System Level Timing Closure using HSPICE

Barry Katz
President / CTO
bkatz@sisoft.com

Todd Westerhoff
VP, Software Products
twesterh@sisoft.com

SiSoft
Maynard, MA

ABSTRACT

Successful high-speed design achieves system timing closure by balancing static timing and signal integrity analysis against decisions made during the design process. A high speed design methodology should allow the designer to determine how design tradeoffs affect both system timing margins and signal integrity performance.

This paper outlines a rigorous, repeatable methodology for approaching high speed design and shows how HSPICE-based signal integrity analyses fit within that process. We use a DDR2 memory example to illustrate the different signal integrity analyses that must be performed using HSPICE.

Table of Contents

Table of Figures

1.0 High Speed System Design

There is an important difference between high speed system design and signal integrity that is often overlooked. Signal integrity studies the analog switching behavior of digital signals, modeling effects like over/undershoot, crosstalk, inter-symbol interference and the impact of power subsystem design. Signal integrity will not tell the designer whether a high speed interface will function at speed – this requires integration of the signal integrity results with timing analysis (jitter, clock distribution, output skews, setup and hold requirements).

Successful high speed design requires a methodology that ensures positive design margin across all combinations of:

- Component timing (process)
- Voltage & temperature
- Package & PCB routing lengths
- PCB manufacturing variations (Z_0, loss)

Integrating static timing analysis and signal integrity across all these system variables allows accurate prediction of system margin.

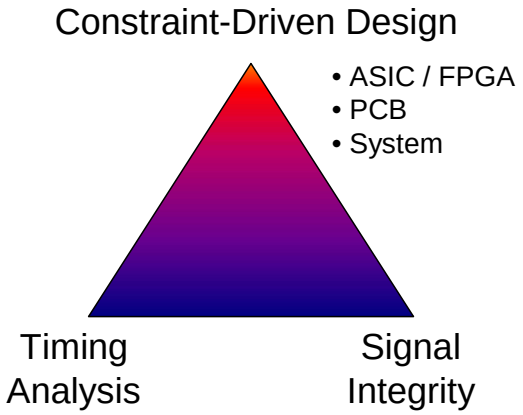

Figure 1 – High speed design relationships

Keeping different aspects of high speed system design in perspective is important. Every decision the designer makes - termination type, PCB line lengths, output buffer type or device pinout – has the potential to affect both signal integrity and system timing. Successful high speed design is a "balancing act" between physical design, static timing and signal integrity considerations (Figure 1). This requires that the high speed designer have a strong understanding of how different elements of the system are represented by timing and signal integrity models, and how potential design decisions are likely to impact system margins. Understanding how each design detail affects everything else is essential for achieving system level design closure.

2.0 Interface-based design

High speed designers separate a system into a collection of independent interfaces that are analyzed independently. Relationships between interfaces are considered when making decisions that affect physical design – PCB line length targets may be affected by adjacent interfaces and components, for example. Pre-route analysis defines routing strategies and budgets for allowable crosstalk noise that are later verified through post-route signal integrity simulations.

Each high speed interface will have one or more "transactions" that require timing closure. Typical transactions for a high speed memory interface include address/control, data read, data write, and relationships between data strobe and clock lines. Each transaction has timing requirements (usually setup and hold) that must be met. We can view high speed system design as a "tree" of analyses that are performed to validate these requirements.. Each system has multiple interfaces with one or more transactions. Each transaction involves multiple nets, each of which must be analyzed across multiple operating corners. Each net may be analyzed for a single routing strategy or over a range of lengths to provide flexibility to the PCB layout engineer. The number of simulations required for the entire range of interfaces / transactions / nets / corner cases and physical variations can be quite large. A complete high speed design methodology identifies which analyses need to be performed and ensures correct integration of static timing and signal integrity data. The methodology should also help manage the data produced during the analysis process.

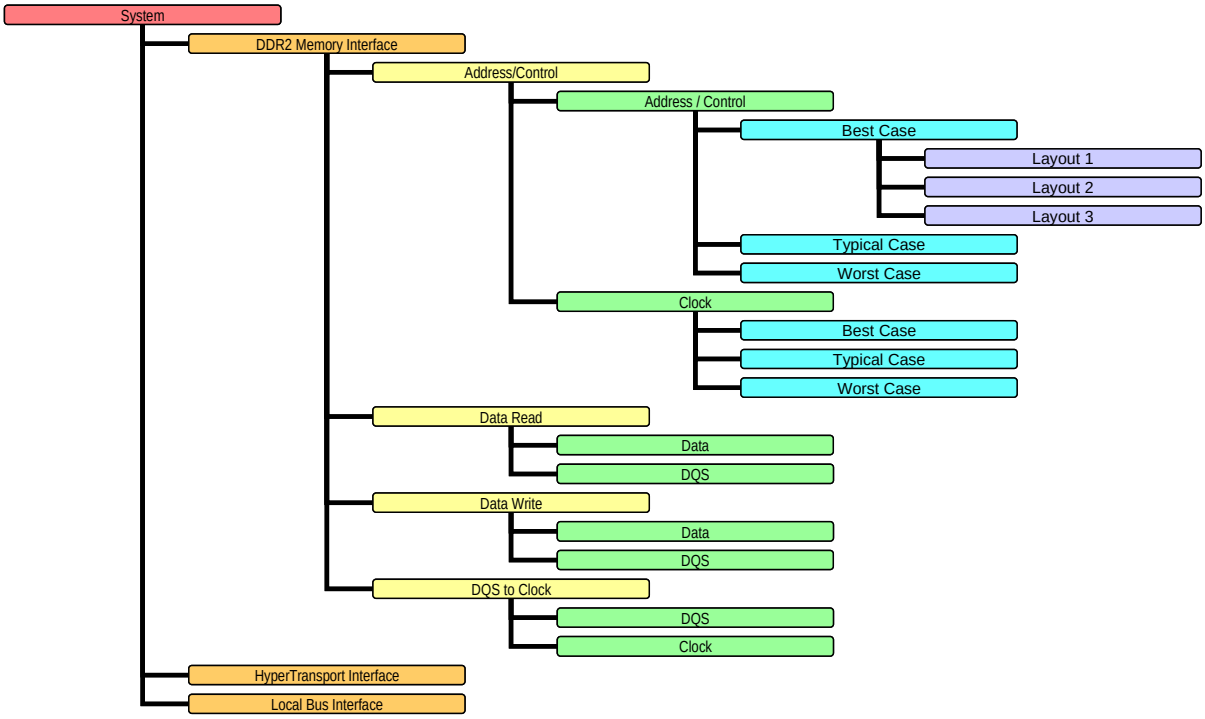

Figure 2 – High speed design analysis "tree"

Each time the design changes, affected interfaces should be re-analyzed and margins updated. The tree diagram in Figure 2 shows this can be a daunting task. Clearly, re-analyzing everything for each design change will be untenable, and there is a lot of data to keep organized. We refer to a procedural method for keeping volumes of static timing and signal integrity data organized as an "executable timing model". There are many ways to accomplish this – design documents, spreadsheets, custom scriptware, and ideally, EDA tools. No matter how this is accomplished, having a well-defined methodology for driving analysis and managing resulting data is crucial.

3.0 Static Timing Analysis

Analysis of any high speed design interface should start with a timing budget. Although static timing is often overlooked (or taken for granted), setup and hold margins cannot be properly calculated without taking timing into account. Initial timing analyses are performed using simplified interconnect delays; resulting timing margins provide guidance about how detailed signal integrity and power modeling simulations need to be. Signal integrity simulation involves a tradeoff between speed, modeling effort and accuracy – there's no point in running detailed simulations predicting interconnect delays to the picosecond level when an interface has multiple nanoseconds of margin.

Let's use an example to illustrate timing analysis and the role played by signal integrity. Figure 3 shows a 16 bit clock-forwarded interface. In order to perform timing analysis, we need four distinct pieces of information as shown:

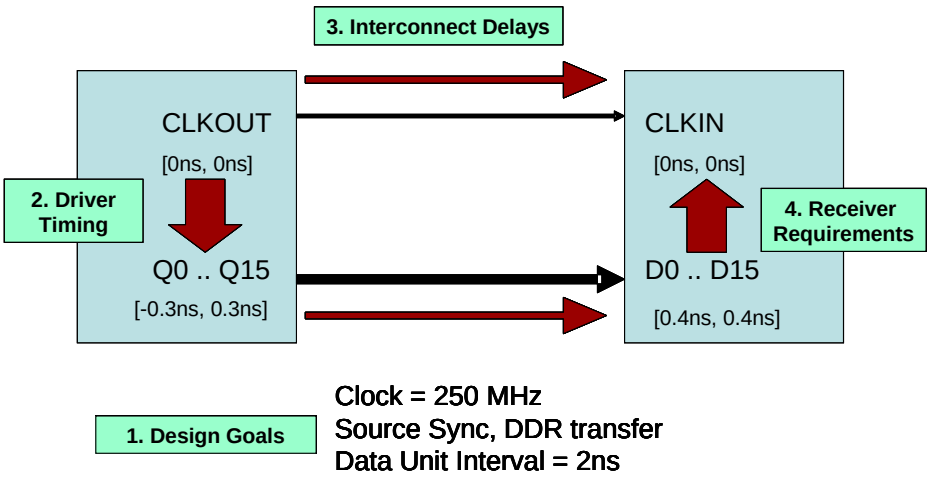

Figure 3 - Static timing example

Design goals tell us what speed the design is intended to operate at – both in terms of the clock speed and which clock edges transfer data. The data unit interval (UI) for this example is 2ns, so data will be transferred 500 million times per second.

The driver's timing characteristics establish the "best case" data valid window. In this example, the driver's outputs fire within +/- 300ps of the clock output, so the data valid window at the driver output is 2ns – 600ps = 1.4ns. The output clock and the data signals are edge aligned, so the output clock must be shifted to latch data at the receiver. The output clock should be shifted by ½ of the data unit interval (1 ns). This shift can either be realized through additional PCB etch or inside the receiving component.

The signals will accumulate additional skew as they propagate from the driver to the receiver. Our first-order analysis assumes the interconnect delay is a simple, linear function of net length. Signal integrity analysis will replace these estimates with more precise values later on if required.

We also need to know the receiver's timing requirements. In this case, receiver setup and hold requirements are 400ps each (800ps total). Since we had 1.4ns of data valid time at the driver, subtracting the receiver's setup/hold requirements gives us 1.4ns – 0.8ns = 0.6ns of margin, assuming no interconnect skew.

Given this information, we can derive the timing equations for this transaction. In order to account for interconnect skew, we'll allow for both minimum and maximum interconnect delay values. We represent the minimum, maximum delays on the clock net as [a1,a2] and the corresponding delays for the data nets as [b1,b2]:

Setup margin = [early clock] – [late data] – [setup requirement]
= [0ns + a1] – [0.3ns + b2] - [0.4ns]
= a1 – b2 – 0.7ns
Hold margin = [Data UI] + [early data] – [late clock] - [hold requirement]
= [2ns] + [-0.3ns + b1] – [0ns + a2] - [0.4ns]
= 1.3ns + b1 – a2

We'll use 180ps per inch as a first-order approximation for interconnect delay. Let's assume data nets are 3 inches long (540ps delay), and clock timing is adjusted using additional PCB etch, making the clock interconnect delay 1.54ns. Our first-order approximation for interconnect delay does not differentiate between min and max values, so a1 = a2 = 0.54ns and b1 = b2 = 1.54ns. Plugging interconnect delays for clock and data back into our timing equations, we compute our first-order estimate for setup and hold margin:

Setup margin = a1 – b2 – 0.7ns
= 1.54ns – 0.54ns – 0.7ns
= 0.3 ns
Hold margin = 1.3ns + b1 – a2
= 1.3ns + 0.54ns – 1.54ns
= 0.3ns

Our estimation of combined setup and hold margin thus represents 0.6ns, or 30% of our 2ns data unit interval.

4.0 The Role of Signal Integrity

Of course, real-world interconnect delays don't behave as neatly as our first-order approximation assumes they do. A first-order approximation assumes signals have no measurable rise or fall times – essentially, square-wave behavior. Real-world signals have rising and falling slopes, exhibit over/undershoot and ringing, and are affected by the load's input capacitance. Signal integrity analysis –analog analysis of the switching behavior of digital signals – is used to replace a first-order estimate of interconnect delay with a more accurate number based on the circuit's actual physical and electrical characteristics. A design is said to have achieved *signal integrity closure* when simulated signal behavior meets predetermined waveform quality requirements.

Component timing specifications represent a delay **from** one point **to** another point on the component and are based on vendor-specified loading and measurement conditions. We use signal integrity analysis to give us an interconnect delay value that can be plugged back into the static timing model. It is imperative that we understand the specific conditions under which component timing is specified, and "normalize" our signal integrity measurements to those conditions. The term *flight time* refers to the normalized delays thus derived from signal integrity analysis. In the IBIS modeling standard, model terms like Vref, Cref, Rref and Vmeas are used to specify the component timing information needed to normalize simulation results.

Timing closure occurs when the integration of timing and signal integrity flight times show acceptable setup/hold margins. The definition of "acceptable setup and hold margins" will vary from interface to interface. Computing setup and hold margin as a percentage of the data unit interval is useful, and centering margins (i.e. setup margin = hold margin) is also a common goal. *System design closure* occurs when both signal integrity closure and timing closure are achieved. Whatever the design's specific requirements are, achieving system design closure means the design is ready for manufacture from a high speed standpoint.

4.1 IBIS vs. HSPICE models

There are two common ways to simulate digital drivers and receivers. Circuit-level (SPICE) models describe buffer circuitry and associated device parasitics at the transistor level. Simulations using transistor-level models normally have good correlation to measurement because of the level of detail modeled, but simulations are slow. The Input/Output Buffer Information Specification (IBIS) is an EIA standard for modeling digital buffers at a "black box", or behavioral level. The IBIS buffer model describes behavior of buffer models by characterizing key device characteristics (output voltage vs. current, output voltage vs. time into a specified load) and populating elements of a pre-defined modeling template. IBIS models simulate much faster than their transistor-level counterparts (typically 10X or more) because less of the device's internal detail is represented. Even though IBIS models have less detailed information about the device, they produce equivalent results for most system level analyses.

The IBIS specification goes well beyond providing black-box models for I/O buffers. IBIS is also a component-level specification, listing all the pins on the physical design and providing device pin to buffer type mapping. IBIS provides parasitic information for the device package and supports several different formats for providing this data. Even when transistor level buffer simulation is required, IBIS files can provide data needed to model package parasitics and map physical device pins to their corresponding transistor-level buffer models.

4.2 Pre-route signal integrity

Pre-route signal integrity analysis is used to study the effect of different physical layout strategies on flight times. A proposed routing scheme can be changed and re-simulated to optimize driver selection, termination location, termination value, and net physical length. Pre-route analysis should not be performed in isolation; it is most effective when preliminary layout assessments provide the high speed designer with estimated routing lengths and tolerances. Remember that high speed design is a tradeoff between static timing, signal integrity and physical design. It serves no purpose to study routing strategies that are physically unachievable. Pre-route SI is a powerful tool for assessing design tradeoffs and optimizing design performance – but only when everything is properly taken into perspective.

Pre-route SI typically simulates one net for each representative set of nets – for instance, in an interface with 32 address nets, pre-route SI will simulate a single net to represent the routing strategy for the whole address bus.

Flight times from pre-route SI analysis are plugged back into the interface's static timing model to compute timing margins for each transaction. Once acceptable timing margins are achieved, routing strategies used for each net are used to drive physical design. It's important to note that tighter layout constraints aren't necessarily better. Tight physical constraints may maximize timing margins, but they also may increase the cost and complexity of the physical design unnecessarily.

Consider the example from Figure 3. Timing analysis using first-order interconnect delays yielded 600ps of combined setup and hold margin. Let's further assume that pre-route signal integrity slew rates and min/typ/max behavior reduced margin by another 200ps, leaving 400ps of combined setup/hold margin. If we specify data lengths precisely, we will maximize timing margins – for instance, we might control data net lengths to +/- 50 mils. If we assume PCB delays of 180ps/inch, 100 mils of length variation subtracts about 18ps from our timing margin. If we relax the routing requirement for data lines to +/- 250 mils, we increase timing uncertainty to about 90ps, but the increased layout flexibility may enable a simpler (and cheaper) physical design. The designer must make the correct tradeoff based on their specific design goals.

Once the designer has defined a routing strategy for the interface, design rules (pin ordering, segment lengths, length matching) are communicated to the PCB designer. Specific rule formatting is dependent on the PCB layout tool being used. Design rule check (DRC) capabilities of the CAD system should always be used to ensure the design rules are followed.

4.3 Post-route SI

In theory, if the PCB is routed according to the strategies defined by pre-route SI (including spacing rules for crosstalk), then confidence is high that the design will function as intended. Unfortunately, while it's easy to study a single interface in isolation, it's difficult to predict how physical design requirements for multiple interfaces will conflict when tight routing tolerances are required. Conflicts arise that require adjusting routing strategies, re-running pre-route SI to assess impact, and then updating physical design rules. This is an iterative process that usually requires multiple cycles before the PCB layout "settles down". High speed design updates must occur quickly and efficiently as the board is being laid out to provide the PCB designer with quick feedback.

Post-route SI analysis also helps optimize setup and hold margins. Even when routing tolerances are tightly controlled, small differences in PCB parasitics can alter desired relationships between data/control signals and associated clocks. Integrated post-route SI and timing analysis identifies corrections needed to balance setup and hold margins. The corrections may be small, but they may be important, depending on interface margins.

Post-route SI analysis needs to occur as quickly as possible during PCB layout – an overnight run is OK, but analysis requiring several days is not, as the board will have changed by then. Post-route SI should leverage simulation work done during pre-route. The same devices and models should be applicable to both pre and post-route analysis.

Post-route design analysis should compute setup/hold margins and signal quality metrics for all interfaces / transactions in the system. This validates that timing closure has been achieved before the design is released to fabrication.

5.0 Reusing High Speed Analysis

Pre-and post-route SI are really just two sides of the same coin. Pre-route SI studies each net class to predict interconnect delay based on the net class' planned routing strategy. Post-route SI simulates all the nets in the net class based on how they were physically implemented – but the output of post-route SI is still interconnect delay, albeit many values instead of a single one.

Post-layout analysis should use the same simulation models, setups and timing equations as pre-layout analysis. Ideally, we would have an automated way of mapping the single net used during pre-layout to the multiple corresponding nets in the physical design. We would then simulate those nets using the same simulation models and setup information from pre-layout, and plug the results back into the timing model.

Mapping a pre-layout net to its post-layout counterparts allows reuse on a broader scale. Interfaces rarely get designed just once; many designs are derivatives and extensions of earlier work. If we can use an automated technique to perform pre- to post-route mapping for our first design, we can use the same pre-route setup for the second and subsequent implementations. This approach allows us to speed subsequent designs and amortize the effort needed to set up the first analysis.

6.0 DDR2 Example

DDR2 presents a number of unique signal integrity requirements in addition to the usual challenge of achieving system design closure. These issues include proper modeling of on-die termination (ODT), adjusting measured flight time according JEDEC DDR2 slew rate specifications, and analyzing different possible memory combinations.

6.1 On-die terminations

DDR2 memories use a dynamic termination scheme for data nets. The location and value of termination changes during device operation. Termination depends on how many memory devices are present, which device is driving, and which device is receiving. A buffer simulation model therefore requires several different states – driving, receiving with termination disabled, and receiving with termination enabled.

Each configuration of driving and receiving devices will have corresponding simulation model settings. This presents another level of simulation complexity – instead of having one set of model assignments for the data net, model assignments are now state-dependent. The high speed designer must have a strategy for handling this – either by simulating multiple versions of the net (each with its own set of model assignments), or by being able to define different simulation states and model assignments and then having those assignments managed automatically.

The table below (Figure 4) shows recommended ODT locations and values for a 2 slot memory write. Note that different types of memory cards can be plugged into each slot, so the termination depends not only on which slot is receiving, but the type of memory module plugged into that slot as well.

Write Configurations						
Configuration	Write to	DQ Active-Term Resistance				
		Controller	Dram at Slot 1		Dram at Slot 2	
			Front Side	Back Side	Front Side	Back Side
2R / 2R	Slot 1	No Term	No Term	No Term	50 or 75 ohm	No Term
	Slot 2	No Term	50 or 75 ohm	No Term	No Term	No Term
2R / 1R	Slot 1	No Term	No Term	No Term	50 or 75 ohm	Empty
	Slot 2	No Term	50 or 75 ohm	No Term	No Term	Empty
1R / 2R	Slot 1	No Term	No Term	Empty	50 or 75 ohm	No Term
	Slot 2	No Term	50 or 75 ohm	Empty	No Term	No Term
1R / 1R	Slot 1	No Term	No Term	Empty	50 or 75 ohm	Empty
	Slot 2	No Term	50 or 75 ohm	Empty	No Term	Empty
2R / Empty	Slot 1	No Term	150 ohm	No Term	Empty	Empty
Empty / 2R	Slot 2	No Term	Empty	Empty	150 ohm	No Term
1R / Empty	Slot 1	No Term	150 ohm	Empty	Empty	Empty
Empty / 1R	Slot 2	No Term	Empty	Empty	150 ohm	Empty

Figure 4 - ODT configurations for 2 slot DDR2 memory

6.2 Slew rate derating

Device inputs are usually considered to have “switched” when they pass through specified voltage levels (VIL, VIH). These voltage levels relate to how parts are tested in manufacturing. Production tests are performed with controlled slew rates. An actual input signal that switches faster or slower than the reference slew rate will cause the input buffer to switch earlier or later than the reference condition, with a corresponding (but normally uncharacterized) effect on the device’s timing characteristics.

DDR2 is the first memory standard to address the issue of signal propagation time through the receiver input buffer. The DDR2 standard specifies adjustments to the signal’s measured flight time based on the slew rate at the device input pin. Depending on whether the slew rate is faster or slower than the reference, the adjusted flight time can be longer or shorter than the original measurement.

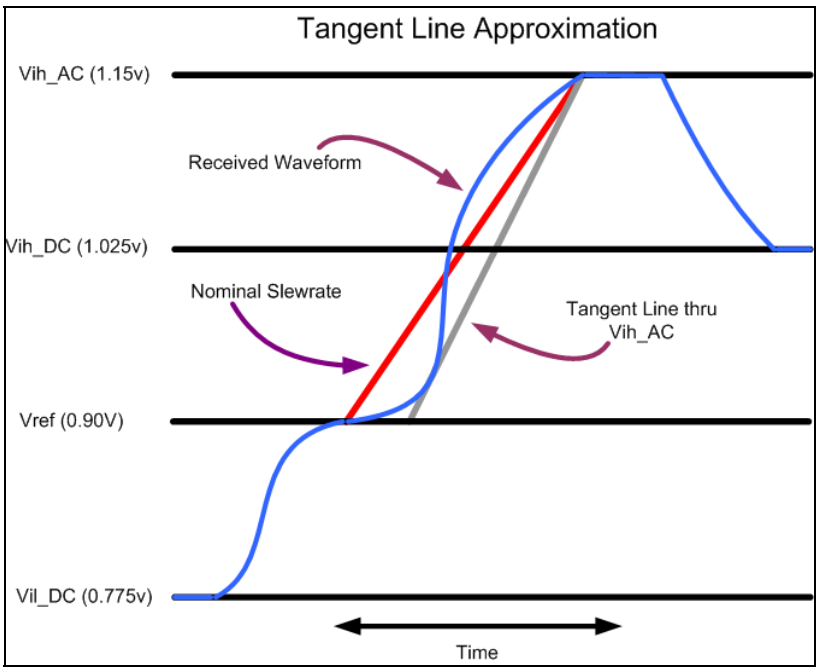

Figure 5 - DDR2 Slew rate derating

Adjustments to flight times can be made by performing simulations and then manually inspecting waveforms, or the process can be automated. Given the hundreds or thousands of simulations that may be required to validate a DDR2 interface, automation of this process is highly desirable.

6.3 Memory population support

Memory "populations" are another significant challenge facing the high speed designer. Let's consider a system with two Small Outline DIMM (SoDIMM) memory slots. SoDIMM modules come in different configurations, depending on overall module capacity, along with individual component width and depth. JEDEC currently publishes reference designs ("raw cards") for six different module types as shown in Figure 6.

DDR2 SoDimm Raw Card	SDRAM Width	# SDRAMs	# Ranks
A	x16	8	2
B	x8	8	1
C	x16	4	1
D	x8 (stacked)	16	2
E	x8	16	2
F	x8	16	2

Figure 6 - Reference DDR2 SoDIMM designs

There are 48 different ways to combine these six card types in a two slot system, as shown in the following table (Figure 7):

Population Name	Slot 1	Slot 2
mb_2slot_RCA_Empty	RCA	Empty
mb_2slot_RCA_RCA	RCA	RCA
mb_2slot_RCA_RCB	RCA	RCB
mb_2slot_RCA_RCC	RCA	RCC
mb_2slot_RCA_RCD	RCA	RCD
mb_2slot_RCA_RCE	RCA	RCE
mb_2slot_RCA_RCF	RCA	RCF
mb_2slot_RCB_Empty	RCB	Empty
mb_2slot_RCB_RCA	RCB	RCA
mb_2slot_RCB_RCB	RCB	RCB
mb_2slot_RCB_RCC	RCB	RCC
mb_2slot_RCB_RCD	RCB	RCD
mb_2slot_RCB_RCE	RCB	RCE
mb_2slot_RCB_RCF	RCB	RCF
mb_2slot_Empty_RCA	Empty	RCA
mb_2slot_Empty_RCB	Empty	RCB

Population Name	Slot 1	Slot 2
mb_2slot_RCC_Empty	RCC	Empty
mb_2slot_RCC_RCA	RCC	RCA
mb_2slot_RCC_RCB	RCC	RCB
mb_2slot_RCC_RCC	RCC	RCC
mb_2slot_RCC_RCD	RCC	RCD
mb_2slot_RCC_RCE	RCC	RCE
mb_2slot_RCC_RCF	RCC	RCF
mb_2slot_RCD_Empty	RCD	Empty
mb_2slot_RCD_RCA	RCD	RCA
mb_2slot_RCD_RCB	RCD	RCB
mb_2slot_RCD_RCC	RCD	RCC
mb_2slot_RCD_RCD	RCD	RCD
mb_2slot_RCD_RCE	RCD	RCE
mb_2slot_RCD_RCF	RCD	RCF
mb_2slot_Empty_RCC	Empty	RCC
mb_2slot_Empty_RCD	Empty	RCD

Population Name	Slot 1	Slot 2
mb_2slot_RCE_Empty	RCE	Empty
mb_2slot_RCE_RCA	RCE	RCA
mb_2slot_RCE_RCB	RCE	RCB
mb_2slot_RCE_RCC	RCE	RCC
mb_2slot_RCE_RCD	RCE	RCD
mb_2slot_RCE_RCE	RCE	RCE
mb_2slot_RCE_RCF	RCE	RCF
mb_2slot_RCF_Empty	RCF	Empty
mb_2slot_RCF_RCA	RCF	RCA
mb_2slot_RCF_RCB	RCF	RCB
mb_2slot_RCF_RCC	RCF	RCC
mb_2slot_RCF_RCD	RCF	RCD
mb_2slot_RCF_RCE	RCF	RCE
mb_2slot_RCF_RCF	RCF	RCF
mb_2slot_Empty_RCE	Empty	RCE
mb_2slot_Empty_RCF	Empty	RCF

Figure 7 - DDR2 SoDIMM population options

The question the system designer has to answer– which of these configurations will be supported, and how will those configurations be validated? Each card type presents different loading characteristics to the system. This causes different signal integrity behavior with a corresponding impact on system timing. If all the supported configurations require their own signal integrity simulations, this creates another layer of analysis data the high speed designer must create and manage. Note that we've only mentioned JEDEC "raw card" designs. If the analysis needs to include specific manufacturer's modules and variations due to use of different memory components, then the analytical problem becomes even more complex.

Here again, automation is the key. No individual simulation is difficult by itself – creating and simulating the topology for a data net based on a particular raw card is neither particularly challenging nor time consuming. Creating simulations for all supported combinations of cards and memory devices across operating corners is a huge task, both from a simulation resource and data management standpoint. This is the type of problem that lends itself well to automation – the creation and management of a large set of data.

7.0 Summary

Signal integrity is critically important, but design decisions that optimize signal integrity must be balanced against timing margins and impacts on system complexity / cost. Hopefully, we've shed some light how signal integrity analysis relates to these other elements of a high speed design methodology.

We believe a good high speed system design methodology helps designers find compromises between conflicting goals – the system needs to be low cost, but reliable with good signal integrity – performance needs to be high, but PCB routing layers should be minimized, and so on. An integrated approach to high speed design gives the designer information to make the right decisions for their specific design goals. The high speed designer's ultimate question is – will this work, and with how much margin? A methodology that determines design margin for different scenarios allows the designer to evaluate design tradeoffs effectively.

A good high speed design methodology has three key attributes – it is rigorous, repeatable and reusable. A rigorous process takes both static timing and signal integrity into account, predicting interface margin and optimizing design guidelines as a result. A repeatable process gives the same answer each time the same question is asked. This may sound trite, but a process that includes many manual steps is prone to error, and repeatability suffers as a direct consequence. Automation is key to ensuring repeatability in complex, multi-step processes. Finally, reusability implies that once the analysis has been setup and performed for one design, that setup should be adaptable to future designs with minimum effort.

High speed system design is fundamentally a balancing act, and a good high speed design methodology provides the information needed to quickly evaluate design alternatives. Simply put, a good high speed design methodology serves the designer, not the other way around.