ADS for Signal Integrity optimization

Characterizing a channel: Where we are coming from!

In the beginning there was the transient simulation! Some time ago the best way to ensure a functional system was to perform a time domain simulation with a text file controlled SPICE simulator. The result was a time domain waveform that was evaluated during a post processing for Signal integrity quality. The main thing to do was to calculate a dataeye based on an "imaginary" perfect clock as phase reference. Due to the rising signaling speeds and the decreasing margins in the timing the concept was improved and adjusted to the signaling concept of the investigated system. For communication systems that used an embedded clock the PLL behavior was taken into account and the Jitter simulated or subtracted from the remaining timing margin of the eye. For source synchronous signals the clock was simulated in addition and was used for setup/hold calculations or data eye generation. In addition to the concept changes the simulation accuracy had to be improved to account for the decreased margins. This resulted in a complex modeling that takes into account even small parasitics. In addition to this the simulation needed to include the worst case combination of all negative signaling effects. To figure out the real margin of a system the worst case ISI had to be combined with worst case X-talk, The simplest way to do so is to perform a simulation with a PRBS pattern long enough to account for the memory of the channel and combine it with even and odd X-talk. This is simple … but takes a LOT of calculation time. And still this is a process taking two steps: the simulation and the post processing. Just the simulation time could require to run the simulation over night and do the post processing the next day just for a single simulation. Due to the huge amount of data the post processing can not be done in realtime. The post processing time is short compared to the simulation, but still requires minutes to get the results.

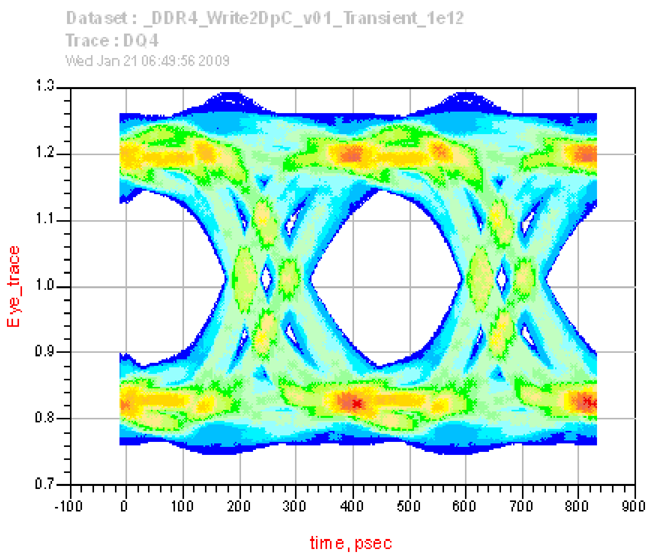

Fig 1: Time Domain simulation with a 2^12 PRBS pattern and even and odd X-talk

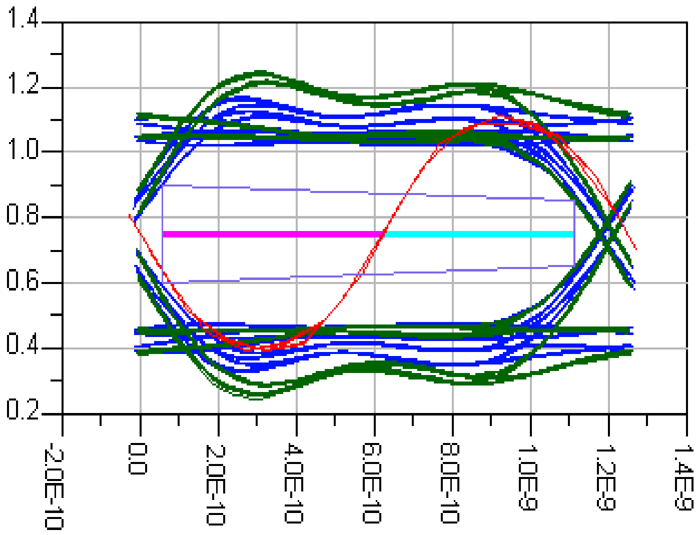

Fig 2: Data Eye with Phase reference and AC/DC tSH measurement

Characterizing a channel: Where we are today!

The simulation accuracy was always adjusted to the available calculation power. For optimization of a channel several long running simulations have been needed. Therefore it was necessary to reduce the accuracy and to optimize each effect separately. Even today it is a good thing to separate ISI and X-talk. Only the final evaluation, to judge about pass or fail is a simulation that takes into account all effects!

To keep calculation time short the methods have gotten a big improvement in the last time. The channels are characterized by a step or (im)pulse response. This method is used e. g. by the free tool "Stateye" [1]. So instead of doing a long time domain simulation with a PRBS source only a single rising and falling edge is simulated (see Fig. 3). Out of these edges the channel behavior and the worst case dataeye can be calculated. Of course this can be done based on a S-parameter characterization of the channel too. But the use of the step response does have one significant advantage: It's straight forward to use a real driver and receiver. It's possible to use SPICE or IBIS models for the driver and Receiver. So the information on the nonlinearity of this devices is already included in the information of the step responses. Most statistical eye tools allow to use S-parameter as channel description too, but in this case the user has to use an ideal linear (50Ohm) driver or build a behavioral model of the Driver and Receiver. This statistical eye tools can calculate deterministic ISI and X-talk effects. So on a first look several simulations needs to be done: One for the data signal (the victim), stimulating it with a rising (and falling) edge and observing the output. One for each aggressor and observing the victims response. This can be simplified by stimulating all aggressors at once and only observing the combined impact on the victim. But even simpler is it with the boundary condition, that a channel on a PCB is a passive linear system. In this case the response on the Victim from a stimulated aggressor is the same as the other way round. Again this might not be exactly true in real worlds systems (e. g. or when doing a layout accurate simulation) as the aggressors might have a slight different routing as the victim, but in a first order the result is accurate enough. Using this approach allows to evaluate a system with multiple aggressors with a single simulation!

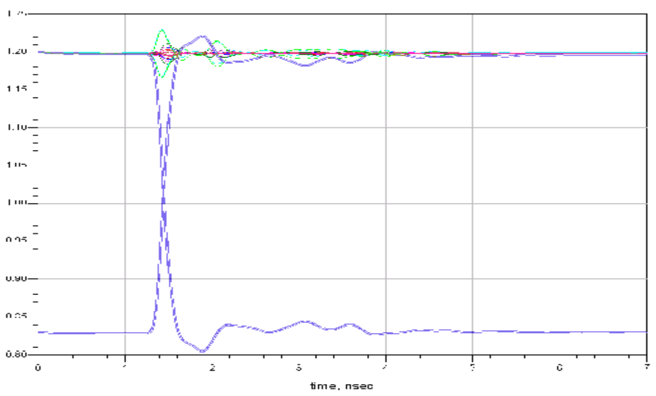

Fig 3: Simulation result for a rising and falling step. Usually real systems does not have symmetric edges so both are needed ...

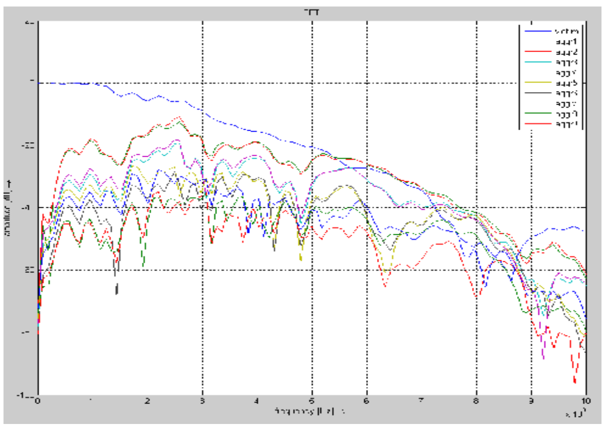
Fig 4: The FFT gives a quite good impression for the quality (SNR) of the Channel

This results in quite short simulation times, but adds some burden to the post processing. But as already the post processing for a long time domain simulation took some time, there is still the same time needed now (some seconds to minutes). Another big advantage of this method is, that there can be evaluated some other effects without doing a new simulation. The tools can add random jitter for TX and RX circuits and calculate Bit Error Rates. Pre-Emphasis and equalization can be added, and and even the optimized taps can be calculated. The pattern of the signals can be changed form a PRBS to a Clock pattern. Doing an FFT (Fig. 4) on the signal shows the channel characteristics in the Frequency domain, and so on and on ...

Using this method required a third party tool before ADS2008. One could use the export from the data display to write the step response to a simple ASCII file. The better way is to write it directly out from the simulation by using the “write_var” command in a “MeasEqn” or even better from a separate netlist. Using a netlist has the huge advantage, that the equations can be written a normal text editor with all copy and past functions. Even not intended for this feature it is possible to generate a fully automated process for data evaluation for a third party tool using “write_var”. When doing a sweep it is possible to create subdirectories by the “mkdir” command for each sweep step. All variables of the design can be written into a text file to document the simulation conditions. And of course the simulation result is exported. Additionally even a script can be generated that runs the third party tool over all the sweep cases to do the post processing and data evaluation!

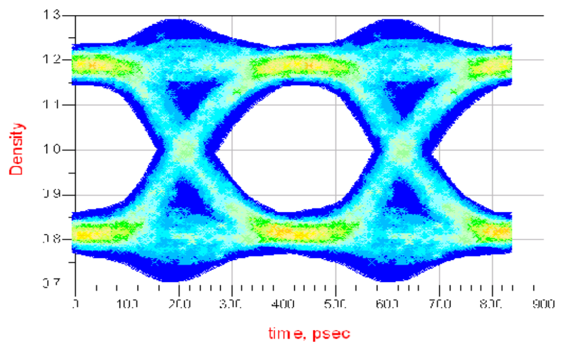

Fig 5: DataEye generated with ADS FastChannelSim feature

Even this would not be necessary in ADS2008 due to the implementation in the digital Ptolemy simulator SI engineers might prefer the RF design environment to perform this simulations. In ADS 2009 this feature is now implemented in the “FastChannelSim”. As some might have implemented already their own statistical tool for data evaluation with additional features they might still stay with the external approach. But if you are starting now to use this method the “FastChannelSim” should be the first thing to explore!

Optimizing a Channel: How to do it efficiently?

So overall there are a lot of ways to investigate the quality of an existing channel. But now the difficult thing comes into the game: How to optimize the channel? The simple approach is to do parameter sweeps over all interesting variables. This is quite some work, but straight forward on a Point to Point connection. On each of the Parameters (e. g. Board Impedance Z_0, Driver Impedance R_{on}, Termination Impedance R_{TT}) the dependency can be found, and the best case can be taken. In a perfect system all impedances should match, so a configuration where $Z_0 = R_{on} = R_{TT}$ is a reasonable starting point. But in real world systems e. g. the routing impedance is not perfect, as it is changed by packages and via's. It is not completely true that each parameter can be optimized on it's own, as all parameter are interact with each other (e.g. R_{on} and R_{TT}), but in a first order you will get an optimized value for each swept variable. If than a simulation is performed with the set of variables found in separate sweeps the result might not be the absolute optimum, but it is definitely a quite good configuration!

But it is going to be complicated on a multi drop bus. One quite complex example is a source synchronous, burst type, bidirectional, single ended multidrop memory bus. Independent if you are talking about DDR2/DDR3 or the upcoming DDR4 one of the most important requirements is to connect as much memory as possible at maximum speed to one channel. Such a configuration creates a huge matrix of dependent and independent variables that needs to be varied to get the optimum configuration. Due to the asymmetric nature of the channel the signal integrity characterization needs to be done twice: separately for writes and reads where some of the variables are fixed for

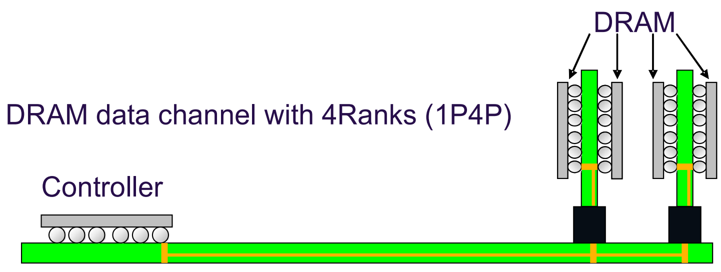

Fig 6: Example for a MultiDrop memory bus

both cases (Board topology and impedance), but some can be optimized independently (dynamic ODT settings).
And worst of all, the outcome will be not a perfect eye, but the “best” solution will be achieved by a trade off between timing and voltage margin. So the difficult question to answer is: Which is the best solution ?

The simple approach to optimize each parameter separately is going to fail due to high interaction between the variables. For example there are different termination schemes (which of the devices on the bus is terminating with which value) that interacts highly with the topology on the bus, the driver and board and DIMM impedance.
The brute force attempt so solve the problem by performing a multidimensional sweep and taking the best parameter combination is a nice idea. But just due the huge amount of possible combinations this is not a reasonable solution.

Optimizing a Channel: Let ADS do the job

“Luckily” (or was this intended?) there is a nice feature included in ADS: the Optimizer! But how to utilize it in such a complex system? The first difficulty is, that the optimizer needs some feedback out of the simulation. Here is one feature of ADS that many people forget about: Each equation used in the data display also works in the schematic. So once it is proven, that the eye calculation is working in the data display, this equation can be used on schematic level too. And now the result can be used as input of the optimizer. Unfortunately it is not that easy to debug such a setup and even worse: A long running simulation is needed to create a dataeye. The previous described method of using the step response for eye calculation can not be used, as this requires an external tool. Even the FastChannelSim in ADS2009 will not be able to handle the problem as it's result can not be used for optimization (at least not in it's first implementation).

But the solution is only some Equations or lines of AEL code away. Instead of using a step response it turns out to be usefull to use a UI (Unit Interval) wide pulse as basis for the optimization. First it's a good idea to normalize the pulse in the voltage scaling to have a solid basis for different input signals. Otherwise different combinations of R_{on}, R_{TT} and V_{TT} (Termination voltage) might shift the signal and the equation is possibly failing.

Now let's separate ISI optimization from X-talk again. The first task is now to slice the pulse waveform into UI fine pieces. Cutting in parts is easy, but where to make the cuts? As usual the answer is: That depends … in this case from the real configuration. A quite good approach is to take the maximum of the pulse and center the UI around. If you do have a clock or strobe signal you might want to use this as a phase information for centering the UI. The outcome is now one slice with the information on the transmitted pulse (green area) and a lot of slices with the information on pre- and post pulse disturbances (yellow area), called ISI. Subtracting now each of the ISI UI's from the transmitted pulse UI reduces the pulse like

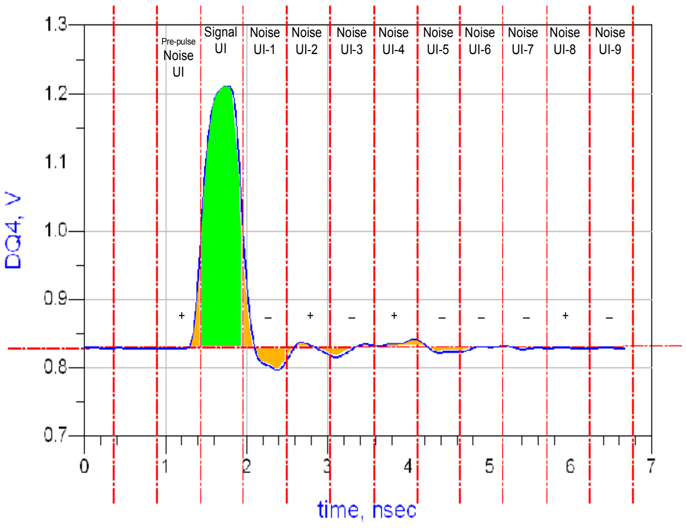

Fig 7: Pulse response simulation result as input for the ADS optimizer

the ISI does! As you see in this example the signal is nearly settled in the 9th UI after the pulse. So to simulate 10 bits should be enough for this investigation. Trying to catch the worst case pattern with a conventional time domain simulation would require a PRBS simulation with 2^10 bit PRBS pattern!
Now let's think about the evaluation of the data we just created. Having done the subtraction of the noise UI's from the signal UI we can check the eye height or eye width of the remaining pulse and use the outcome as goal for the optimizer. As mentioned before the best eye might be a trade off between voltage and timing margin. So there can be set two optimization goals: one for eye width and one for eye height. But it turns out to be quite difficult to find the correct weighting to get the optimum result for both. A possible solution for this problem is instead of using voltage and time, the energy of the pulse by measuring the area underneath the pulse. Overall this results in less then 10 MeasEqn in the Schematic. Using AEL code of course is the more elegant and flexible solution!

Now as we do have the ISI lets take a look on the X-talk. As described before we assume that we do have a passive linear system. So again we can run only a single simulation stimulating the victim with a UI wide pulse and measure the responses on the aggressors. On the right you see the top of the "victim" pulse as black dashed line, and the responses on the aggressors in different colors. Now the same approach as before can be used by calculating the noise of all Ui's (this time including the UI of the original pulse). This gives the necessary information for the impact from the X-talk. As you can see in this case the x-talk peaks are injected exactly at the crossings of the signal. If this does not fit to your real system (e. g. because you do have a center aligned Strobe), you can just shift the UI borders for slicing the aggressors as needed. Adding this feature results in about 10 additional MeasEqn in the schematic, where the number of equations increases with the number of the aggressors to be considered. If the solution is implemented as AEL a loop can be used to create a quite small but very flexible and fast AEL function.

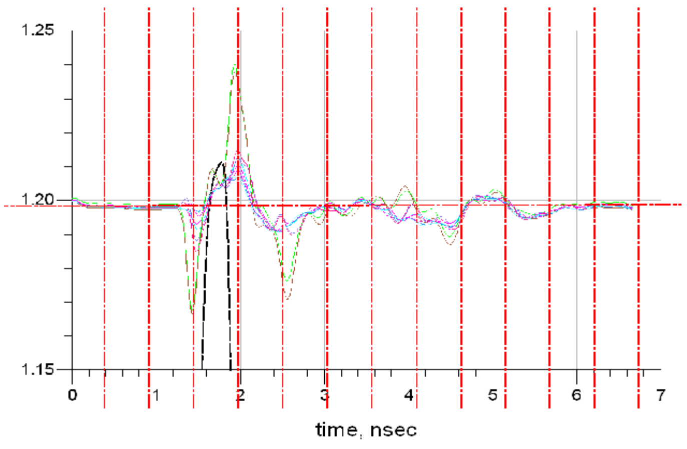

Fig 8: Single 1 pulse and X-talk on 8 aggressors

Additional features:

The nice thing in this simulation is, that you can get quite some more information out of this simulation. Finding the worst caste pattern for a 1 is quite simple. Checking from backwards for each UI whether the level (at the strobe point) is above or below the reference level results in the worst case pattern for the given channel. In the case shown above this is a "0100010101**1**0" starting from UI-9 where the blue "1" is the worst case pulse. As the resolution of the picture does not allow to read out the patter exactly this is indicated by the "+" and "-" signs in the Noise-UI slices. But for the AEL code it is no problem to read out the pattern. For the worst case "0" the pattern just needs to be inverted (... if rising and falling edges of the pulse are symmetric).
One can also calculate the required pattern length for a given accuracy. Assuming you would like to get a pattern length that catches at least e. g. 97% of the ISI noise you need to do one pulse simulation that is for sure long enough. So in this case a pattern of 25 bit would be on the safe side.

Now there are several ways to do the calculation. Each of the implementations below is a bit correct.
One solution is to measure the area under the trace for all slices, and then sum up the positive area of all noise UI's.. Once you crossed the 0.97 from the overall area you know how much bits to simulate. It might be better first to use the abs() function to ensure that each part of the area inside a noise slice is accounted for. So a pulse with the same amount of positive and negative area (like Noise UI- 2 in the example) is not “ignored”. Or you just sum up the abs() of the levels at the sampling point. Non of the 3 solutions above will give an absolute correct result, but all do not create much of programming efforts and give enough accuracy for the goal we want to accomplish.

Conclusion:

With only 10 to 20 lines of AEL code it is possible to replace a multidimensional sweep of a long running PRBS time domain simulation (including manual data evaluation) by a short channel pulse characterization. As shown there are quite some different options to implement calculations. Non of them might be perfectly correct, but this is not necessary at all. We are just looking for the best case configuration and reasonable input for the optimizer. Checking the quality of the resulting parameter set is still a task of conventional simulation and Timing budget calculation. In a practical example this method gave an improvement of 12% eye height over the solution found with sweeps for independent single parameter optimization in much shorter time!.

So using all features of ADS can really speed up your work and improve your results!

[1] www.stateye.org