INVENTIVE

cadence

SiSoft

Mentor Graphics

# Algorithmic Modeling BIRD - Update

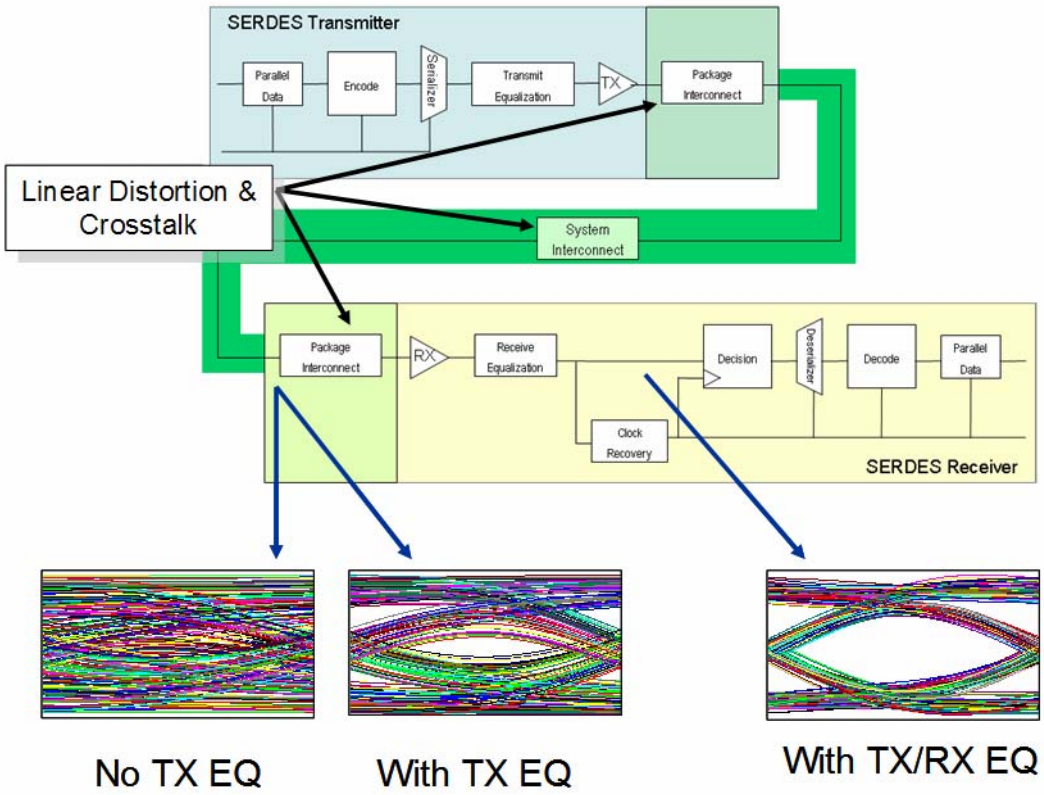
Cadence, SiSoft, Mentor

April 10th, 2007

# Agenda

- Serial Link Design Trends
- IBIS-ATM Goals
- Terminology
- Proposed IBIS File Structure & Syntax
- [Algorithmic Model] Parameter Example
- API Overview
- Delivery Mechanism
- Next Steps

# Overview - Trends in Serial Link Design



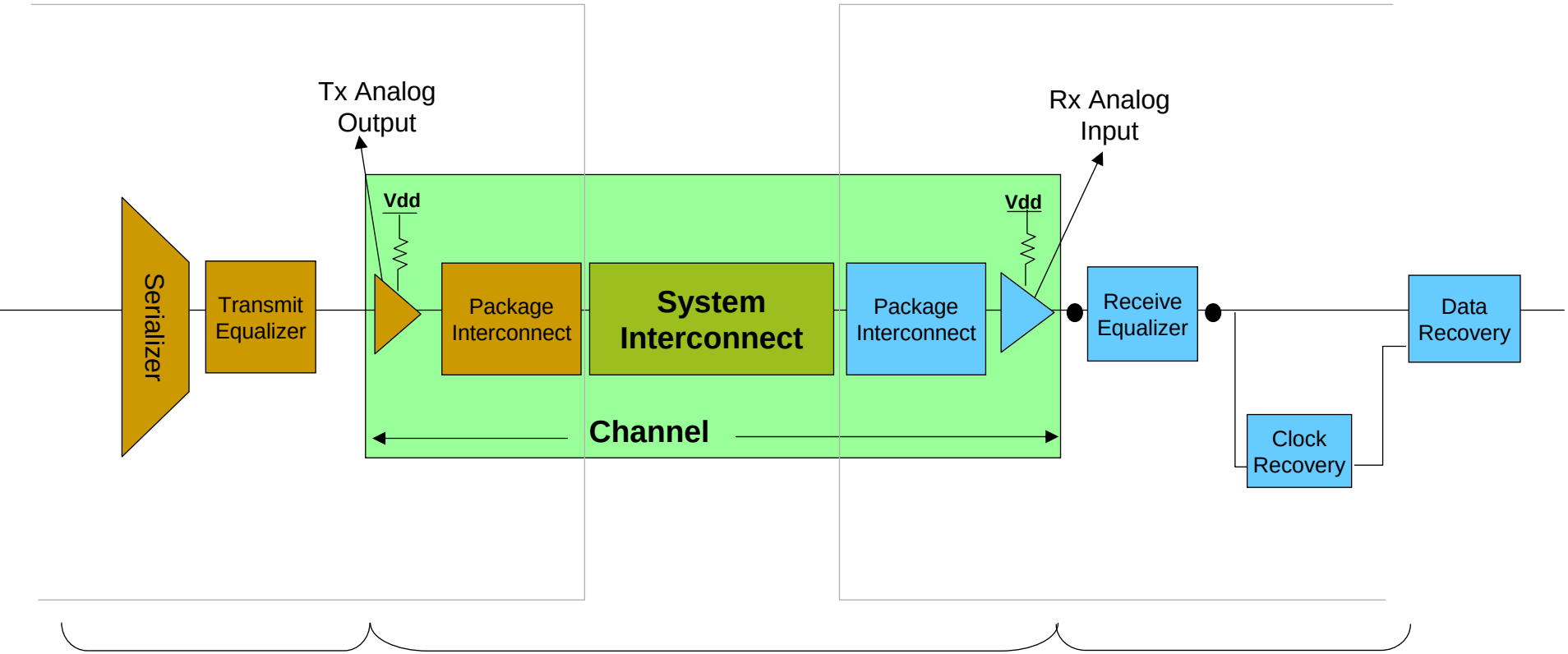

- Increasing speeds and interconnect loss are closing receiver eyes
  - Mask-based design is becoming impossible
- No modeling standards for SerDes devices above 5Gbps
  - Equalization
  - Optimization
  - Clock recovery
- EDA / SerDes IP interoperability is a key system design requirement

# IBIS-ATM Goals

- Establish a modeling standard for SerDes devices that describes
  - Analog I/O & electrical package characteristics
  - Transmit / receive equalization
  - Clock recovery behavior

  } IBIS 4.2
- The standard must:
  - Support prediction of link Bit Error Rate (BER)
  - Enable EDA & SerDes model interoperability
  - Protect Semiconductor vendor IP
  - Support design optimization

# Analytical Methodology

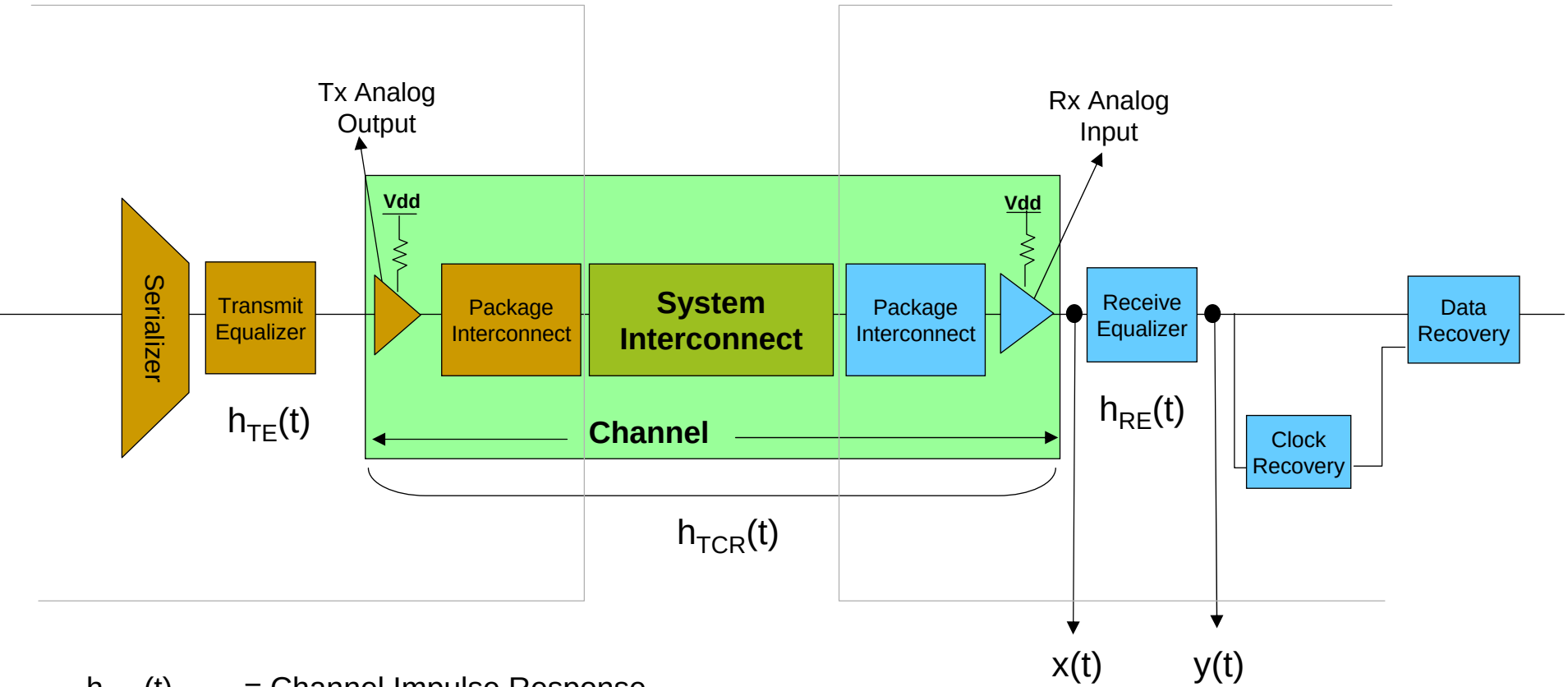



**TX EQ**
Signal processing

**Channel & Analog I/O**
Linear, Time-Invariant
Impulse response derived
from circuit simulation/analysis

**RX EQ, CDR**
Signal processing

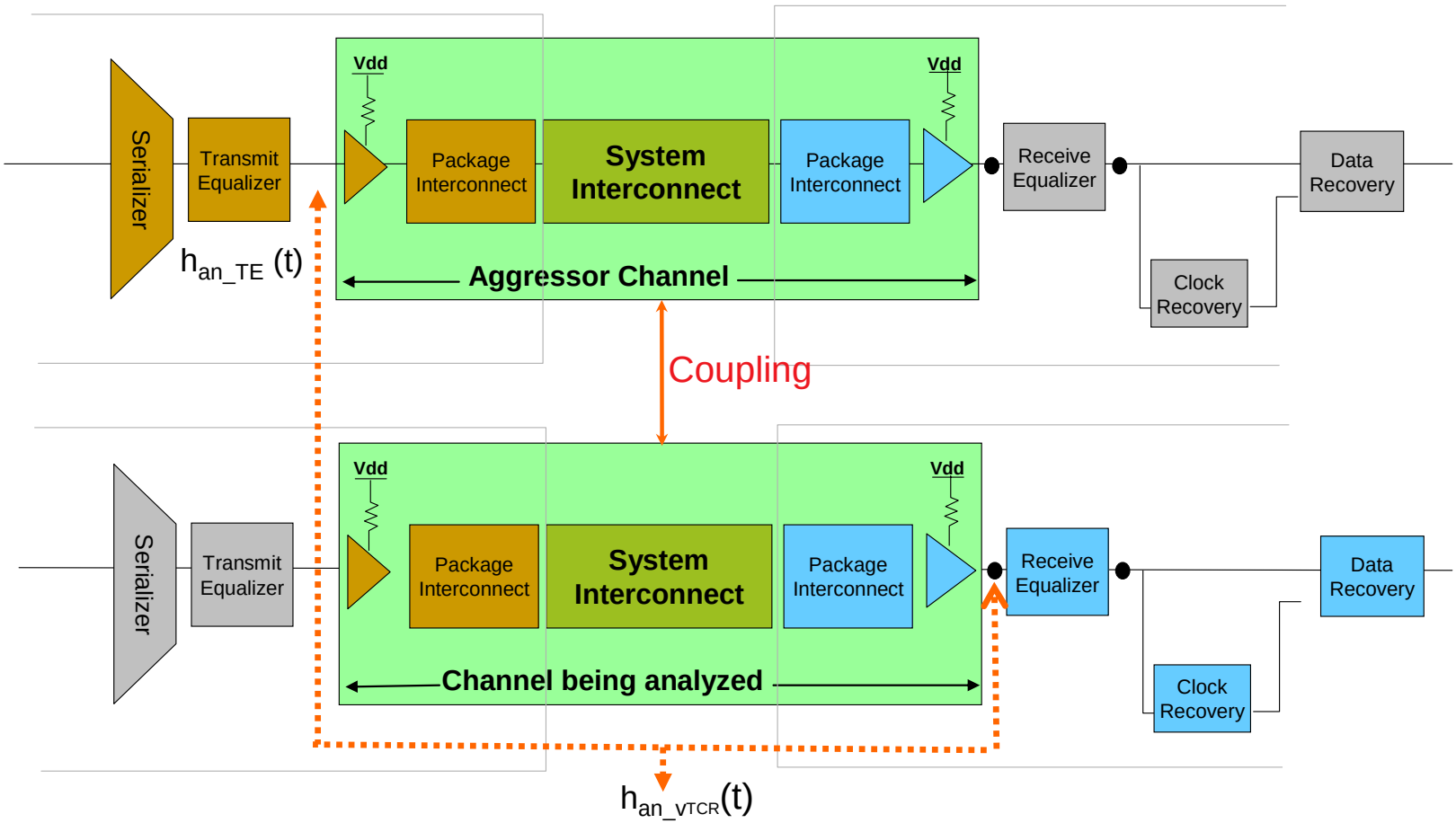
# Terminology




$h_{TCR}(t)$ = Channel Impulse Response
$h_{TE}(t)$ = Impulse Response of TX Equalization
$h_{RE}(t)$ = Impulse Response of RX Equalization
x(t) = Signal at receiver input pad
y(t) = Signal at receiver decision point

Channel description includes return paths etc which are not shown for the sake of clarity

# Crosstalk Terminology

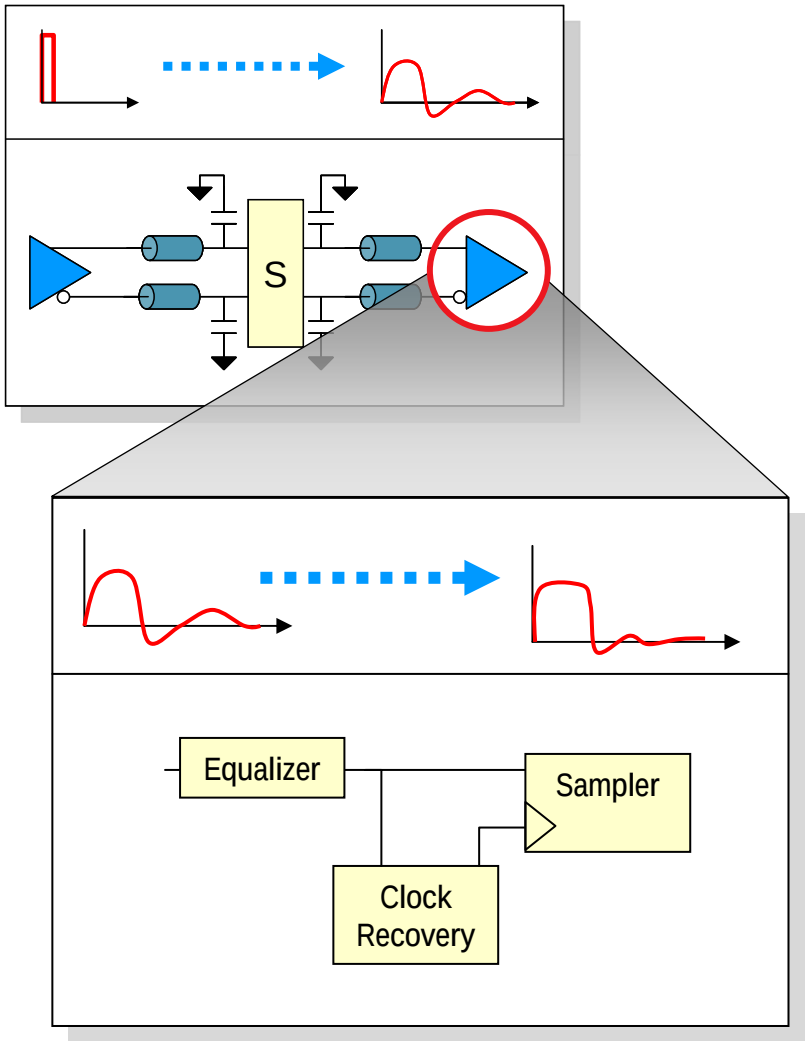



$h_{an\_TE}(t)$ = Impulse Response of Aggressor's TX Equalization

$h_{an\_vTCR}(t)$ = Impulse Response of the aggressor TX front end + coupled aggressor victim circuit + victim RX front end

# Proposed IBIS File Structure

Circuit Simulation




Signal Processing

[Component] MY_COMP

…

[Model] ModelName

Model_Type Output

- Existing IBIS buffer syntax

- [Algorithmic Model]

  …

- [End Algorithmic Model]

# [Algorithmic Model] Syntax

- [Algorithmic Model] section points to executable model code and what parameters are passed from/to the EDA platform
- Parameter definitions are model-specific and not part of the proposed IBIS standard
- Standard proposal covers parameter types (In, Out, Info) and data exchange formats

# [Algorithmic Model] Parameter Syntax

**<keyword name> <usage> <data type> <data format> <values>**

<usage>
- In Parameter is required Input to executable
- Out Parameter is Output only from executable
- Info Information for user or EDA platform

<data type> (default is float)
- float
- integer
- string

<data format> (default to range)
- range <typ value> <min value> <max value>
- list <typ value> <value> <value> <value> ... <value>
- corner <typ value> <slow value> <fast value>
- table #columns
- Gaussian <mean> <sigma>
- Dual-Dirac <mean> <mean> <sigma> | Composite of two Gaussian
- DjRj <minDj> <maxDj> <sigma> | Convolve Gaussian (sigma) with uniform Modulation PDF

# Assumptions for Example Tx and Rx Model

- Transmit Equalization is not LTI
  - End-user sets strength and tap coefficients directly
  - Tx.Init does not return equalized impulse response
  - Tx.Init requires Strength parameter be set; the model may optionally optimize and return updated tap settings

- Receive Equalization is not LTI
  - Rx.Init may optimize the device's peaking filter and tap settings
  - Rx.GetWave requires .5Meg bits to settle on final settings
  - Getwave model returns clock tick data
  - The EDA tool will analyze the Getwave output (waveform & clock ticks) along with the model's Differential_Offset to determine the BER.
  - The EDA tool is responsible for statistically post processing simulation data in order to extrapolate the error rate into the rarer numbers.

# Example IBIS Tx Algorithmic Model Section

[Model] Tx_SerDes
Model_type Output
[Voltage_Range] 1.2 1.1 1.3
…
|
[Algorithmic Model]
|
Algorithmic_Model_Type Tx_SerDes_NRZ
|
Executable Dll Windows tx_serdes.dll
|
Jitter info DjRj mks 0 3ps 2ps
Duty_cycle info range real ui .05 0. .1
Ignore_Bits info real ui 4.
Number_of_Aggressors info integer range 0 0 8
h_te info Boolean False
GetWave info Boolean True
|
Strength in range real 1. 0. 1.
|
Equalizer InOut Tap_Equalizer
Tap_Type ffe
Taps 4
Tap -1 real increment 0. –1. 1. 32
Tap 0 real increment 1. –1. 1. 32
Tap 1 real increment 0. –1. 1. 32
Tap 2 real increment 0. –.25 .25 8
Tap_Sum real range 1. 0. 1.
Tap_Spacing Synch
End_Tap_Equalizer
|
[End Algorithmic Model]

# Example IBIS Rx Algorithmic Model Section

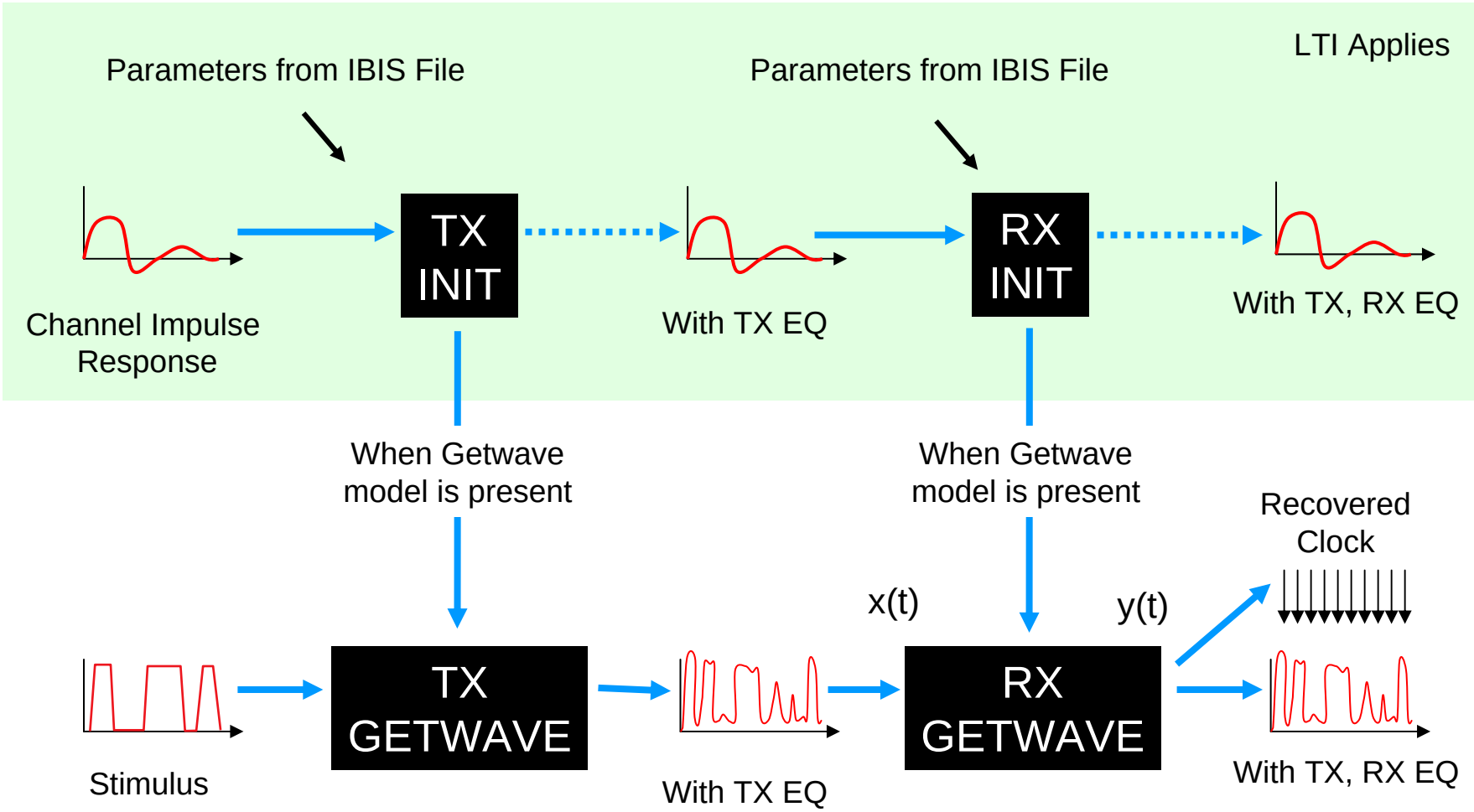
```
[Model] Rx_SerDes
Model_type Input
[Voltage_Range] 1.2 1.1 1.3
…
|
[Algorithmic Model]
|
Algorithmic_Model_Type Rx_SerDes_NRZ
|
Executable Dll Windows rx_serdes.dll
|
Differential_Offset info real 10mV
|      aka Vdiff, Voltage Offset, Receiver Sensitivity
|
Ignore_Bits info real ui .5e6
|
Number_of_Aggressors info integer range 0 0 8
|
h_re info Boolean False
GetWave info Boolean True
Clock_Ticks info Boolean True
|
[End Algorithmic Model]
```

# API Introduction

- Executable computer code models signal processing functions inside the SerDes transmitter / receiver

- **Init:** Impulse response processing
  - Initializes TX / RX model and optimizes device settings
    - Filter coefficients / channel compensation
  - Optionally returns modified input impulse response & equalization settings to EDA platform
- **GetWave:** Continuous time-domain waveform processing
  - Applies equalization & passes waveform to EDA platform
  - Recovers and passes “clock ticks” back to EDA platform
  - Performs any proprietary Post Processing

# IBIS-ATM Algorithmic Models



# Tx – Init

- Inputs
  - (Model & Simulation) Parameters
  - Impulse Response of Channel: $h_{TCR}(t)$
  - Impulse Response of Aggressors: $h_{An\_vTE}(t) \otimes h_{An\_vTCR}(t)$ (optional)
  - Sampling Interval
  - Bit time
  - Number of Aggressors
- Outputs
  - Parameters (optional)
  - Data for GetWave (optional)
  - Modified Channel Response $h_{TE}(t) \otimes h_{TCR}(t)$ (optional)
- DLL call:
  - Long AMI_Init (double *impulse_matrix, long row_size, long aggressors, double sample_interval, double bit_time, char *AMI_dll_parameters, void **AMI_dll_memory_handle, AMI_dll_ msg_type **msg)

# Tx – GetWave

- Inputs
  - Stimulus** waveform with logic values and edge times
  - Data for GetWave (output of TX_INIT)
  - Start of New Message flag
- Outputs
  - Stimulus $\otimes$ $h_{TE}(t)$ $\otimes$ $h_{TCR}(t)$
    - This is a continuous waveform with the sampling interval specified by the TX_INIT call
- DLL Call
  - Long AMI_GetWave (double *wave_in, long wave_size, double *clock_times, void *AMI_dll_memory)

**: Time Voltage Pairs
*Edge time* obtained from IBIS data

# Rx – Init

- Inputs
  - (Model & Simulation) Parameters
  - Impulse Response of Channel with TX EQ: $h_{TE}(t) \otimes h_{TCR}(t)$
  - Impulse Response of Aggressors: $h_{An\_vTE}(t)$ X $h_{An\_vTCR}(t)$ (optional)
  - Sampling Interval
  - Bit time
  - Number of Aggressors
- Outputs
  - Parameters (optional)
  - Data for GetWave (optional)
  - Modified Channel Impulse response (optional)
    - $h_{TE}(t) \otimes h_{TCR}(t) \otimes h_{RE}(t)$
- DLL call:
  - Long AMI_Init (double *impulse_matrix, long row_size, long aggressors, double sample_interval, double bit_time, char *AMI_dll_parameters, void **AMI_dll_memory_handle, AMI_dll_ msg_type **msg)

# Rx – GetWave

- Inputs
  - Waveform* at the input of RX Equalizer [x(t)]
    - LTI processing
      - x (t) = Stimulus(t) ⊗ $h_{TE}$(t) ⊗ p(t) ⊗ $h_{TCR}$(t)
      - x (t) = Stimulus(t) ⊗ $h_{TE}$(t) ⊗ p(t) ⊗ $h_{TCR}$(t) + crosstalk contribution + amplitude noise
    - Waveform Processing
      - x (t) = Output of TX GetWave ⊗ p(t) ⊗ $h_{TCR}$(t)
  - Data for GetWave (output of RX_INIT)
  - Start of New Message Flag
- Outputs
  - Equalized RX data [y(t)]
  - Clock Ticks
- DLL Call
  - Long AMI_GetWave (double *wave_in, long wave_size, double *clock_times, void *AMI_dll_memory)

*: Voltage

# Executable Code Delivery Mechanism

- Executables that include both Init and GetWave
  - Exe needs to stay active between Init and GetWave calls
  - GetWave data needs to be transferred from EDA platform to model executable using either disk or shared memory

- DLLs
  - Provides improved performance over standalone executables when GetWave is used

# Next Steps

- Write Detailed BIRD
- Present Detailed BIRD to IBIS-ATM
  - Target April 24th, 2007
- Create reference models
  - Sample .exe and .dll models as coding examples
- Work with semiconductor vendors to develop & test real models
- Present to IBIS Open Forum
  - Target IBIS Summit at DAC 2007


cadence™


SiSoft™


Mentor
Graphics®