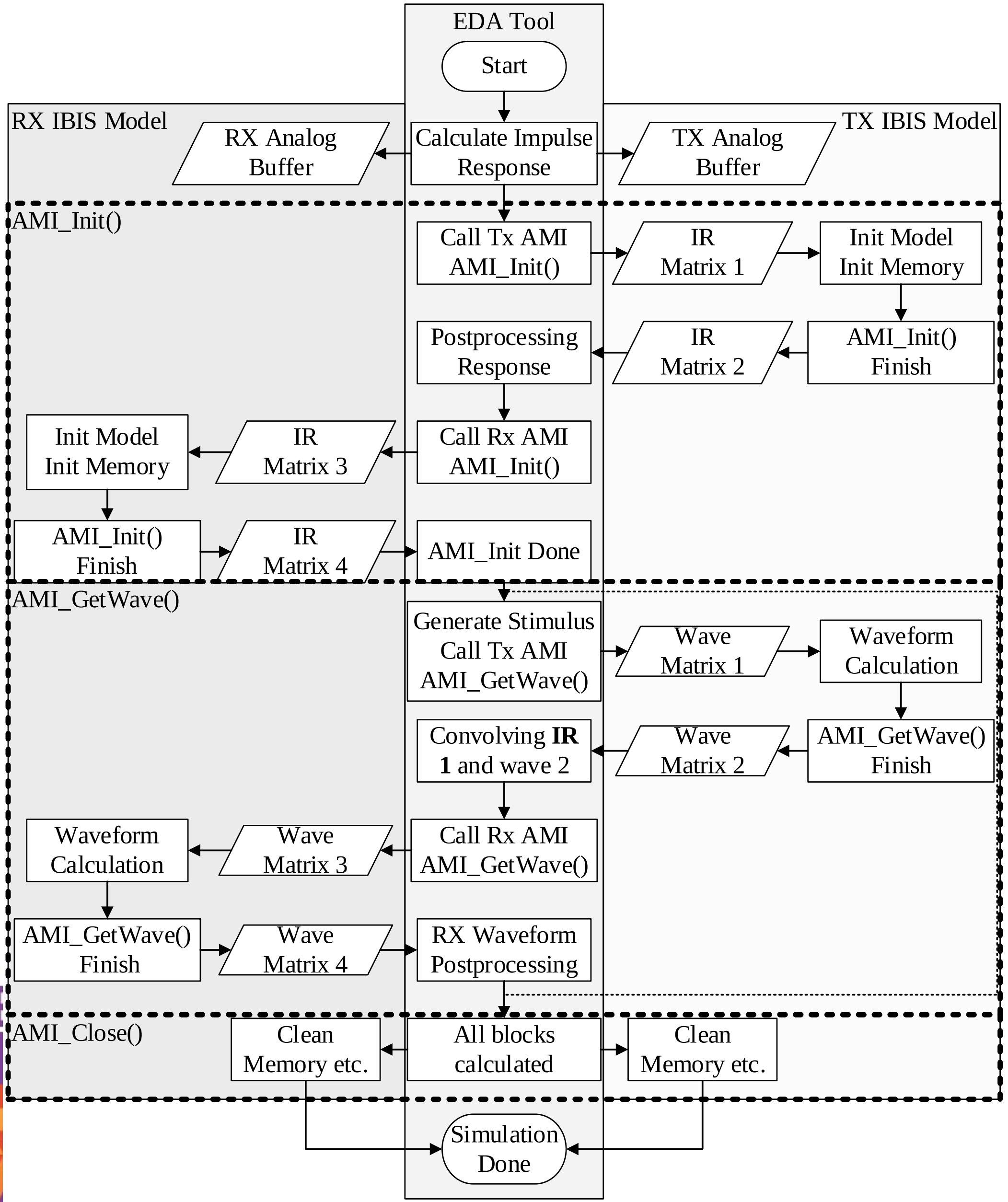
permute(HeightAtBER)	permute(WidthAtBER)	permute(JitterPP)	permute(JitterRMS)
0.993	2.500E-11	4.219E-12	6.502E-13



Results can be **significantly different** in different EDA software with same channel

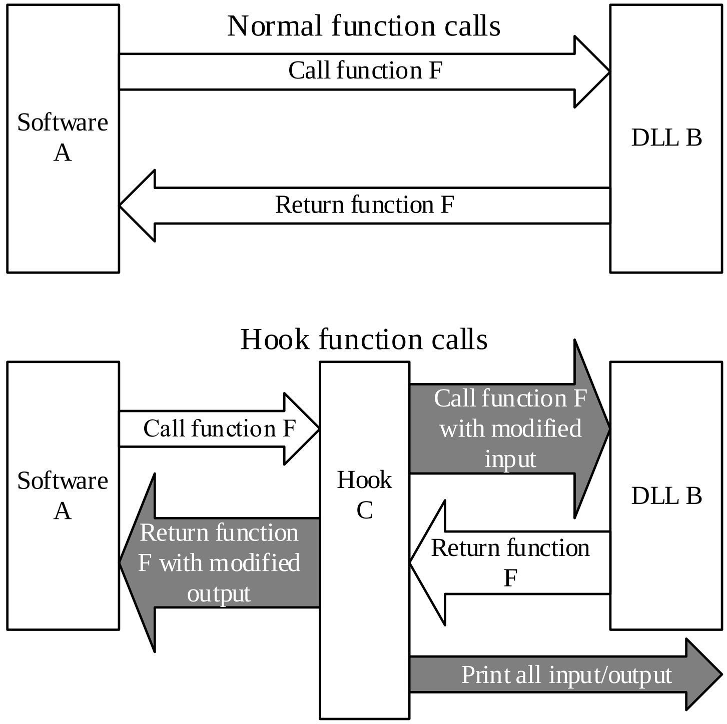
Brief mechanism of IBIS-AMI simulation

- AMI executable model file is packed as DLL or SO format library
- The process of EDA software calling AMI DLL files is a typical API calling process
- There are many API calls in an AMI simulation process.
- An API call contains much information and could be helpful for debugging.



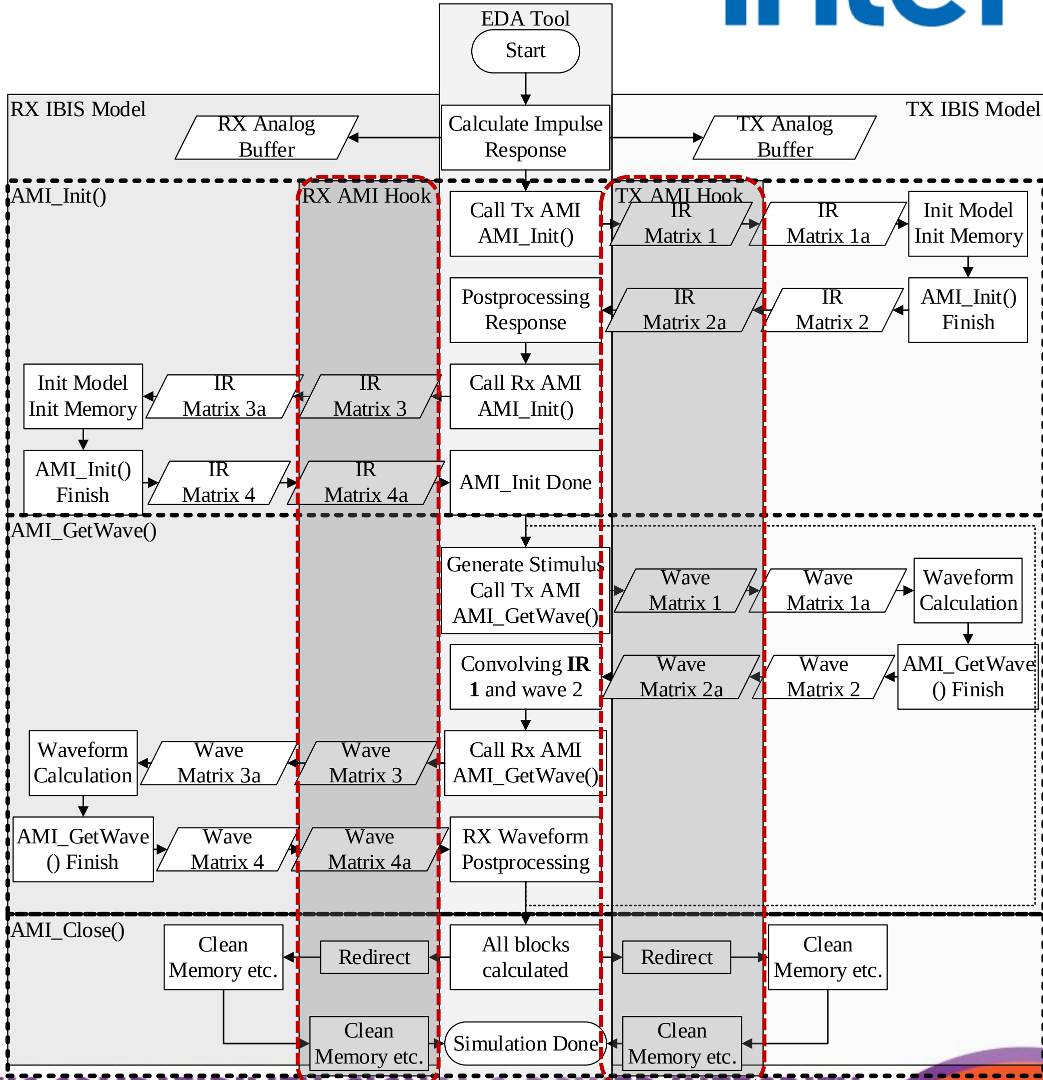
Brief mechanism of hooking

- Hooking is the procedure that changes the system calls in a way that its own functions were used instead of the original ones.
- Create "**breakpoints**" on API calls.
  - "Breakpoint" here means monitoring/dumping data and altering data are possible, just like a breakpoint in software debugging domain.
- Can be used for monitoring application communication and altering behaviors of the API calls.
- Hook functions should be defined exactly the same as the original functions. **Hooking is possible to be injected into the AMI simulation** as AMI functions are defined in IBIS spec.



Brief mechanism of hook debugging in IBIS-AMI simulation

- Injecting hooks creates **four breakpoints** in analytical simulations and at least **eight breakpoints** in transient simulations.
- The breakpoints provided extra dump information and provides the ability to alter the data, which helps debugging in IBIS-AMI simulation process.



Implementation of AMI DLL hook

- Implementation of an AMI DLL hook is divided into two major approaches:
  - API hooking
  - DLL injection
- API hooking programming
  - Purpose is to create a DLL capable of performing "API hooking"
  - Six necessary steps:

1). Export AMI standard API functions

```
extern "C" __declspec(dllexport) long AMI_Init(
    double* impulse_matrix,
    long number_of_rows,
    long aggressors,
    double sample_interval,
    double symbol_time,
    char* AMI_parameters_in,
    char** AMI_parameters_out,
    void** AMI_memory_handle,
    char** msg
);
```

2). Load original model

```
CHAR DllPath[1024] = { 0 };
GetModuleFileNameA(
    (HINSTANCE)&__ImageBase,
    DllPath,
    _countof(DllPath)
);
PathRemoveFileSpecA(DllPath);
PathCombineA(
    DllPath,
    DllPath,
    AMI_RUNTIME
);
this->hAMI = LoadLibraryA(DllPath);
```

3). Obtain original API function addresses

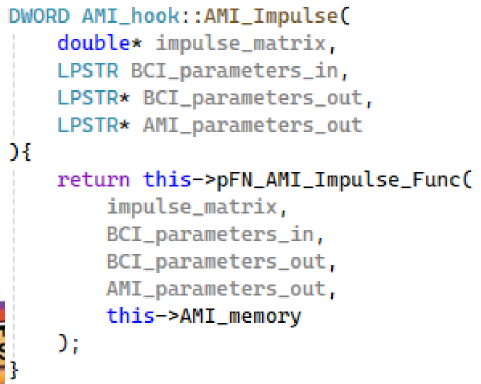
```
typedef long (CALLBACK* pFN_AMI_Init)(
    double* impulse_matrix,
    long number_of_rows,
    long aggressors,
    double sample_interval,
    double symbol_time,
    char* AMI_parameters_in,
    char** AMI_parameters_out,
    void** AMI_memory_handle,
    char** msg
    );
```

```
this->pFN_AMI_Init_Func = (pFN_AMI_Init)GetProcAddress(
    this->hAMI,
    "AMI_Init"
);
this->pFN_AMI_Close_Func = (pFN_AMI_Close)GetProcAddress(
    this->hAMI,
    "AMI_Close"
);
```

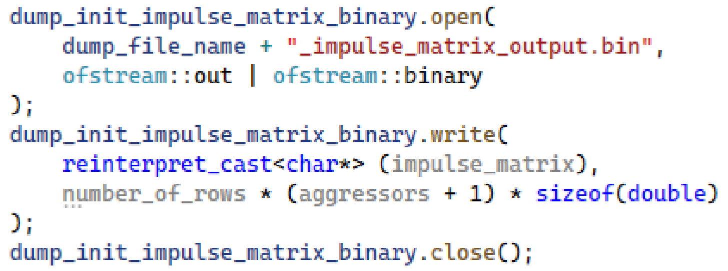
Implementation of AMI DLL hook

- Implementation of an AMI DLL hook is divided into two major approaches:
  - API hooking
  - DLL injection
- API hooking programming
  - Purpose is to create a DLL capable of performing "API hooking"
  - Six necessary steps:

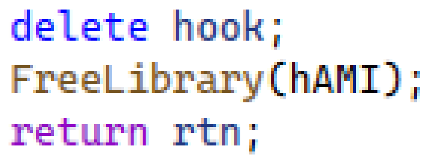
1). Export AMI standard API functions
2). Load original model
3). Obtain original API function addresses
4). Call the original API function
5). Dump and modify
6). AMI executable model releasing



```
DWORD AMI_hook::AMI_Impulse(
    double* impulse_matrix,
    LPSTR BCI_parameters_in,
    LPSTR* BCI_parameters_out,
    LPSTR* AMI_parameters_out
){
    return this->pFN_AMI_Impulse_Func(
        impulse_matrix,
        BCI_parameters_in,
        BCI_parameters_out,
        AMI_parameters_out,
        this->AMI_memory
    );
}
```



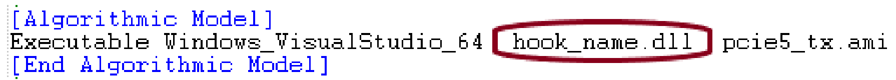
```
dump_init_impulse_matrix_binary.open(
    dump_file_name + "_impulse_matrix_output.bin",
    ofstream::out | ofstream::binary
);
dump_init_impulse_matrix_binary.write(
    reinterpret_cast<char*> (impulse_matrix),
    number_of_rows * (aggressors + 1) * sizeof(double)
);
dump_init_impulse_matrix_binary.close();
```



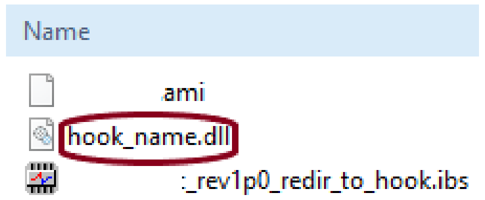
```
delete hook;
FreeLibrary(hAMI);
return rtn;
```

Implementation of AMI DLL hook

- Implementation of AMI DLL hook is divided into two major approaches:
  - API hooking
  - DLL injection
- DLL injection
  - Purpose is to cheat EDA so that the APIs will be called from the hook
  - Change the [Algorithmic Model] keyword declaration in IBIS model. Then put the hook together with the *.ibs, *.dll and *.ami file.



```
[Algorithmic Model]
Executable Windows_VisualStudio_64  hook_name.dll  pcie5_tx.ami
[End Algorithmic Model]
```



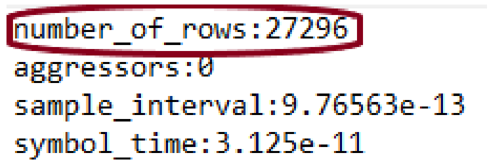
```
Name
            ami
hook_name.dll
            _rev1p0_redir_to_hook.ibs
```

Examples and Use Cases:
AMI DLL hook in SerDes AMI model alternative EDA enabling project

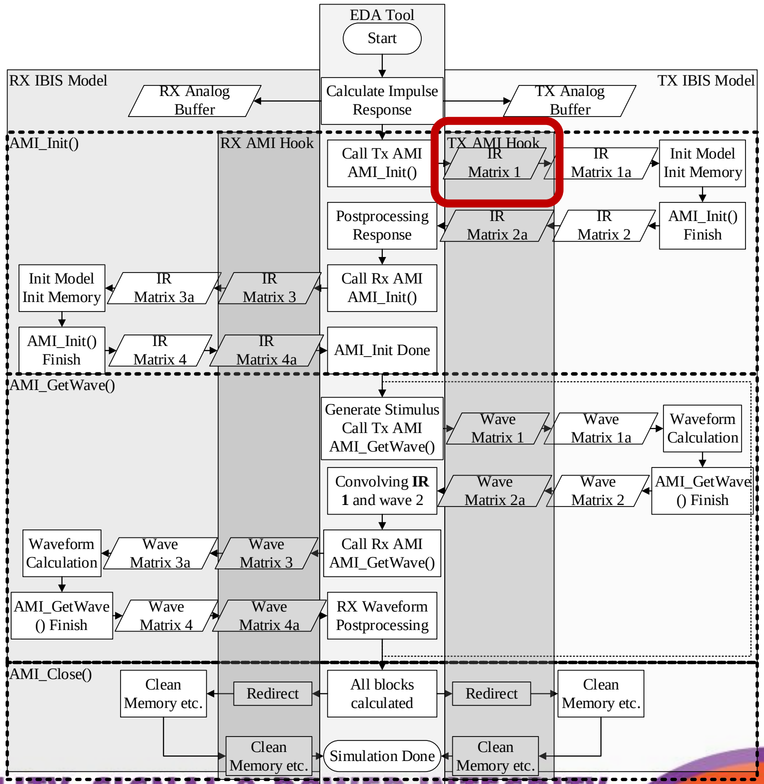
- Practice in SerDes AMI model alternative EDA enabling project:
  - Different results were obtained from two different EDA tools before implementing hooks.
- Found **5 observations impacting results in the first week** of implementing hook.
- 3 observations are user setting misalignments.

Examples and Use Cases:
AMI DLL hook in SerDes AMI model alternative EDA enabling project

- Observation 1: impulse response truncation
  - Found in the first breakpoint by unexpected "number_of_rows" dump.
  - Impulse response passed to AMI model is shorter than the matrix obtained from the software output.
- Solution: Truncate input impulse response matrix to the same in all EDA tools

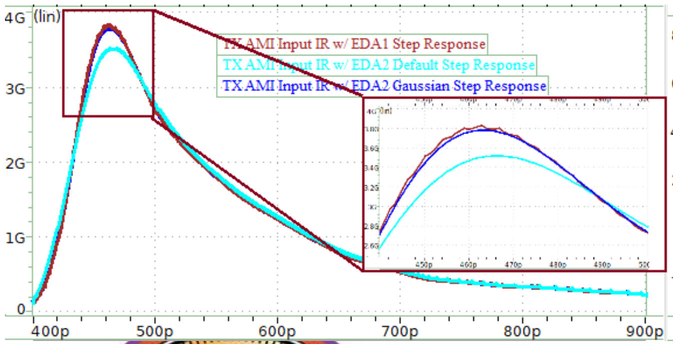


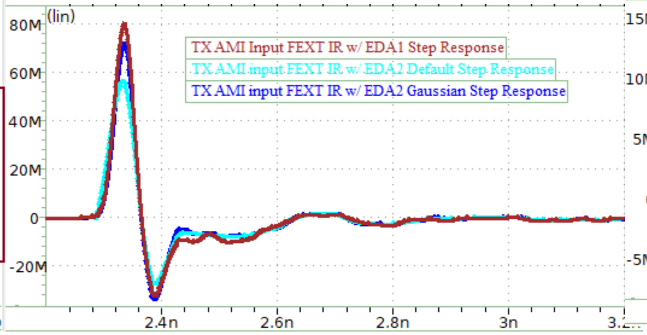
```
number_of_rows:27296
aggressors:0
sample_interval:9.76563e-13
symbol_time:3.125e-11
```

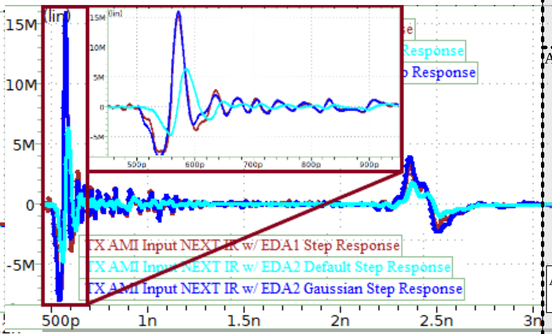


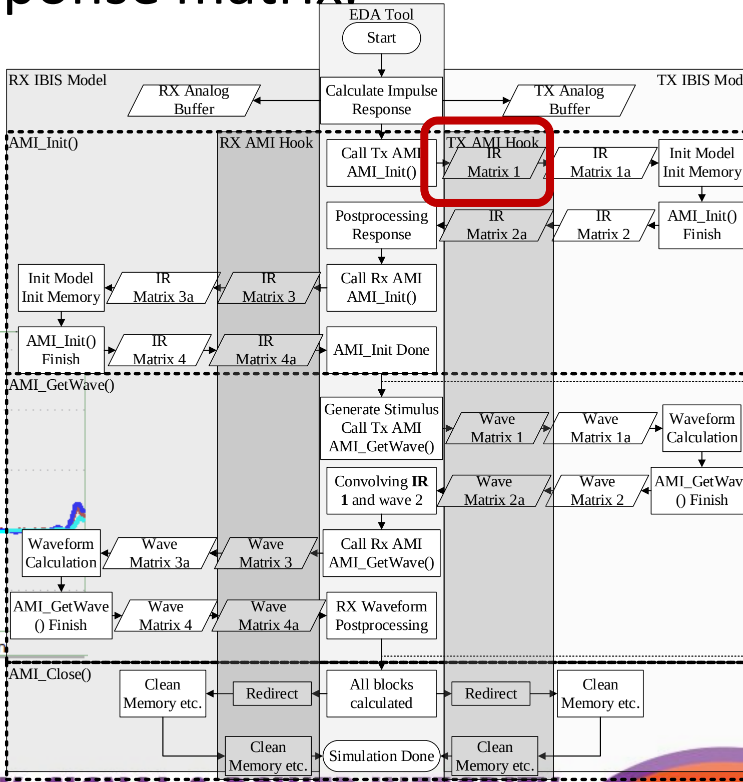
Examples and Use Cases:
AMI DLL hook in SerDes AMI model alternative EDA enabling project

- Observation 2: rising edge shaping difference
  - Found in the first breakpoint by comparing dumped impulse response matrix.
  - Rising edge shaping methods are different in EDAs:
    - one is linear edge, another is gaussian edge.
- Solution: Force one EDA to provide gaussian edge



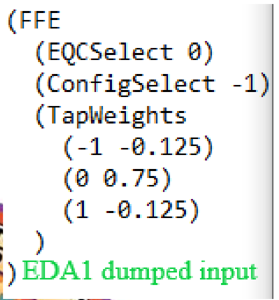






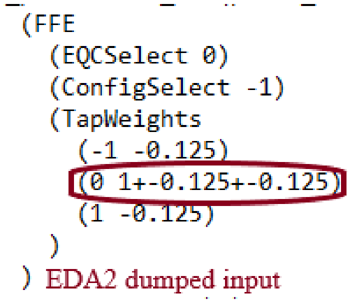
Examples and Use Cases:
AMI DLL hook in SerDes AMI model alternative EDA enabling project

- Observation 3: AMI input parameters formatting issue
  - Found in the first breakpoint by unexpected "TapWeights" keyword dump.
  - One of the EDA tools passes this parameter "as-is" while the input we provided is a formula.
- Solution: Provide numbers instead of formulas to EDAs



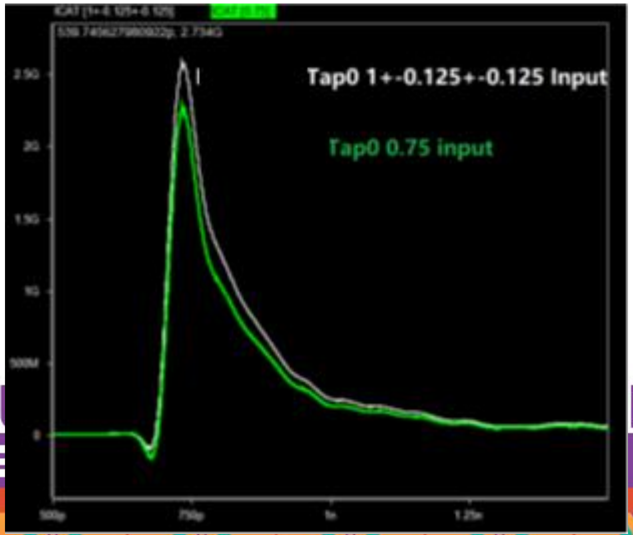
```
(FFE
  (EQCSelect 0)
  (ConfigSelect -1)
  (TapWeights
    (-1 -0.125)
    (0 0.75)
    (1 -0.125)
  )
)
```

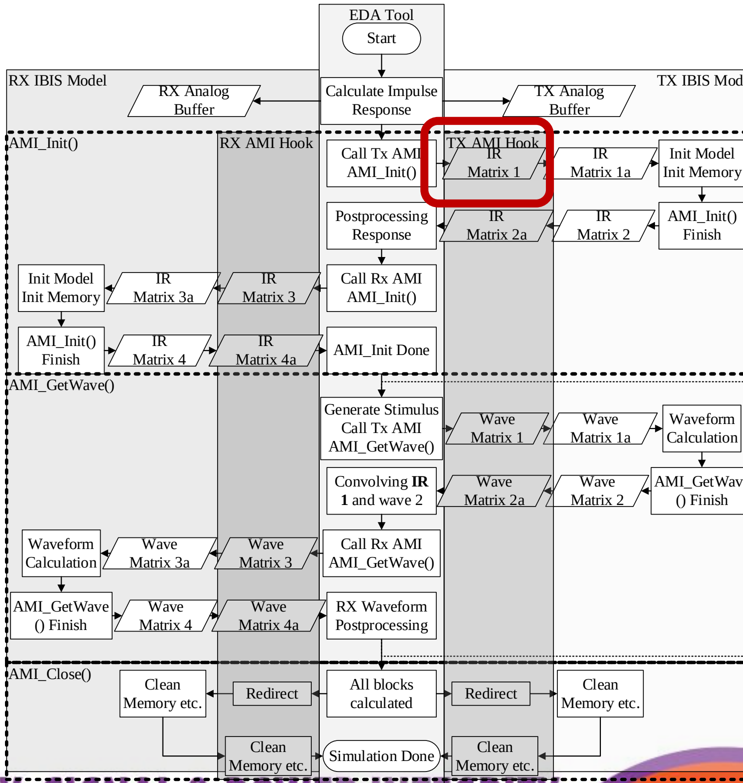
EDA1 dumped input



```
(FFE
  (EQCSelect 0)
  (ConfigSelect -1)
  (TapWeights
    (-1 -0.125)
    (0 1+-0.125+-0.125)
    (1 -0.125)
  )
)
```

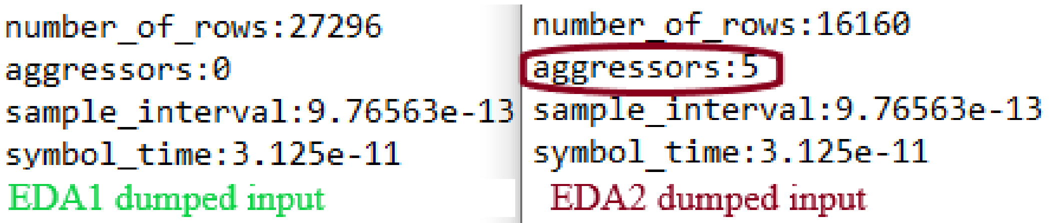
EDA2 dumped input





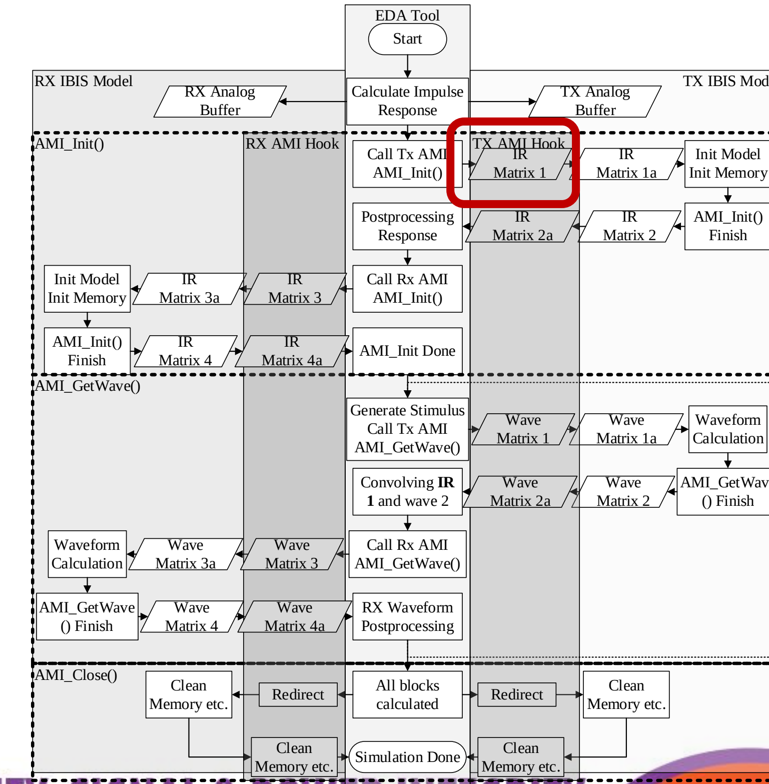
Examples and Use Cases:
AMI DLL hook in SerDes AMI model alternative EDA enabling project

- Observation 4: Different crosstalk handling methods
  - Found in the first breakpoint by different "aggressors" dump.
  - The two EDA tools are handling crosstalk by different methods.
- Solution: We understand this difference and introduced single line simulation comparison as well.



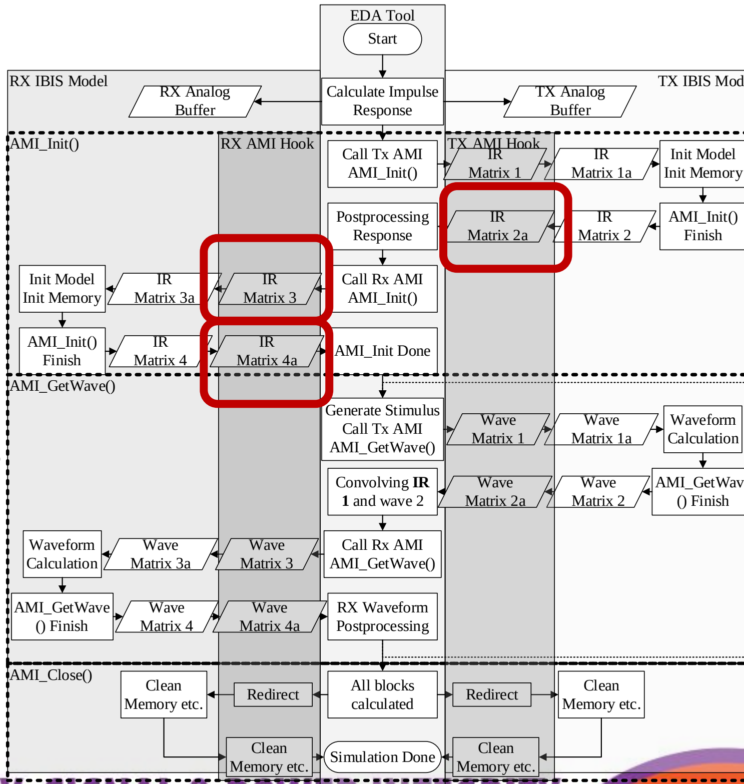
```
number_of_rows:27296              number_of_rows:16160
aggressors:0                      aggressors:5
sample_interval:9.76563e-13       sample_interval:9.76563e-13
symbol_time:3.125e-11             symbol_time:3.125e-11
```

EDA1 dumped input   EDA2 dumped input



Examples and Use Cases:
AMI DLL hook in SerDes AMI model alternative EDA enabling project

- Observation 5: AMI_Init() modified curves were impacting transient simulation results in one of the EDAs
  - Found in the 2nd/3rd/4th breakpoints by altering the return impulse response matrix and compare the results
  - This behaviour is not expected as the transient calculation should only convolve the input impulse response matrix and ignore others
- Solution: We reached that EDA vendor and reported this problem.



Conclusion

- This paper proposed AMI DLL hooking as a novel debugging method for model users or validators to debug IBIS-AMI models across various EDA software.
- The implementation of the AMI DLL hook in the SerDes AMI model alternative EDA enabling project has provided evidence of its effectiveness and utility.
- The AMI DLL hook has allowed for the segmentation of the full channel simulation into several parts to provide additional breakpoints and to debug part by part.
- This effort could contribute to enhancing the consistency of AMI models in the industry.

Future Work

- We may create a "results consistency check tool" to sign off AMI executable models for model vendors by implementing hooks into vendor AMIs.
- We may also reshape it into a more user-friendly debug tool to help more engineers in the industry.

Thanks!

Chuanyu Li
Intel Corporation
chuanyu.li@intel.com

Sleiman Bou-Sleiman
Intel Corporation
sleiman.bou-sleiman@intel.com