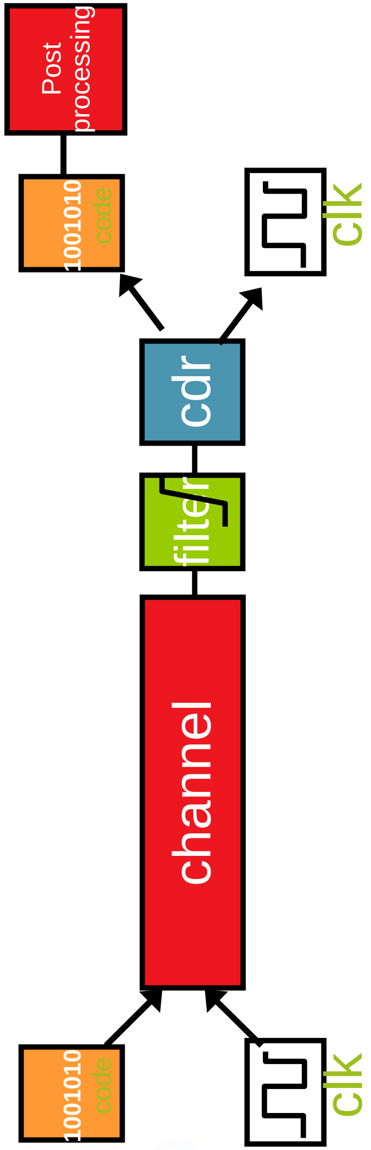
AMI Proposal Background Proposed changes to IBIS API Specifics

C. Kumar, Architect
Cadence

Nov 7, 2006

Advanced Serdes Modeling Challenges

- For 5+Gbps Serdes devices, complex signal processing ***algorithms*** often need to be represented, like:
  - FFE/DFE tap coefficient optimization (with/without crosstalk)
  - CDR algorithms
  - proprietary noise cancellation techniques
  - proprietary post-processing of data
- Architectural level exploration required
  - Algorithms are easy to represent and already exist at design level
  - They are typically modeled in higher level programming languages like C or Matlab
- These algorithms are very difficult to represent with traditional device modeling techniques
  - Long run times even if you can create them
- There is currently no industry-standard way to represent algorithms
  - IP suppliers have developed & distributed their own proprietary tools, increasing their support costs
  - No interoperability for systems company users

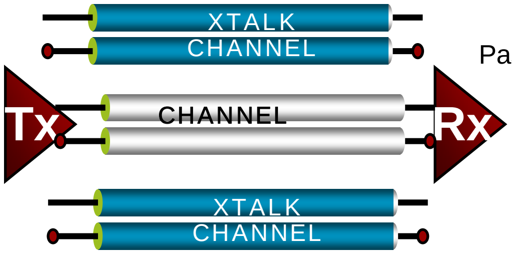


Simple API

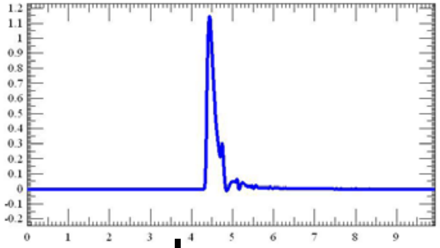
- Init
  - Initialize and optimize channel with Tx / Rx Model
  - This is where the IC DSP decides how to drive the system: e.g., filter coefficients, channel compensation, …
  - Input: Channel Characterization, system and dll specific parameters from config file
    - bit period, sampling intervals, # of forward/backward coefficients, …
  - Output: Modified Channel Characterization, status
- GetWave
  - Modify continuous time domain waveform [CDR, Post Processing]
  - Input: Voltage at Rx input at specific times
  - Output: Modified Voltage, Clock tics, status
- Close
  - Clean up, exit

Parameters passed by the system simulation platform are in red

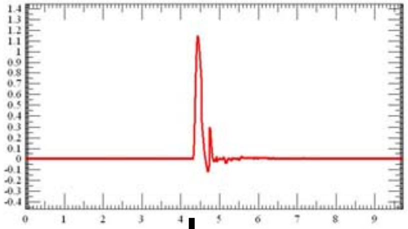
AMI_init



Pass characterization in matrix ‘a’



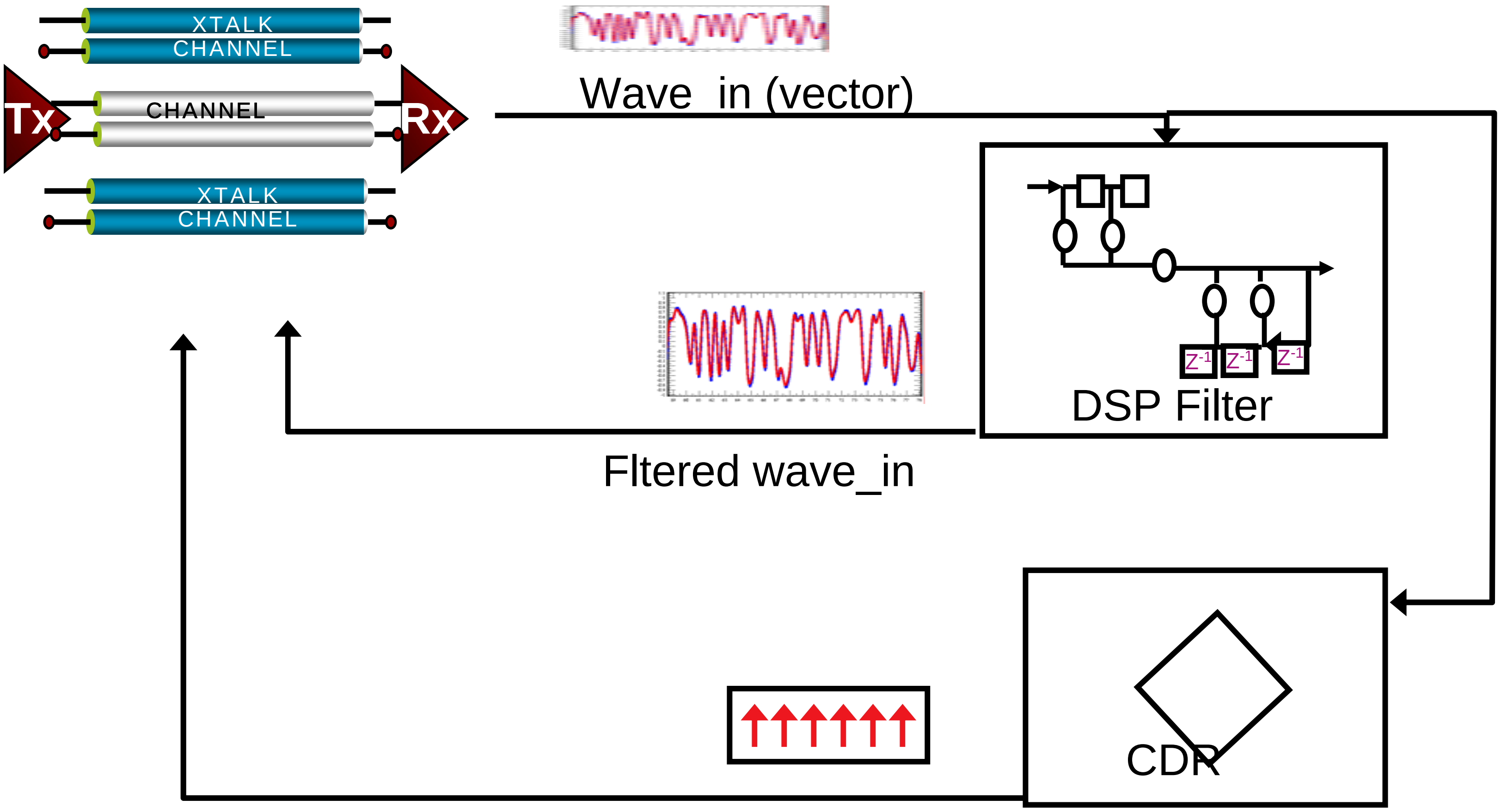
Dsp algorithms , modify characterization



Send modified char back (modify matrix ‘a’)

Internal storage

AMI_getwave



Wave in (vector)

DSP Filter

Fltered wave_in

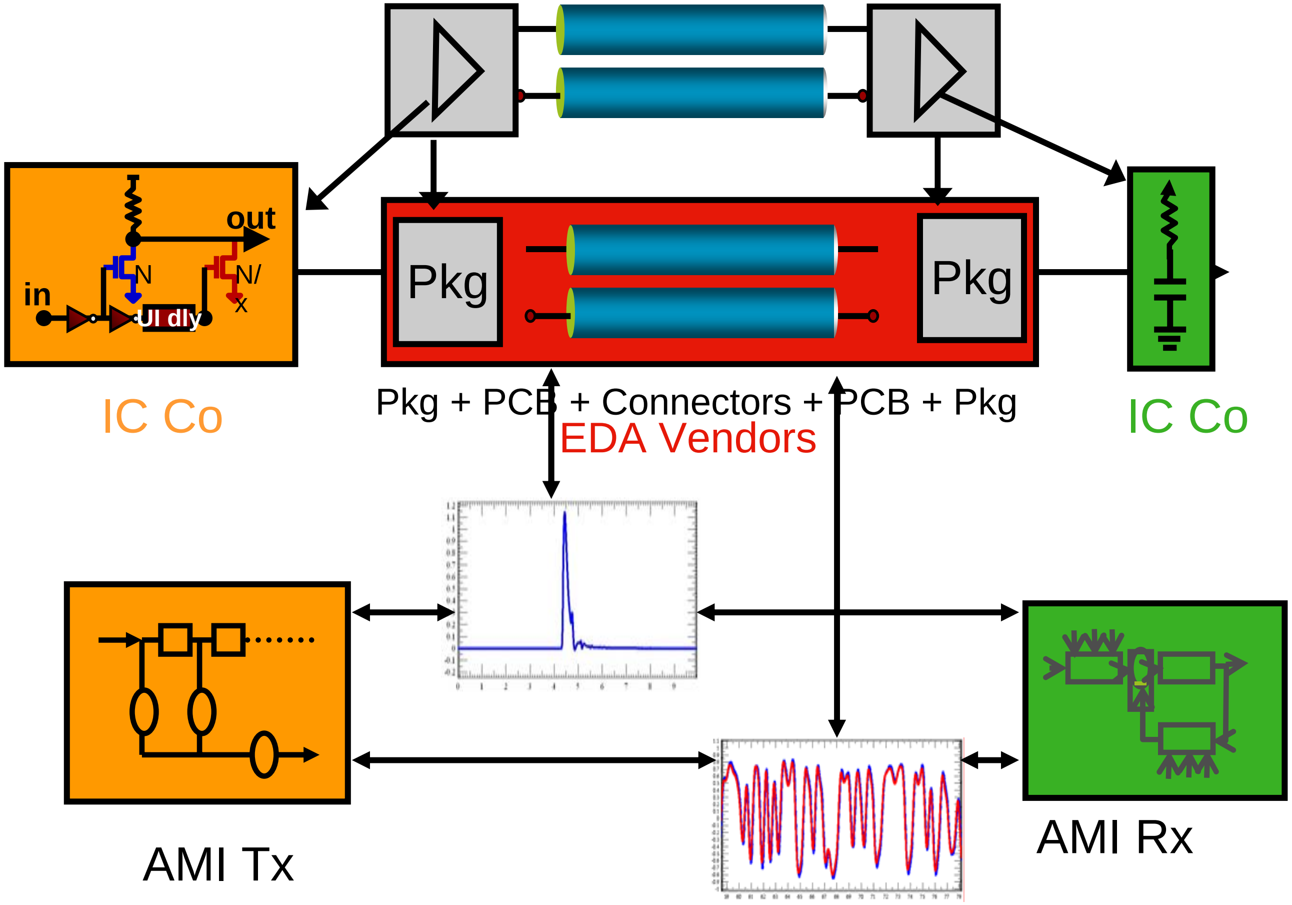
CDR

Clock vector

Proposal is evolutionary

- Leverages existing infrastructure
- Tx front end + channel + rx front end impulse characterization can be done using existing infrastructure

Evolutionary platform



IC Co

Pkg + PCB + Connectors + PCB + Pkg

EDA Vendors

IC Co

AMI Tx

AMI Rx

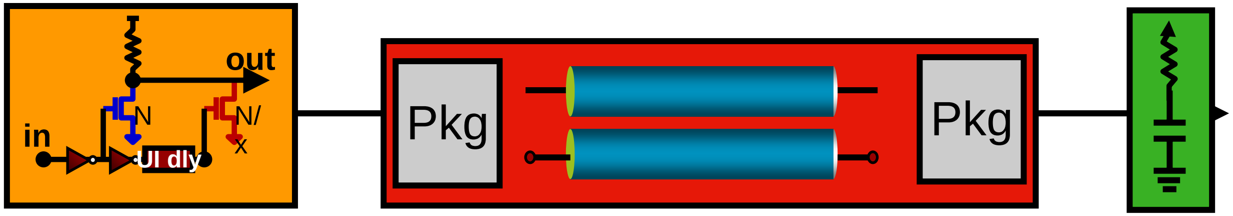
Example – query by Walter Katz – Sisoft

Channel Definition
A channel consists of a differential Tx driver, differential Rx receiver,
and a channel consisting of a 20 inch 100 Ohm differential transmission line.
The channel will be operating at 10Gbps (SymbolTime = 100ps).
The Tx driver is a 50 ohm driver, 1pF, 4 taps, rise and fall time=20ps, vol=0V, voh=1V.
The four taps are controlled by four “registers”,
each can have a value between 0 and 1,
but the sum of the absolute values must be <=1.
The Rx receiver’s electrical model is 1Meg ohm, 1pF.

Questions on this channel
How do I represent the electrical characteristics of the Rx and Tx model in IBIS?
What is the “Channel Characterization”?
What are the Tx and Rx API entry points?
What is the process for determining the 4 tap settings of the Tx?
For a given stimulus, how do I generate a “Voltage at Rx”?
Are time domain waveforms in the form of pairs of (Voltage,Time),
or are they at fixed time points (e.g. SymbolTime/10)?

Case 1: 4 tap Tx completely modeled in ibis

- 4 tap Tx completely described by IBIS
- User fixed tap coefficients
- Rx consists of IBIS load model + algorithm filter model

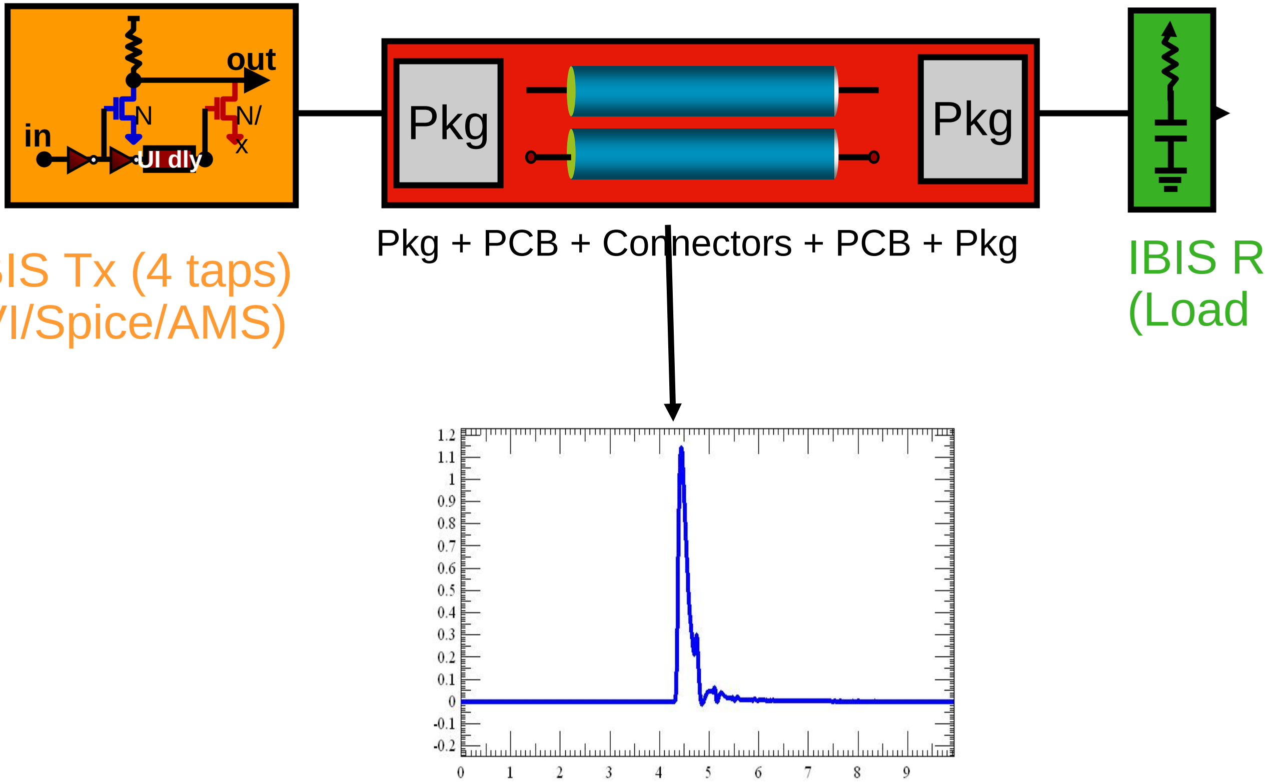


IBIS Tx (4 taps)
(VI/Spice/AMS)

Pkg + PCB + Connectors + PCB + Pkg

IBIS RX
(Load Model Only)

Case 1: AMI_Init – circuit characterization



IBIS Tx (4 taps)
(VI/Spice/AMS)

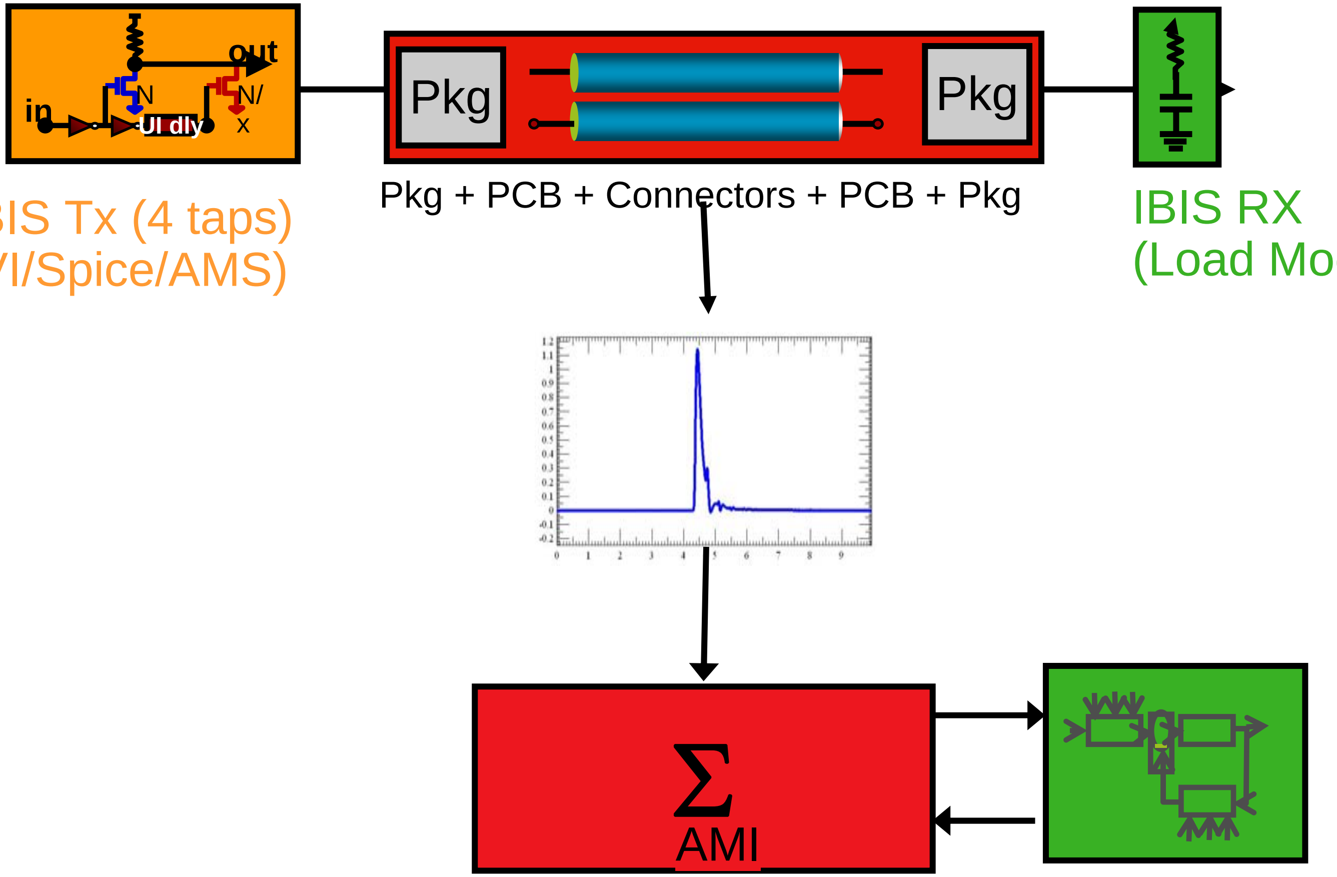
Pkg + PCB + Connectors + PCB + Pkg

IBIS RX
(Load Model Only)

-Generate complete Tx-Rx path characterization using ibis

-Characterization will include Tx tap settings

Case 1: AMI_Init – AMI initialization



IBIS Tx (4 taps)
(VI/Spice/AMS)

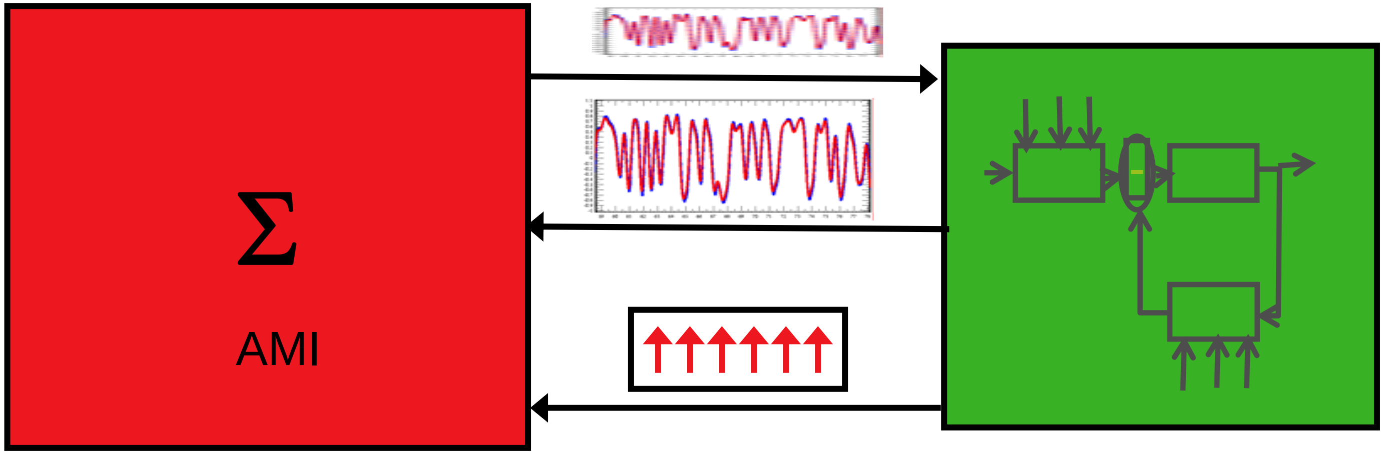
Pkg + PCB + Connectors + PCB + Pkg

IBIS RX
(Load Model Only)

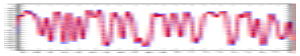
AMI Rx

-AMI Rx may or may not modify the characterization

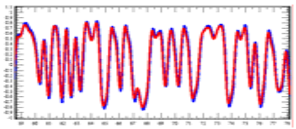
Case 1: AMI_GetWave



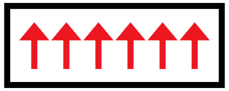
AMI Rx



Wave form input



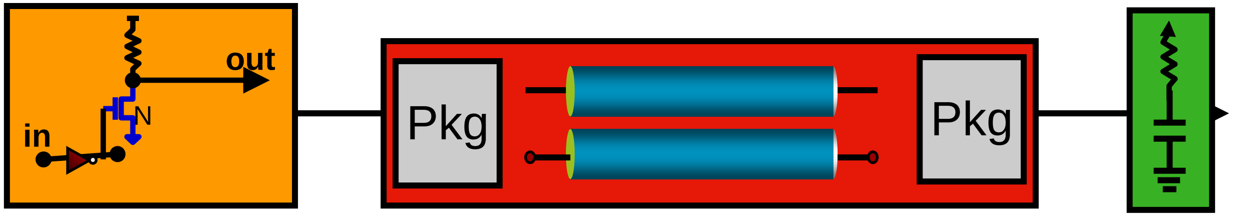
Filtered waveform from AMI Rx



Clocks from AMI Rx

Case 2: 4tap Tx and Rx are algorithm models

- IBIS contains only front end of Tx and Rx
- Both Tx and Rx have algorithm filter models
- Tap/filter coefficients may be automatic and/or user supplied as controlled my parameter inputs to the respective algorithm models

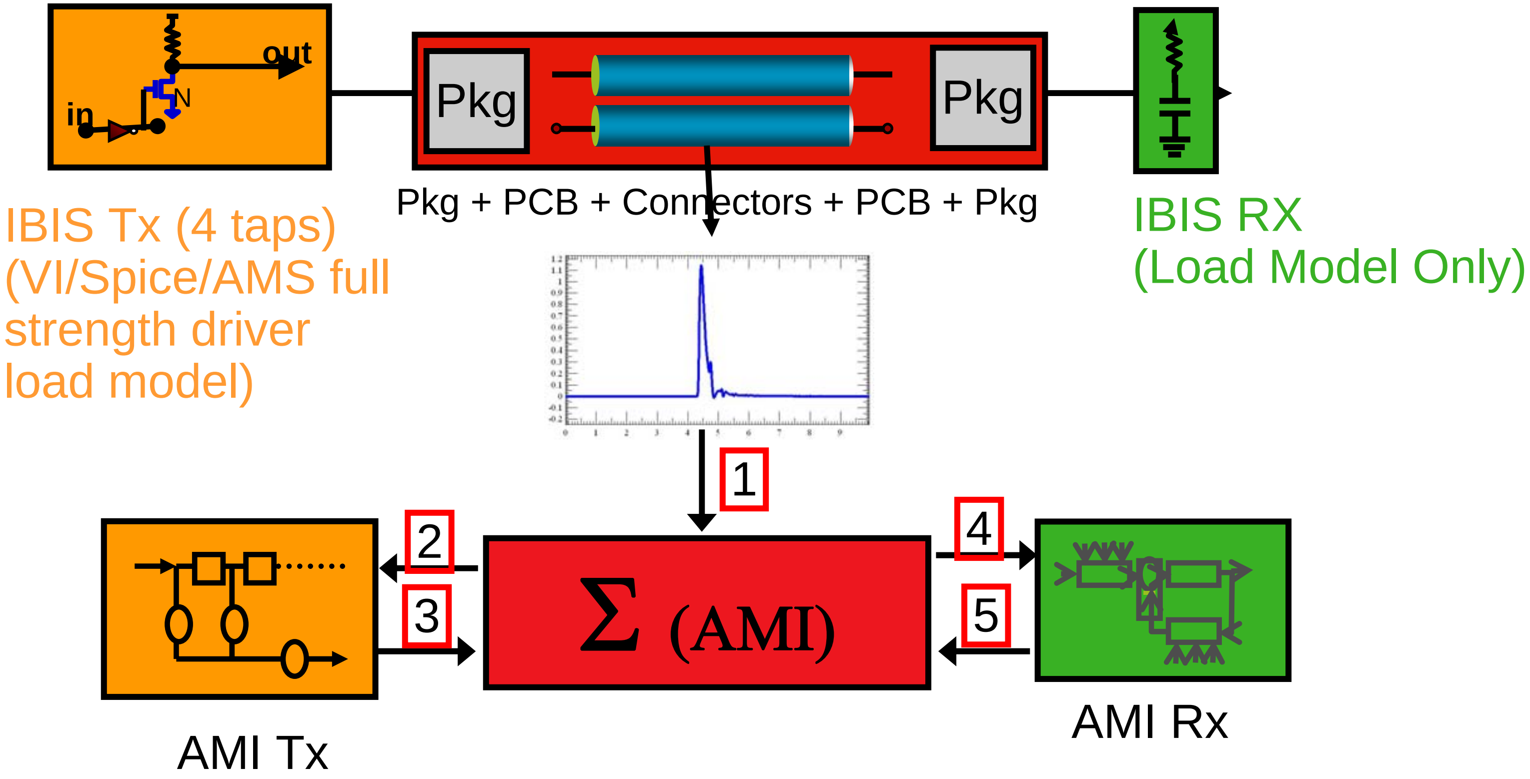


Pkg + PCB + Connectors + PCB + Pkg

IBIS Tx (4 taps)
(VI/Spice/AMS full strength driver load model)

IBIS RX
(Load Model Only)

Case 2: AMI_Init



Pkg + PCB + Connectors + PCB + Pkg

IBIS Tx (4 taps)
(VI/Spice/AMS full
strength driver
load model)

IBIS RX
(Load Model Only)

AMI Tx

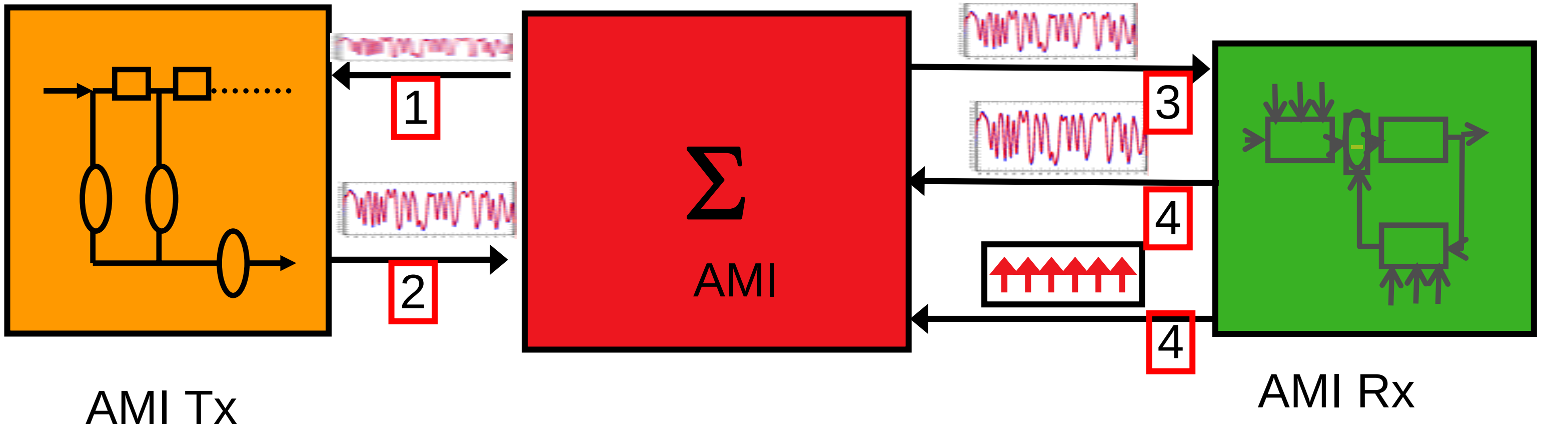
AMI Rx

1 Circuit characterization passed to EDA AMI platform

2 3 Characterization sent/received from AMI Tx

4 5 Characterization sent/received from AMI Rx

Case 2: AMI_getwave



1 EDA AMI platform sends wave form to Tx

2 Tx sends back modified waveform

3 EDA AMI platform sends wave form from step 2 to Rx

4 Rx sends back waveform and clock information

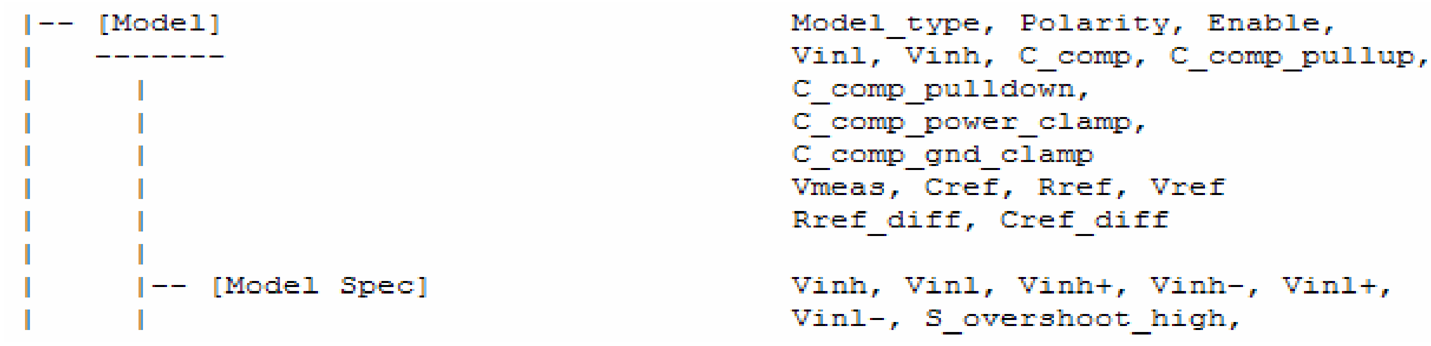
Special Case – only AMI_Init utilized

- It is possible that AMI models may use just AMI_Init to modify the characterization
- AMI_getwave my do nothing
- Only linear channel filtering is modeled and there will be no clock and data recovery information
- Such models may be used for channel compliance testing and early architectural exploration

Proposed changes to IBIS

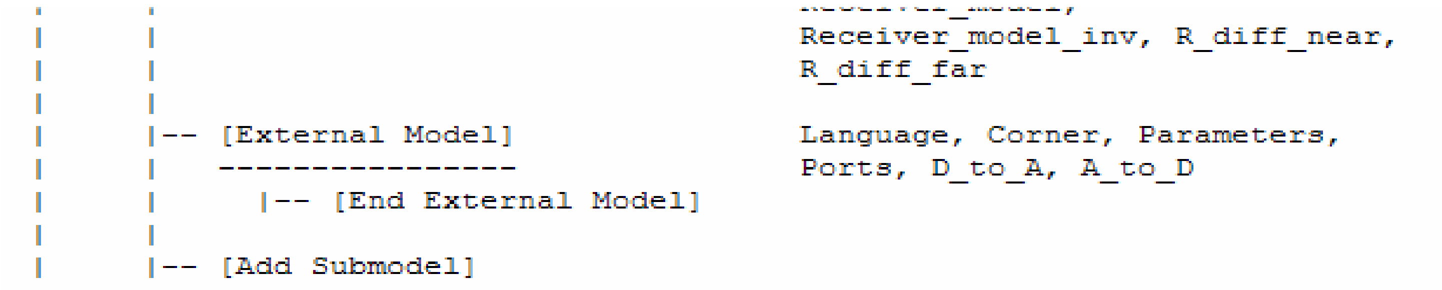
- Introduce a new section (“AMI”) with a unique name that is parallel to External Model construct
- AMI section sits on top of and leverages the circuit simulation infrastructure
  - Algorithmic model requires existing IBIS structure to represent the Tx and Rx load models
  - These Tx, Rx models along with the channel constitute a Linear Time Invariant (LTI) system
- AMI section introduces
  - Three API calls: AMI_Init, AMI_GetWave, AMI_Close
  - Each call provides a means for model developer to pass algorithmic model specific parameters: # of filter taps, filter tap spacing, etc
    - Model developer provides documentation on parameters to model consumer
  - An AMI section can have multiple algorithmic models: for example one for Amplifier (eye opener) and another for DFE/CDR
    - Simulation platform expected to call each AMI section in the order it appears in the AMI section

Syntax Structure



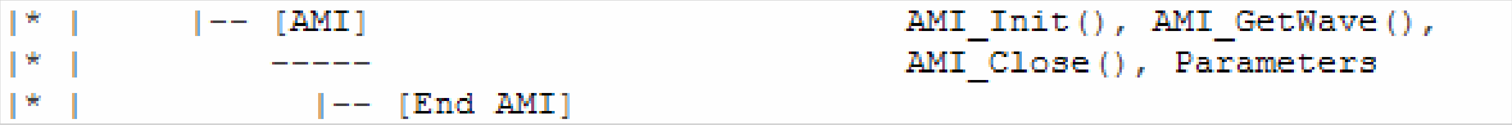
```
|-- [Model]                                          Model_type, Polarity, Enable,
|    --------                                        Vinl, Vinh, C_comp, C_comp_pullup,
|       |                                            C_comp_pulldown,
|       |                                            C_comp_power_clamp,
|       |                                            C_comp_gnd_clamp
|       |                                            Vmeas, Cref, Rref, Vref
|       |                                            Rref_diff, Cref_diff
|       |
|       |-- [Model Spec]                             Vinh, Vinl, Vinh+, Vinh-, Vinl+,
|       |                                            Vinl-, S_overshoot_high,
```

~ ~ ~



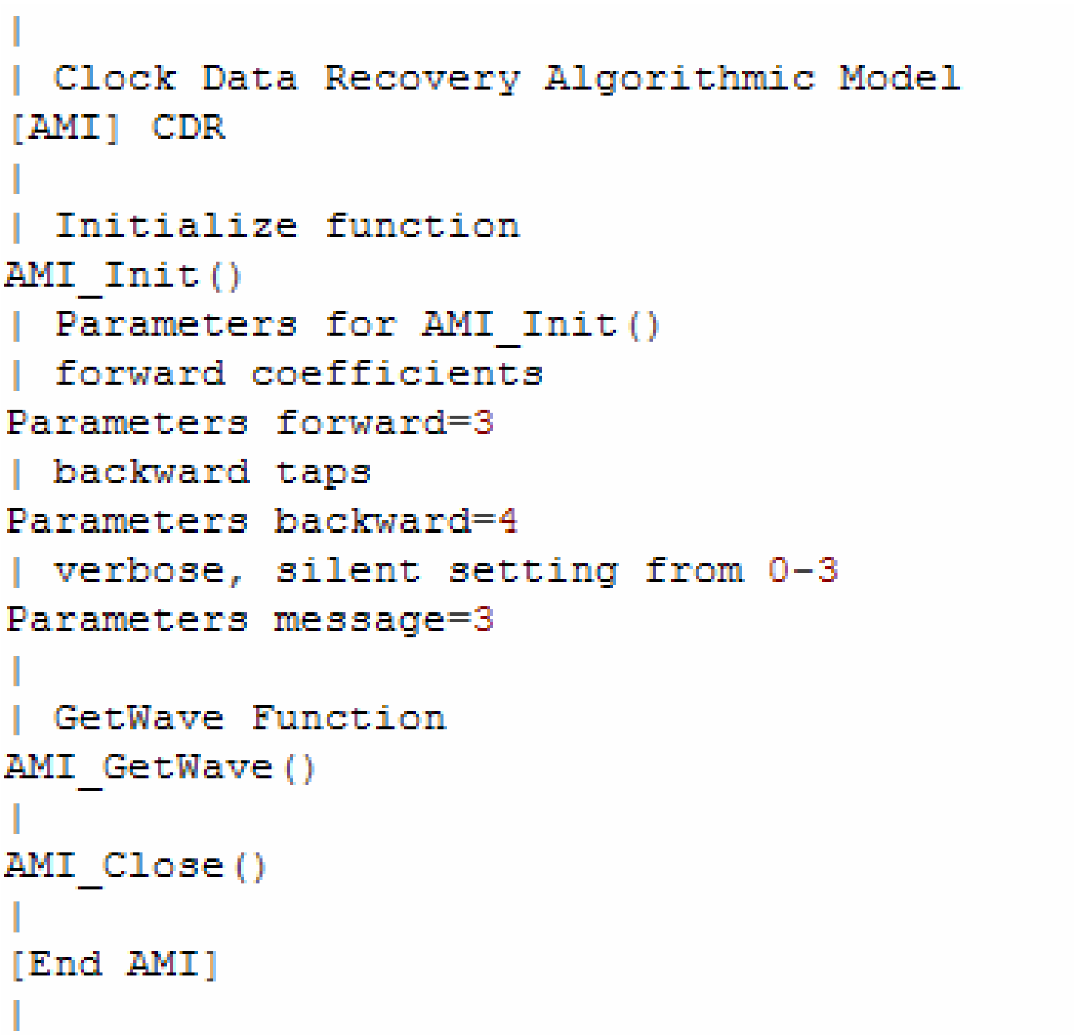
```
|       |                                            Receiver_model_inv, R_diff_near,
|       |                                            R_diff_far
|       |
|       |-- [External Model]                         Language, Corner, Parameters,
|       |   ----------------                         Ports, D_to_A, A_to_D
|       |      |-- [End External Model]
|       |
|       |-- [Add Submodel]
```

Add [AMI] here



```
|* |      |-- [AMI]                    AMI_Init(), AMI_GetWave(),
|* |          -----                    AMI_Close(), Parameters
|* |             |-- [End AMI]
```

[AMI] Syntax



```
|
| Clock Data Recovery Algorithmic Model
[AMI] CDR
|
| Initialize function
AMI_Init()
| Parameters for AMI_Init()
| forward coefficients
Parameters forward=3
| backward taps
Parameters backward=4
| verbose, silent setting from 0-3
Parameters message=3
|
| GetWave Function
AMI_GetWave()
|
AMI_Close()
|
[End AMI]
|
```

API Call Params

- long AMI_***init*** (double *a, long row_size, long col_size, double bitp, double tr, double tf, void **pdll_server_param_obj, void *dll_client_param, char *dllcontrol, [genchdllmsg_type **msg])
  - Input: Channel Characterization, system and dll specific parameters from config file
    - bit period, sampling intervals, # of forward/backward coefficients, …
  - Output: Modified Channel Characterization, status

- long AMI_***getwave*** (double *wave_in, long size, double dt, double *clk, void *dll_server_param_obj, void *dll_client_param, [genchdllmsg_type **msg])
  - Input: Voltage at Rx input at specific times
  - Output: Modified Voltage, Clock tics (dll specific), status

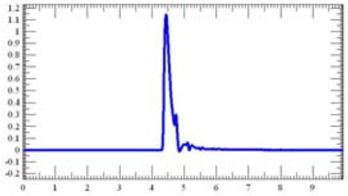
- long AMI_***close*** (void *dll_server_param_obj)
  - Clean up, exit

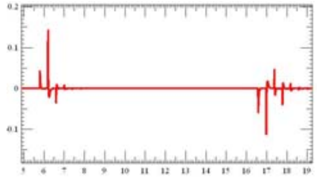
AMI_init

- long AMI_**init** (**double *a, long row_size, long col_size**, double bitp, double tr, double tf, void **pdll_server_param_obj, void *dll_client_param, char *dllcontrol, [genchdllmsg_type **msg])

  ‘a’ – impulse response matrix

Time (s)	Primary Channel (v)	Crosstalk1	Crosstalk2





Index = j * row_size + i

AMI_init - input/output

Inputs:

Impulse response matrix ‘a’

Configuration Parameters ‘dllcontrols’

Returns:

0 – Failure

1 – success

Other Actions:

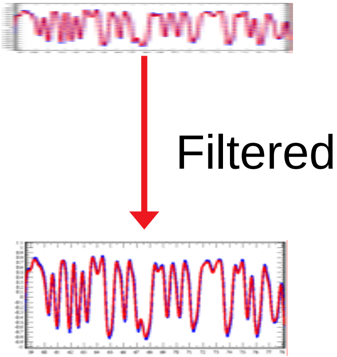
The input impulse matrix ‘a’ may be filtered and modified

Local data space may be returned through ‘pdll_server_param_obj’

AMI_getwave

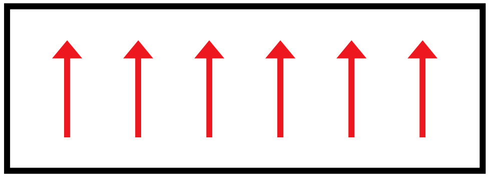
- long AMI_***getwave*** (**double *wave_in**, long size, double **dt**, double ***clk**, void *dll_server_param_obj, void *dll_client_param, [genchdllmsg_type **msg])

double *wave_in: is a pointer to a one dimensional Waveform Vector of length ‘size’



double dt is sample spacing

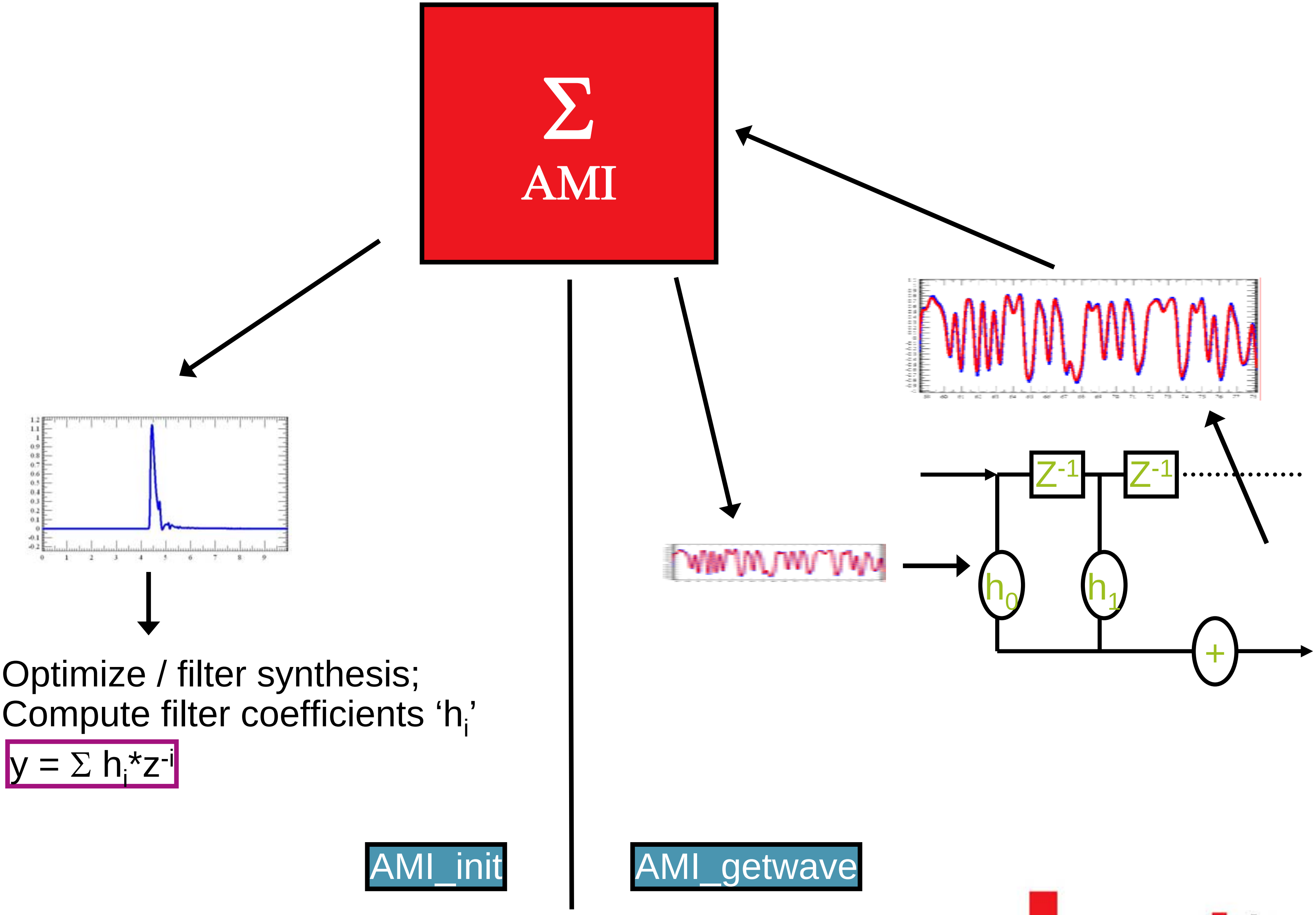
Double *clk is a pointer to a vector of same size as wave_in; provided by the model to the simulation platform



FFE filter sample – chffefilt dll

- Ami_init call
  - Receive the channel impulse response and filter configuration parameters (*a)
  - Optimize the filter coefficients
  - Store the coefficients for later use during ami_getwave call
  - Create a call back object (**pdll_server_param_obj)
  - (Note: in this example, AMI_init does not modify *a)
- AMI_getwave
  - Receive the input wave form (*wave_in)
  - Apply the filter to input wave_form
  - Write back the filtered wave form (*wave_in)

FFE filter



Σ AMI

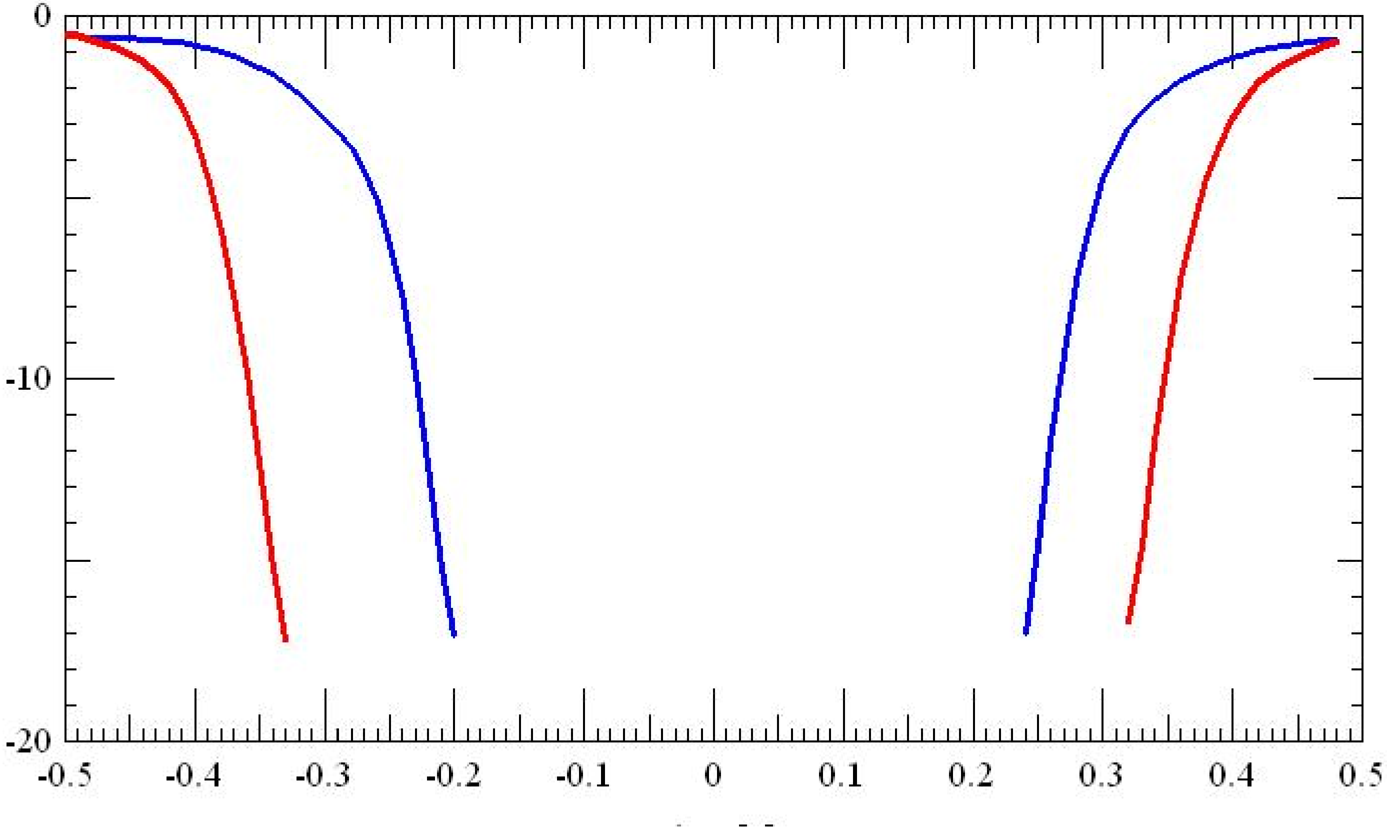
Optimize / filter synthesis;
Compute filter coefficients ‘h_i’

$\boxed{y = \Sigma\, h_i * z^{-i}}$

AMI_init

AMI_getwave

FFE results



cadence™