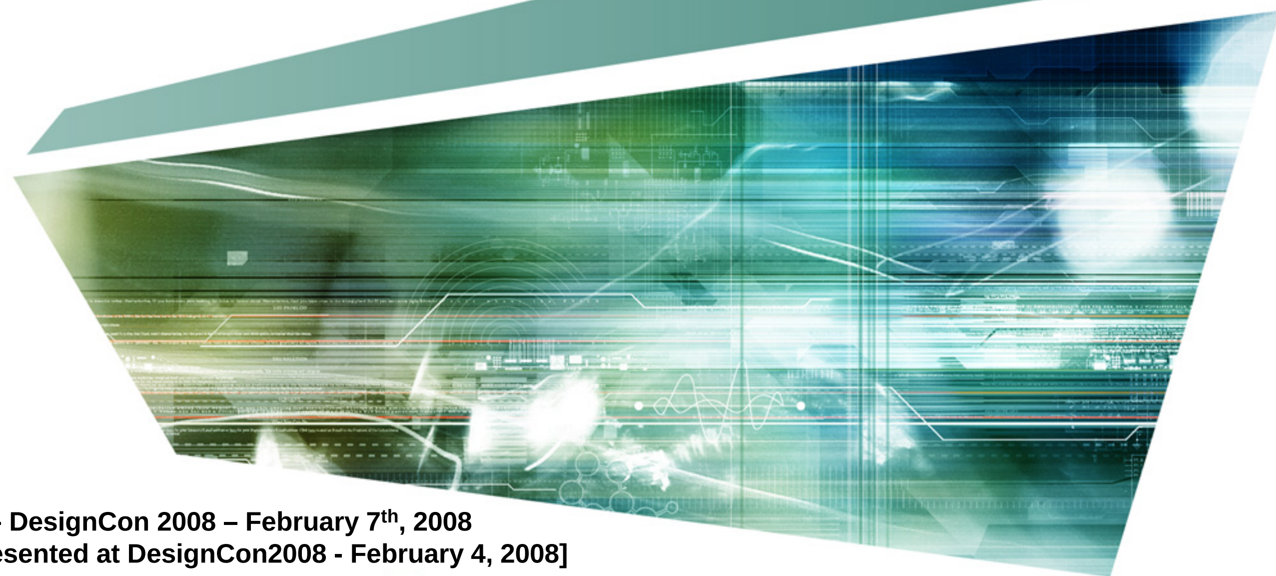
Experiences in Developing and Correlating Eight Interoperable Algorithmic Models

Adge Hawes IBM
Ken Willis Cadence Design Systems



IBIS Summit – DesignCon 2008 – February 7th, 2008
[Originally presented at DesignCon2008 - February 4, 2008]

AMI Modelling Tips and Tricks

Making life easier for the AMI modeller

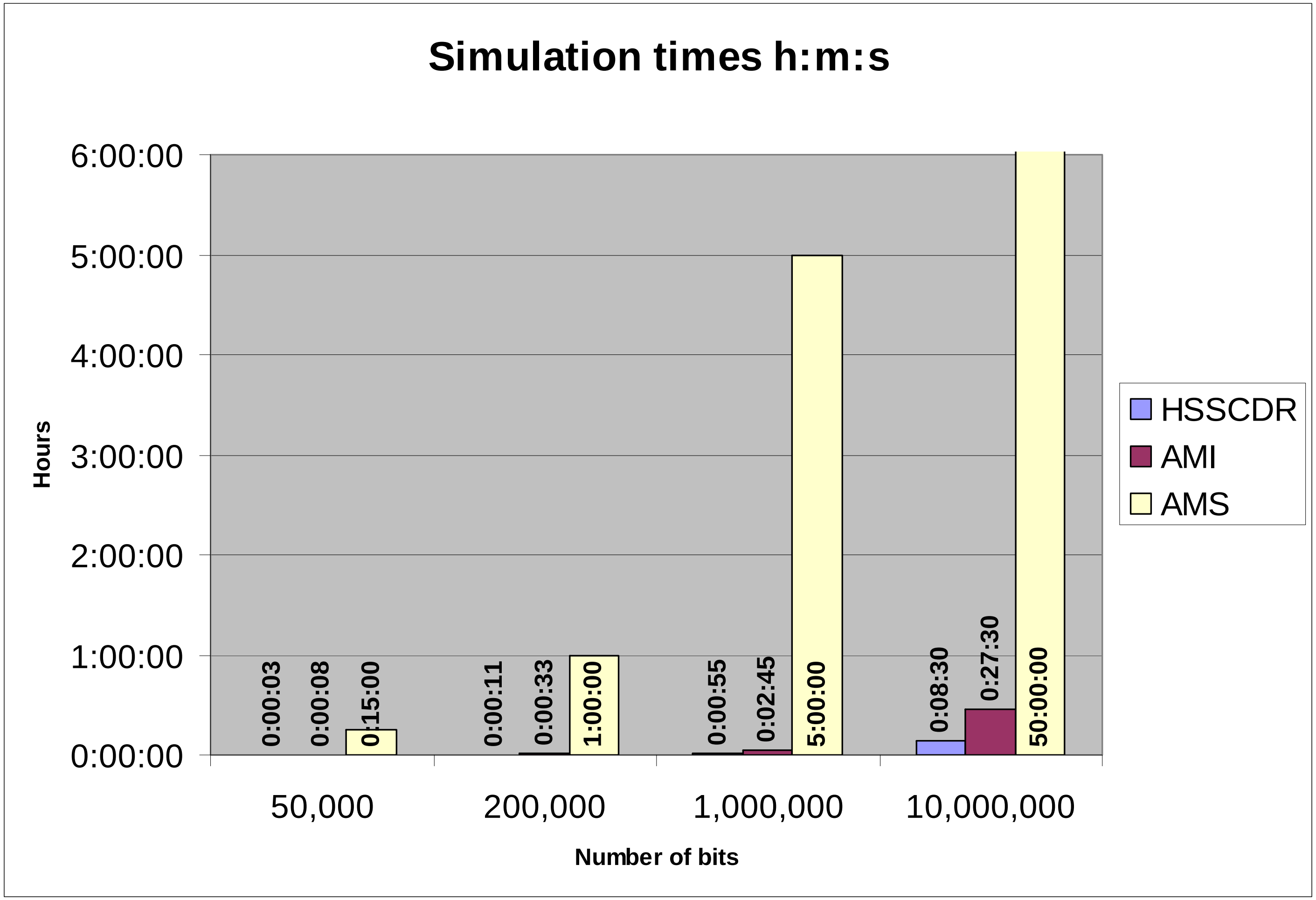
Adge Hawes, IBM

IBM AMI Models

Tech	*Speed (Gb/s)*	*Model Available*		*H/W Correlated**	*Features*
		Linux	*Windows*		
90nm	6.40	Jul 2007	Oct 2007	Yes	4-tap FFE, AGC, 3- or 5-tap DFE
	11.1	Sep 2007	Oct 2007	Yes	3-tap FFE, AGC, 3- or 5-tap DFE, advanced equalization
65nm	6.40	Jan 2008	Jan 2008	Yes	4-tap FFE, AGC, 3- or 5-tap DFE
	10.55	Jan 2008	Jan 2008	In progress	3-tap FFE, AGC, 5-tap (up to 8Gb) or 1-tap (10Gb) DFE, advanced equalization

*via internal sim tool

AMI Simulation times



AMI Modelling

- ANSI standard C (not C++)
- Closed Source (proprietary)
- Functional and algorithmic
- Represents your hardware
- Available for Windows and Linux

Use Free Tools

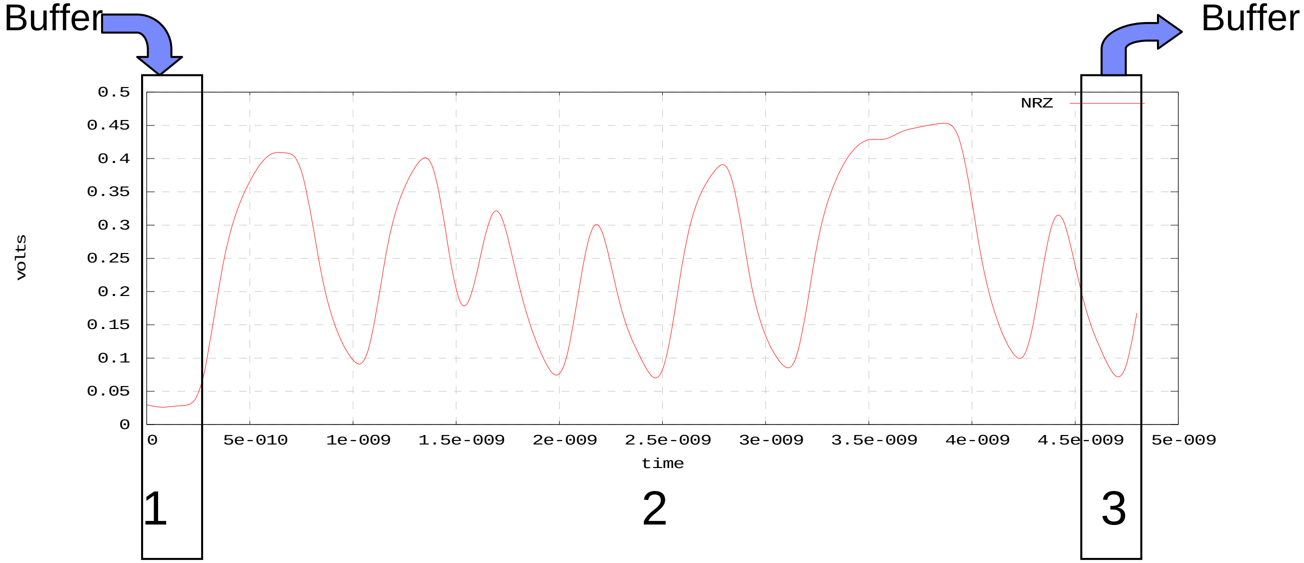
- Can't afford MATLAB or not enough licenses?
  - Open-source OCTAVE available for Windows and Linux (www.octave.org)
  - Euler for Windows/Linux (http://mathsrv.ku-eichstaett.de/MGF/homes/grothmann/euler/)
- Use latest GCC for Linux compilation
  - `gcc –shared –o dllname.dll source.c`
- Free Visual C++ Express for Windows
  - Include libraries (`/MT`)
- Free Editors have syntax highlighting
  - Crimson Editor (www.crimsoneditor.com)
  - Codeblocks (www.codeblocks.org)

Beware Open-Source

- Tempted to get common routines from open sources (e.g. FFTW)?
- GPL has “viral” effect – may require you to publish source code that uses it
- LGPL (Lesser or Library) may be acceptable, if dynamically linked
- Some code may not allow commercial use
- If in doubt, consult your IP Law

AMI_Getwave Buffering

- EDA tool will break input wave at arbitrary points
  - Boundaries will not coincide with clock edges
- Clock cycle processing may straddle AMI_Getwave calls
- Recommended processing:
  1. Leftover waveform + partial cycle (as whole clock)
  2. Bulk of waveform
  3. Remaining part-cycle
- Allocate enough space for buffering



Usual C Advice

- Remember to:
  - Check that pointers are not NULL
  - `free` what you've `alloc`'ed
  - Be aware of floating-point accuracy
  - Watch types (`double`, `long`, `int`, `float`, etc., use "`l`" for long or double input)
  - Check case and spelling (e.g. `AMI_`**`G`**`etwave`)

Windows and Linux

- Aim for common source
  - #include os.h:
    - ```
      #define DllExport __declspec(dllexport)          /* Windows */
      #define WIN32 1
      #define LINUX 0
      //#define DllExport extern                       /* Linux   */
      ```
- Watch path settings (`C:\` vs `/home/test`)
- Expect minor differences
  - MS vs GCC
  - Accuracy in 4th or 5th decimal place
  - Don't let differences accumulate


      ```

# Code example: Data structures

```
typedef struct dll_obj_str {
    long sample;   /* count of the cycle number */
    char path[MAXPATH];   /* the place to find the executables */
    char pathsep;   /* path separator */
    double vmeas;   /* The crossing point, in volts */
    double vmax;   /* The maximum signal, in volts */
    double vmin;   /* The minimum signal, in volts */
    double vamp;   /* The amplitude of the signal */
/* ... */
    int lastbits[MAXDFE];   /* the last bits stored, +/- 1 */
    int ndfe;   /* the number of DFE taps */
    cdr_ext *cd;   /* a pointer to the cdr_ext structure */
} dll_obj_type;
```

Code example: AMI_Init (1)

```
DllExport long AMI_Init(double *impulse_matrix,
                        long row_size,
                        long aggressors,
                        double sample_interval,
                        double bit_time,
                        char *AMI_parameters_in,
                        char **AMI_parameters_out,
                        void **AMI_memory_handle,
                        char ** msg)
{
/*  ...  */
    dll_obj_type *dll_obj = 0;  /* a pointer to our parameter object */
/*  ...  */
    /* set up config controls */
    stree_type *config;
    config = streeRead(AMI_parameters_in); /* config points to tree */

    /* generate the storage for our parameters */
    dll_obj = (dll_obj_type *) calloc(1, sizeof(dll_obj_type));
    /* dll_obj is now a pointer to some allocated space big enough
       to hold our dll parameter object */
```

Code example: AMI_Init (2)

```
    /* now allocate space for the cdr variables, and set to zero */
    dll_obj->cd = (cdr_ext *) calloc(1, sizeof(cdr_ext));
/*  ... */
    /* determine if Linux or Windows */
    if (WIN32) {
        dll_obj->pathsep = '\\';
    } else {        /* assume Linux */
        dll_obj->pathsep = '/';
    }

    /* Initialize some parameters */
    dll_obj->sample = 0;          /* number of clock cycles processed */
    dll_obj->wave_time = 0.0;
    dll_obj->thiscycletime = 0.0;
/*  ... */
    streeDestroy(config);

    return 1;                     /* for success */
}
```

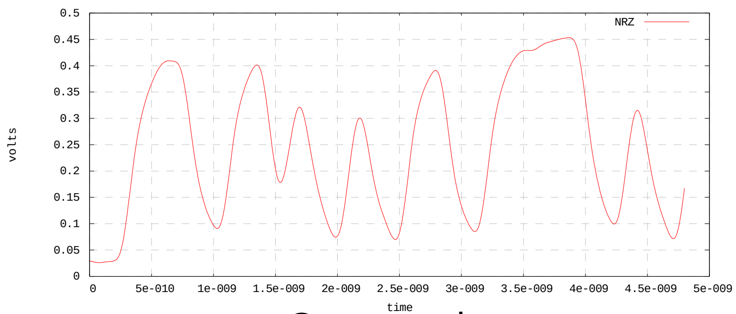
Code Example: AMI_Close

```
DllExport long AMI_GetWave(double *wave_in,
                           long    size,
                           double *clock_times,
                           char   **AMI_parameters_out,
                           void   *AMI_dll_memory)
{
/*  ...  */

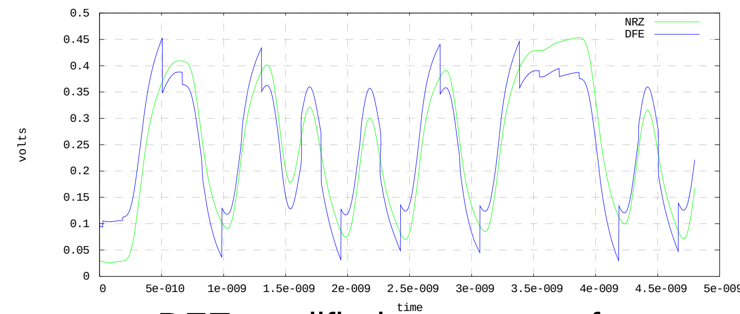
    return 1;
}

DllExport long AMI_Close(void *AMI_dll_memory)
{
/*  ...  */
    dll_obj_type *dll_obj = AMI_dll_memory;
    free(dll_obj->cd);              /* free the cdr variables */
    free(dll_obj);                  /* free the whole object */
    return 1;
}
```

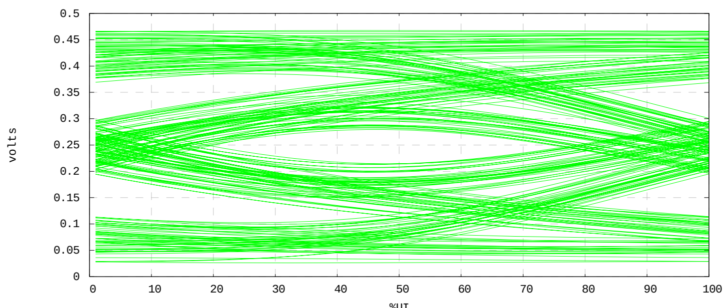

Execution examples



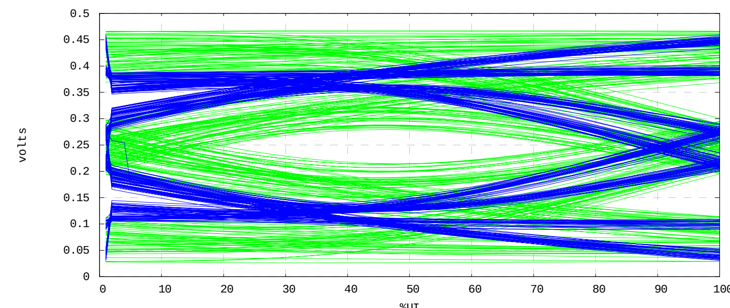
Getwave data



DFE-modified return waveform



Eye from clock ticks



DFE eye (before offset)

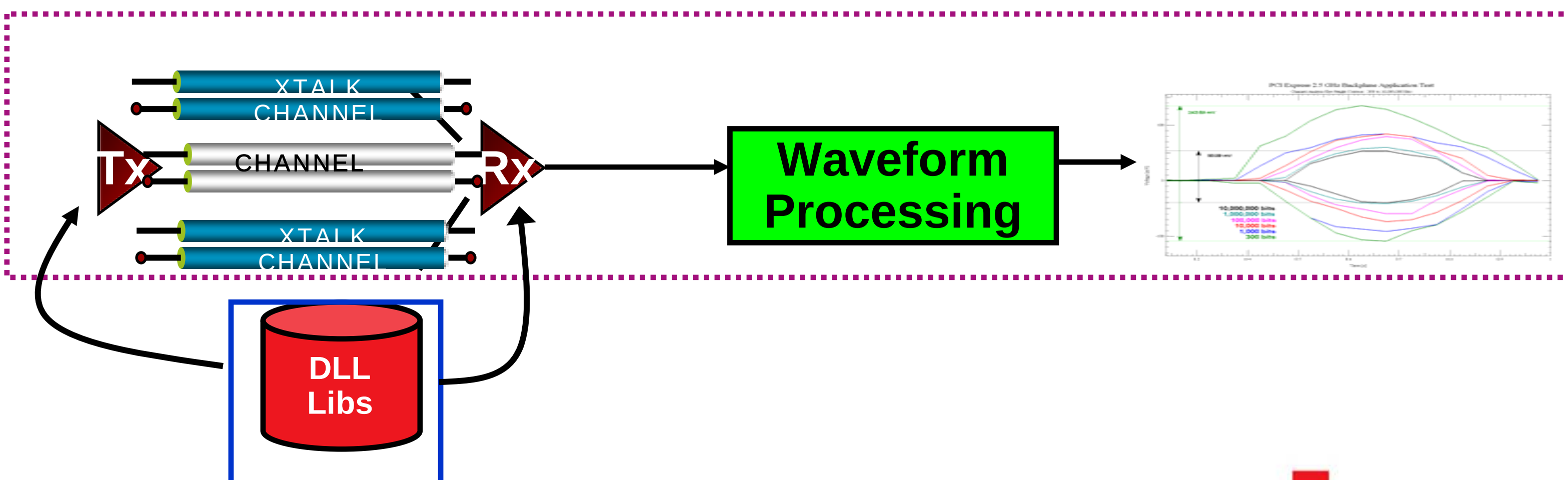
Correlating Algorithmic Models

Ken Willis, Cadence Design Systems Inc.

cādence™

Algorithmic Model Correlation
Tips, Tricks, & Pitfalls

- Assumptions
- What are we correlating?
- Basic strategy
- Common pitfalls
- Summary



Assumptions

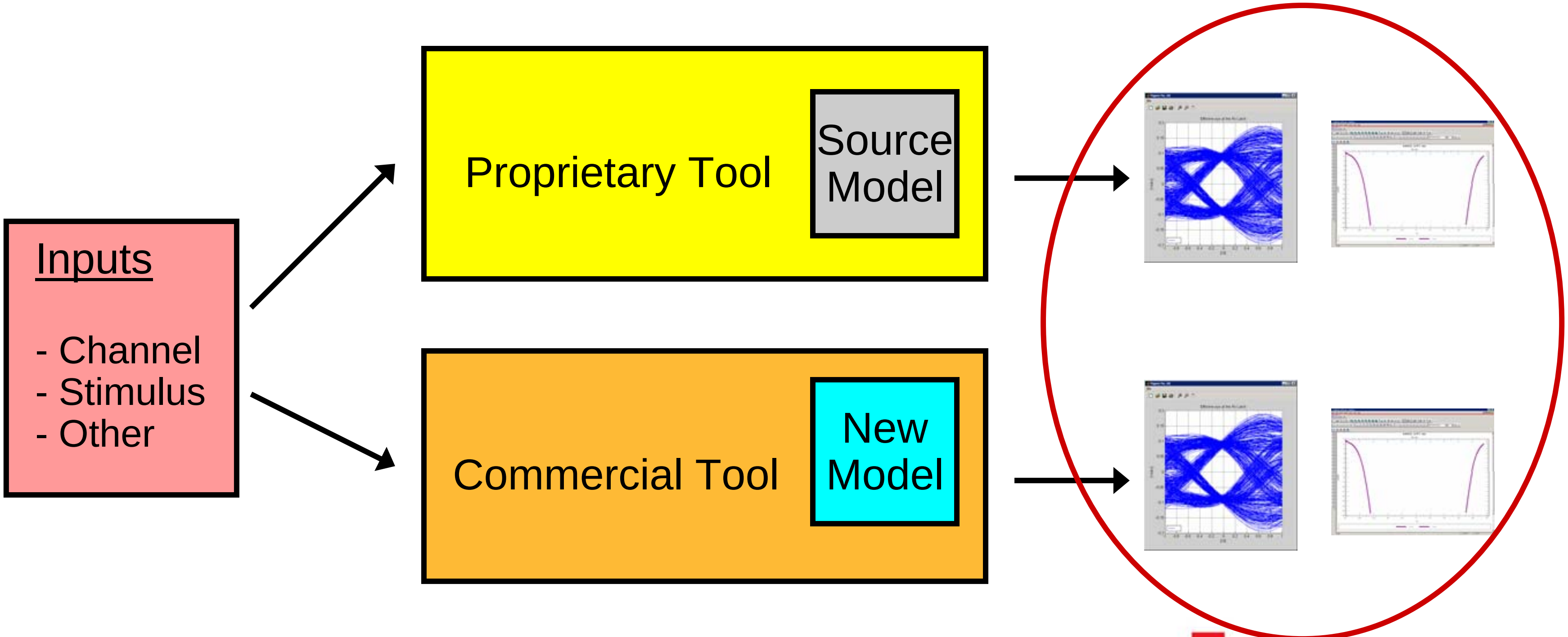
- Algorithmic model exists in some proprietary format, and is consumable by a proprietary tool

- Requirement: Correlate an IBIS AMI API based algorithmic model running in a commercial tool to known reference *silicon or proprietary tool* given same inputs

- For this discussion, assume FFE (i.e. pre-emphasis) for Tx, DFE (Decision Feedback Equalization) for Rx

What are we correlating?

- Simulation results between proprietary tool using “source” algorithmic model, and commercial EDA tool using “new” algorithmic model, with identical inputs

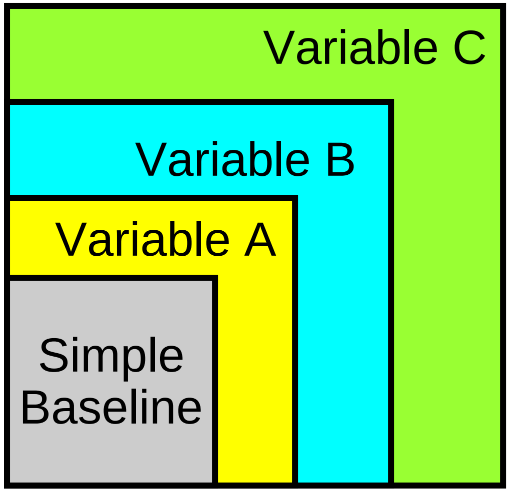


Basic Strategy – Layering of Variables on Established Baseline

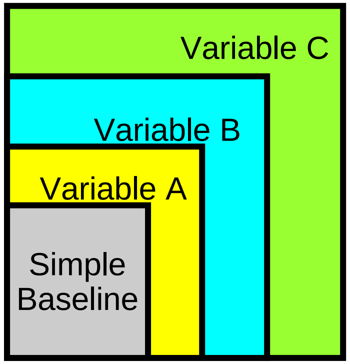
- Start simple
  - Correlate the easy case first
  - Lossless channel with terminations
  - No filtering
  - Pulse stimulus

- Next add:
  - Complex passive channel
  - Bit stream
  - Filtering
  - Jitter injection
  - Other elements

"Alphabet Soup" of variables!



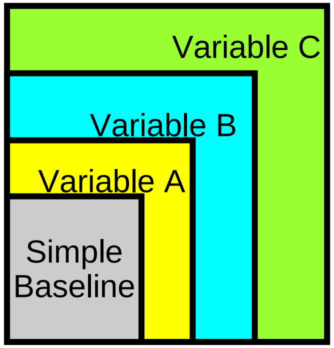
Tx correlation approach



1. Lossless Channel
   – Ideal Tx, Ideal Rx
   – simple pulse with no FFE/DFE
   – Establish rise time and voltage swing of driving source
2. Select common “lossy” channel model to use in both tools
   – Realistic case with “moderate” results is desirable (eye not fully closed when filtering applied)
   – Impulse response is best option to guarantee consistent representation of channel
3. Establish common stimulus for both tools, ex. PRBS 31 pattern of 1 million bits

cādence™

Tx correlation approach (contd.)



4. Correlate ideal Tx *no-package parasitics* with FFE through channel to ideal Rx termination
   – Verify same tap coefficients generated for same channel model in both tools
5. Correlate non-ideal (best/nominal/worst case) Tx through channel to ideal Rx termination
   – Include on-chip parasitics from Tx, account for process/temp/voltage variations
6. Add jitter injection and other effects (ex. Rj, Sj, Tx/Rx freq. offset, etc.) one variable at a time

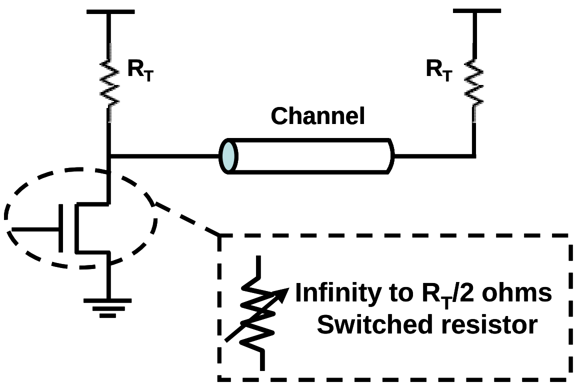
cādence™

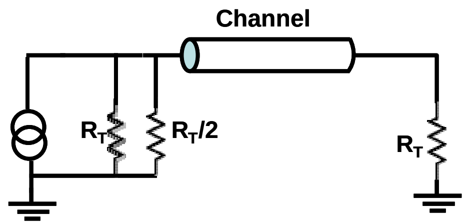
Common Pitfalls

- Tx/Rx circuit model assumptions
- Magnitude scaling
- S-parameter simulations
- Stabilization time
- Consistent measurements
- Supporting multiple platforms

Tx/Rx Circuit Model Assumptions

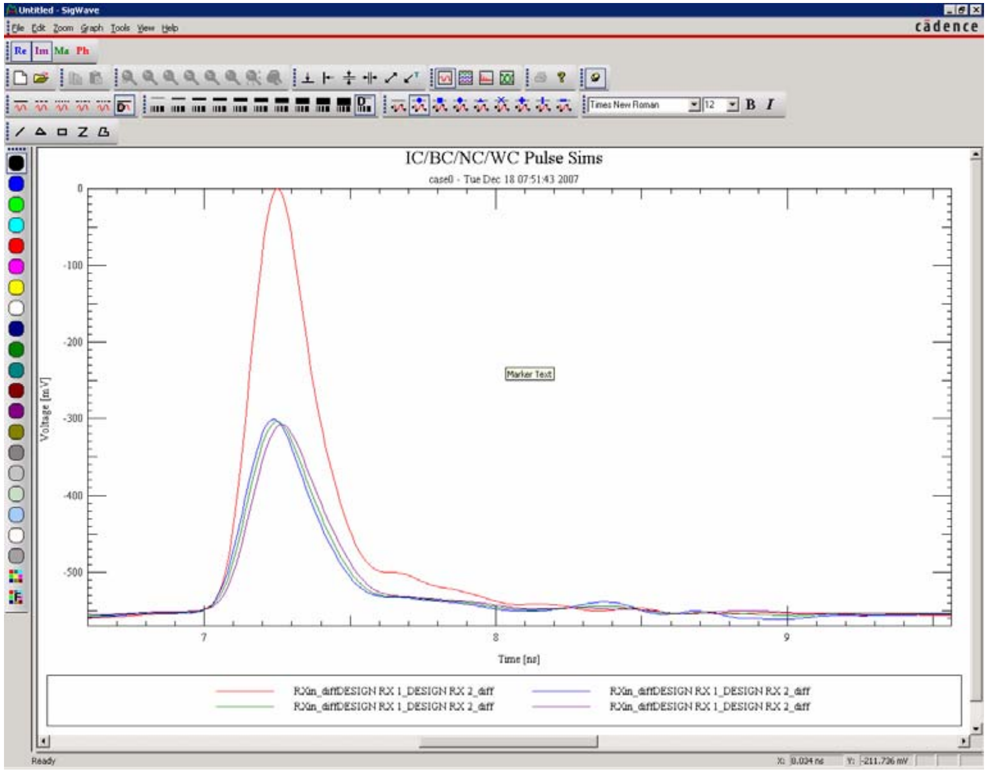
- Consistent front end circuit model required
- Same circuit model should be assumed in both tools
- Make sure you know what is being used in the tool you are correlating with!





Magnitude Scaling

- Consistent Impulse Response definition necessary
  - Internal tools can contain “hidden” scaling factors for corner cases
- Affects eye height magnitude correlations

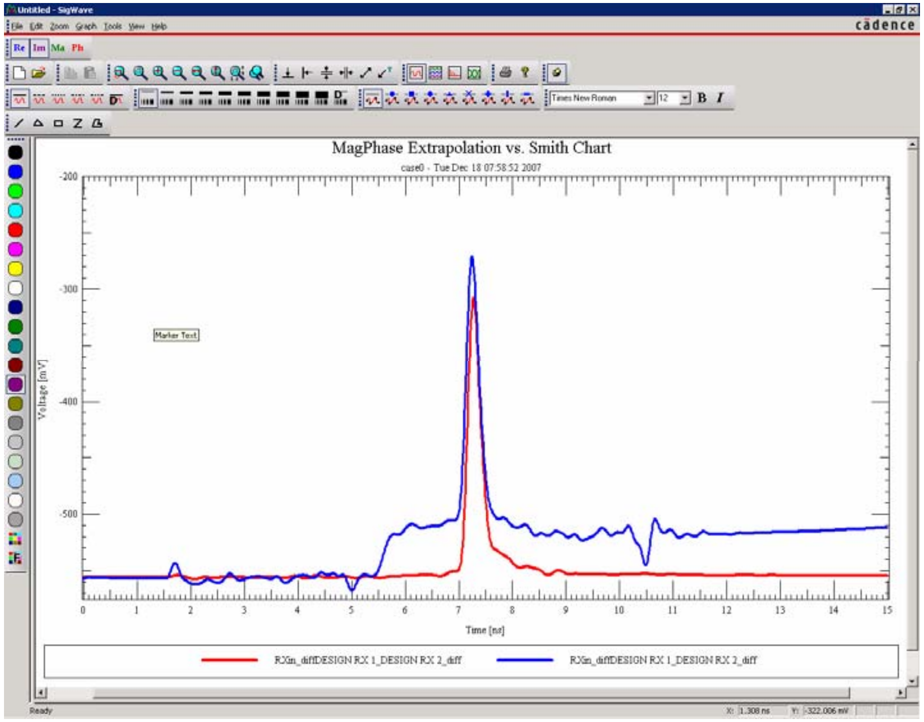


cadence

S-Parameter Simulations

- More stringent S-parameter criteria required for time domain simulation
  - Start/end freq
  - Number of steps
  - Linear steps

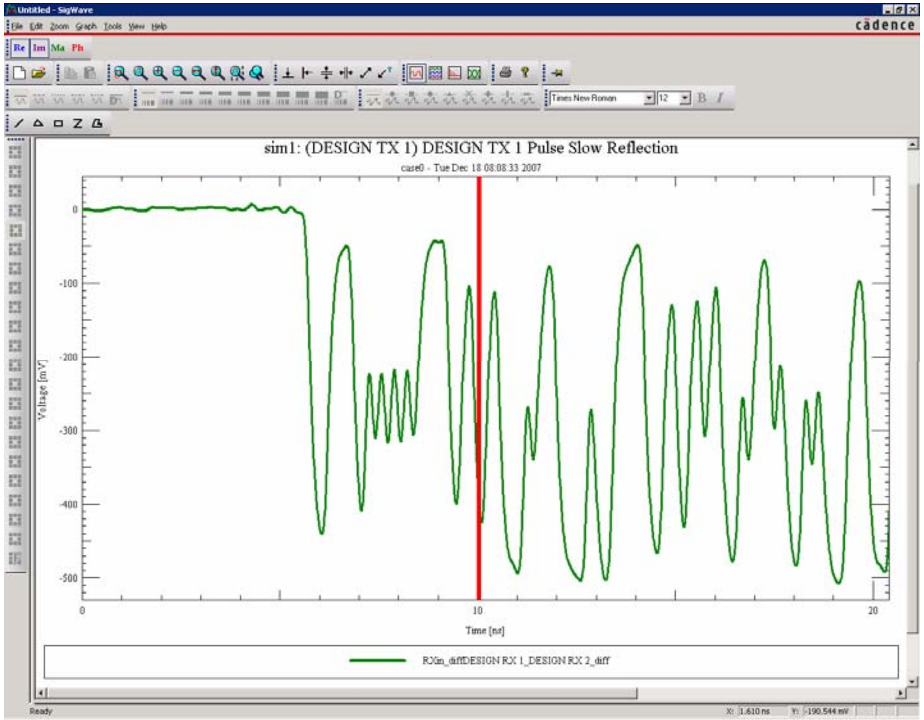
- Robust DC extrapolation techniques needed for time domain s-parameter simulations



cādence™

Stabilization Time

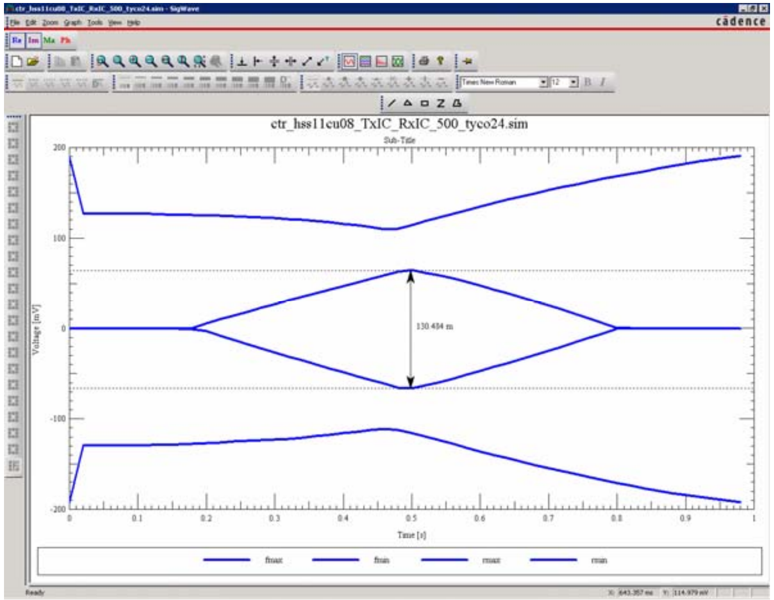
- Clock recovery algorithms may need to run some traffic before locking in
- Ideal (underdamped) cases may need more stabilization time than non-ideal (overdamped) best/nominal/worst case corners
- Allow scenario to stabilize before recording data for measurements
- Particularly important with DFE waveform processing (“AMI_GetWave” call)

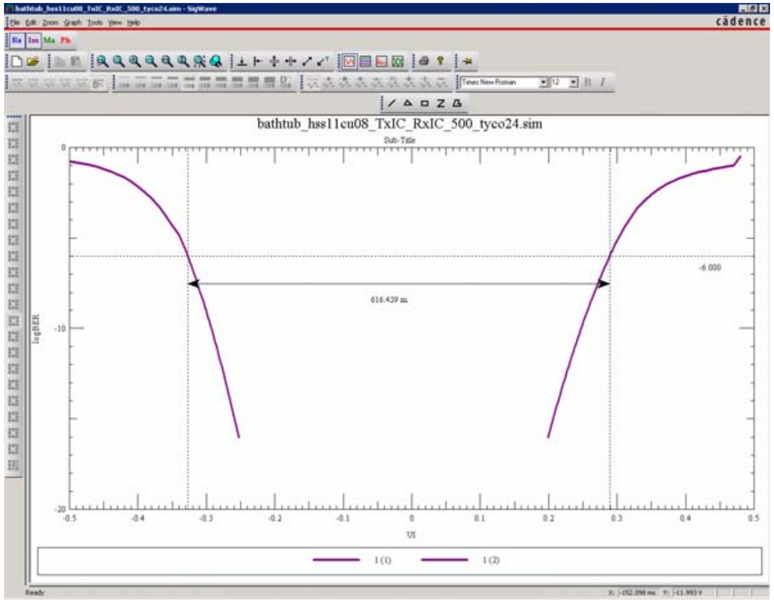


cādence™

Consistent Measurements

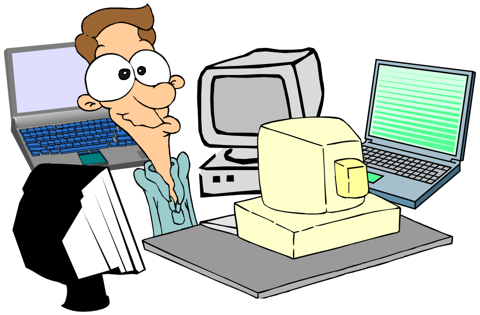
- Ensure “apples-to-apples” measurement criteria for outputs of proprietary and commercial tool





Supporting Multiple Platforms

- Small numerical differences between hardware platforms (ex. Linux & Windows) can significantly influence results
- Establish baseline ***regression tests*** for a given DLL across all supported platforms
- Validate each new version of DLL vs. previous "golden" results with standard testbench



So many platforms, so little time …

cādence™

Summary

- Algorithmic models should be correlated against the source *silicon or proprietary tool*

- Success requires an organized and methodical approach

- Avoid common pitfalls and accelerate model releases!

cadence™