CDNLive! Paper – Signal Integrity (SI) for Dual Data Rate (DDR) Interface

Prithi Ramakrishnan
iDEN Subscriber Group
Plantation, Fl

Presented at

cadence designer network

cdn LIVE™
Silicon Valley 2007

Introduction

The need for Signal Integrity (SI) analysis for printed circuit board (PCB) design has become essential to ensure first time success of high-speed, high-density digital designs. This paper will cover the usage of Cadence's Allegro PCB SI tool for the design of a dual data rate (DDR) memory interface in one of Motorola's products. Specifically, this paper will describe the following key phases of the high-speed design process:

- Design set-up
- Pre-route SI analysis
- Constraint-driven routing
- Post-route SI analysis

DDR interfaces, being source synchronous in nature, feature skew as the fundamental parameter to manage in order to meet setup and hold timing margins. A brief overview of source synchronous signaling and its challenges is also presented to provide context.

Project Background

This paper is based on the design of a DDR interface in an iDEN Subscriber Group phone that uses the mobile Linux Java platform. The phone is currently in the final stages of system and factory testing, and is due to be released in the market at the end of August 2007 for Nextel international customers. The phone has a dual-core custom processor with an application processor (ARM 11) and a baseband processor (StarCore) running at 400MHz and 208MHz respectively. The processor has a NAND and DDR controller, both supporting 16-bit interfaces. The memory device used is a multi-chip package (MCP) with stacked NAND (512Mb) and DDR (512Mb) parts. The NAND device is run at 22MHz and the DDR at 133MHz. The interface had to be supported over several memory vendors, and consequently had to account for the difference in timing margins, input capacitances, and buffer drive strengths between different dies and packages.

As customer preference for smaller and thinner phones grows, the design and placement of critical components and modules has become more challenging. In addition to incorporating various sections such as Radio Frequency (RF), Power Management, DC, Audio, Digital ICs, and sub-circuits of these modules, design engineers must simultaneously satisfy the rigid placement requirements for components such as speakers, antennas, displays, and cameras. As such, there are very few options and little flexibility in terms of placement of the components. This problem was further accentuated by the fact that several layers of the 10 layer board (3-4-3 structure with one ground plane and no power planes) were reserved for power, audio, and other high frequency (RF) nets, leaving engineers with few layers to choose from for digital circuitry.

Figure 1. Memory Interface routes

With the DDR interface data switching at 266MHz, we had very tight margins — 600ps for data/DQS lines, 280ps for the address lines, and 180ps for control lines. However, with the NAND interface we had larger margins that were on the order of a few tens of nanoseconds. In these situations, choosing a higher drive strength and using terminators of appropriate values (to meet rise times and avoid overshoot/undershoot) has become a common practice in DDR designs. However, due to the lack of space on the board, we were not in a position to use terminators. Therefore, we used programmable buffers on our processor, and with the help of Cadence SI tools were able to fine-tune the design.

Our group migrated from using Mentor Graphics to Cadence SI during this project. As one might expect, this made the task of designing a high speed DDR interface even more challenging. To help overcome this, we worked extensively with Cadence Services, where Ken Willis supported us on the SI portion of the design.

The Source Synchronous Design Challenge

Before discussing the specifics of the Motorola DDR interface, a brief overview of source synchronous signaling is provided here for context. Historically, digital interfaces have utilized “common clock” signaling, as shown in the figure below.

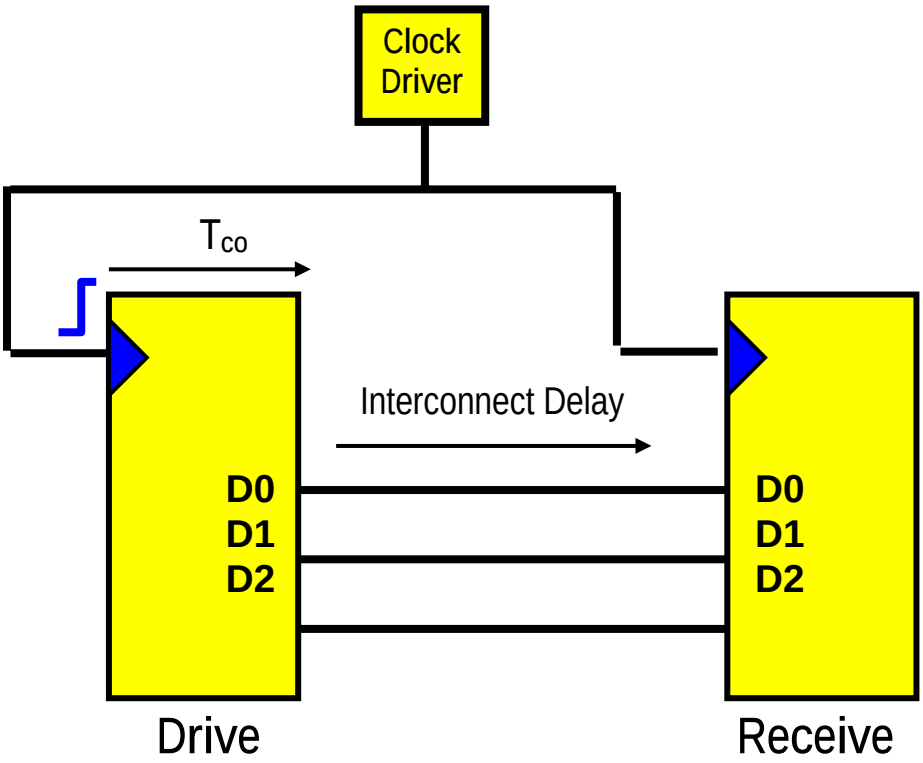


Figure 2. Common clock design

With common clock interfaces, the clock signal is provided to the driving and receiving components from an external component. The magnitude of the driver's Tco (time from clock to output valid) and the interconnect delay between the driving and receiving components becomes a limiting factor in the timing of the interface. From a practical standpoint, it becomes increasingly challenging to implement interfaces of this type above several hundred megahertz.

In order to accommodate requirements for faster data rates, source synchronous signaling emerged as the new paradigm. This is illustrated in the figure below.

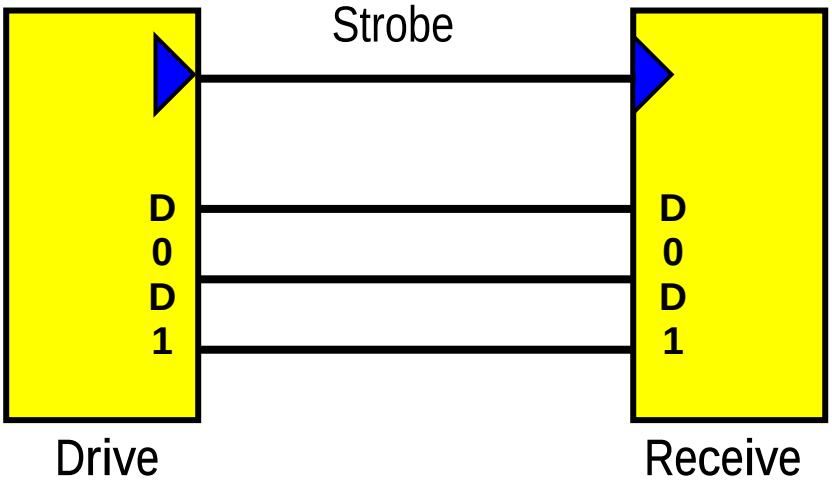


Figure 3. Source synchronous design.

In a source synchronous interface, the “clock” is provided locally by the driving component, and is generally called a “strobe” signal. The relationship between the strobe and its associated data bits is known as it leaves the driving component, with setup and hold margins pre-established as the signals are put onto the bus.

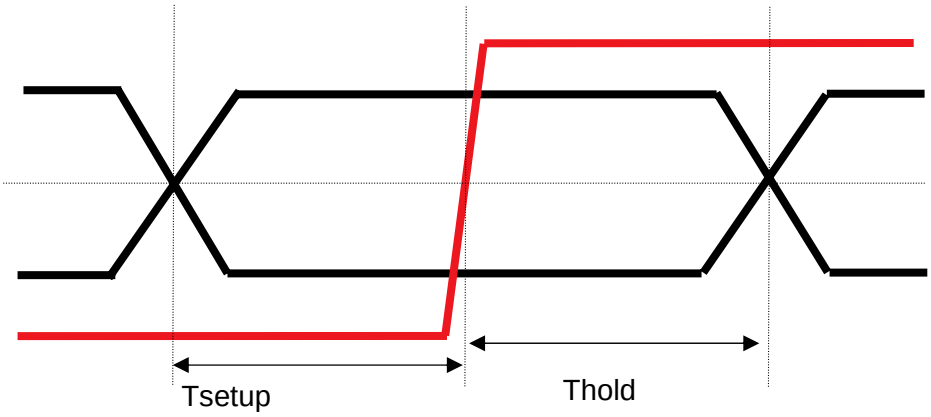


Figure 4. Timing diagram.

This essentially takes the driver’s Tco as well as the magnitude of the interconnect delay between the driving and receiving chip out of the timing equation altogether. The timing challenge then becomes to manage the skew between the data and strobe signals such that the setup and hold requirements at the receiving end are still met.

Technical Approach

The general technical approach used in this project can be broken down into the following key phases of the high-speed design process:

- Design set-up
- Pre-route SI analysis
- Constraint-driven routing
- Post-route SI analysis

First the PCB design database is set up to enable analysis with Allegro PCB SI. Before routing is performed, initial trade-offs are examined at the placement stage, and constraints are captured to facilitate constraint-driven routing. When routing is completed, detailed analysis is performed, interconnect delays extracted, and setup/hold margins are computed. Any adjustments required are fed back to the layout designer, and the post-route analysis is repeated. This basic process is diagrammed below.

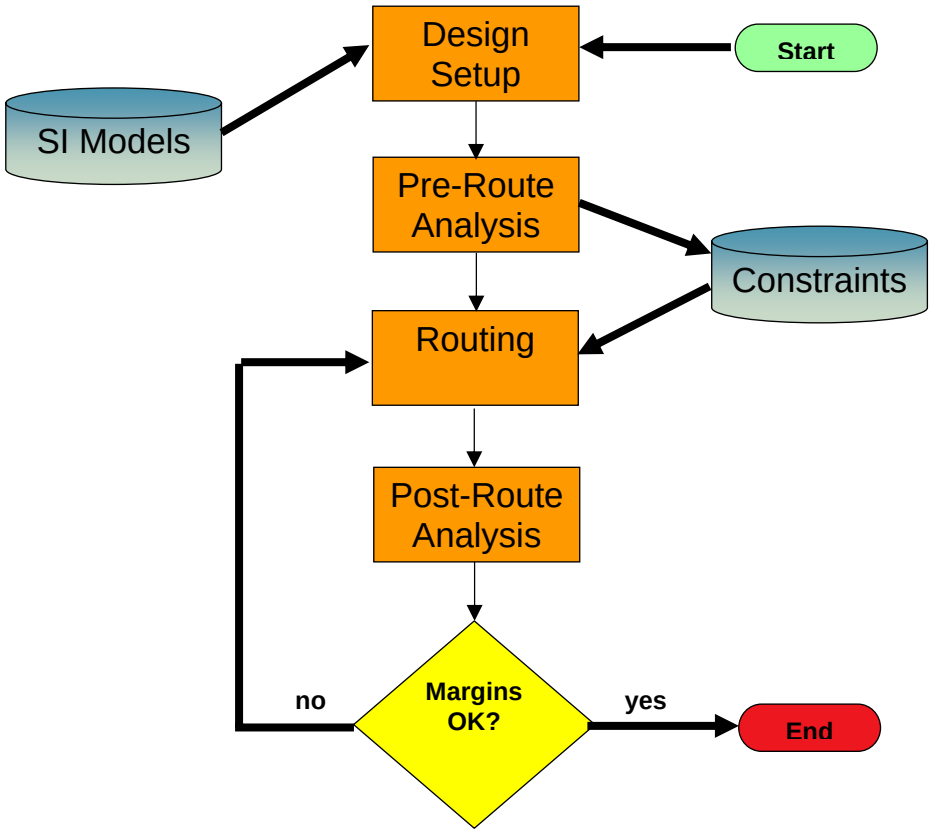


Figure 5. SI design process flow.

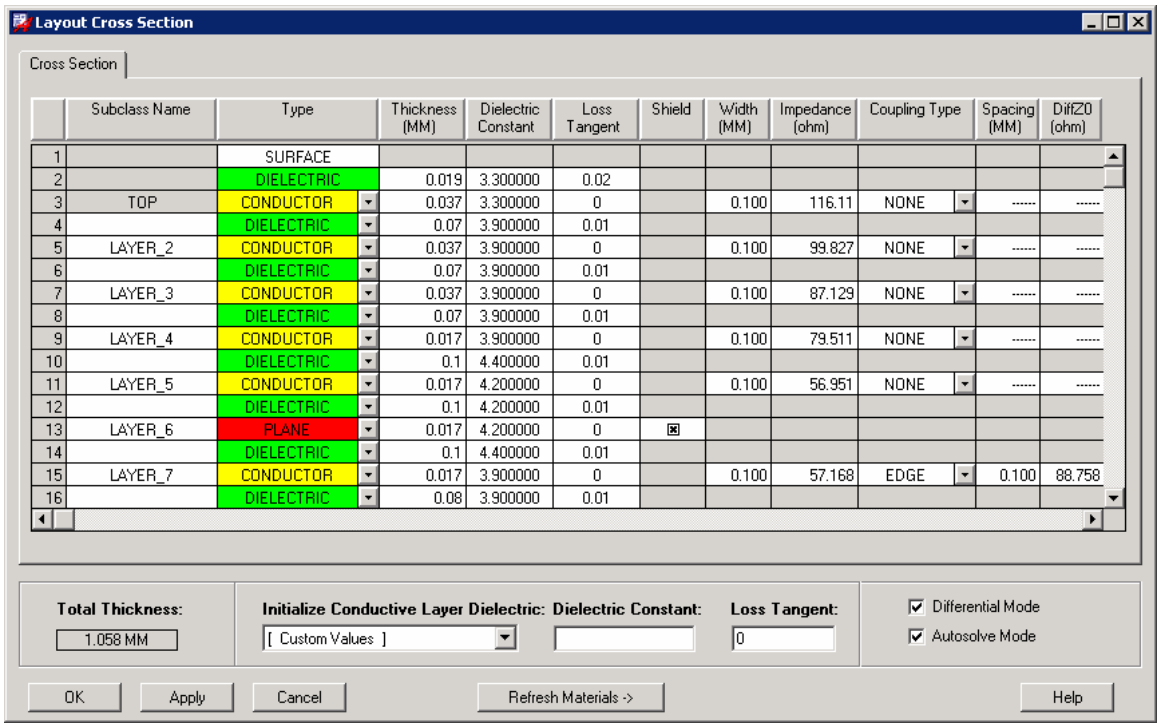
Detail on the major design phases are provided in the subsequent sections.

Design Setup

By virtue of its direct integration with the Allegro PCB layout database, Allegro SI analysis requires that the design be set up to facilitate the automated extraction, circuit building, netlisting, simulation, and analysis that it performs. This essentially means adding the needed intelligence to the physical Allegro database that allows the tool to do its job. This setup involves the following:

- Cross section
- DC nets
- Device definitions
- SI models

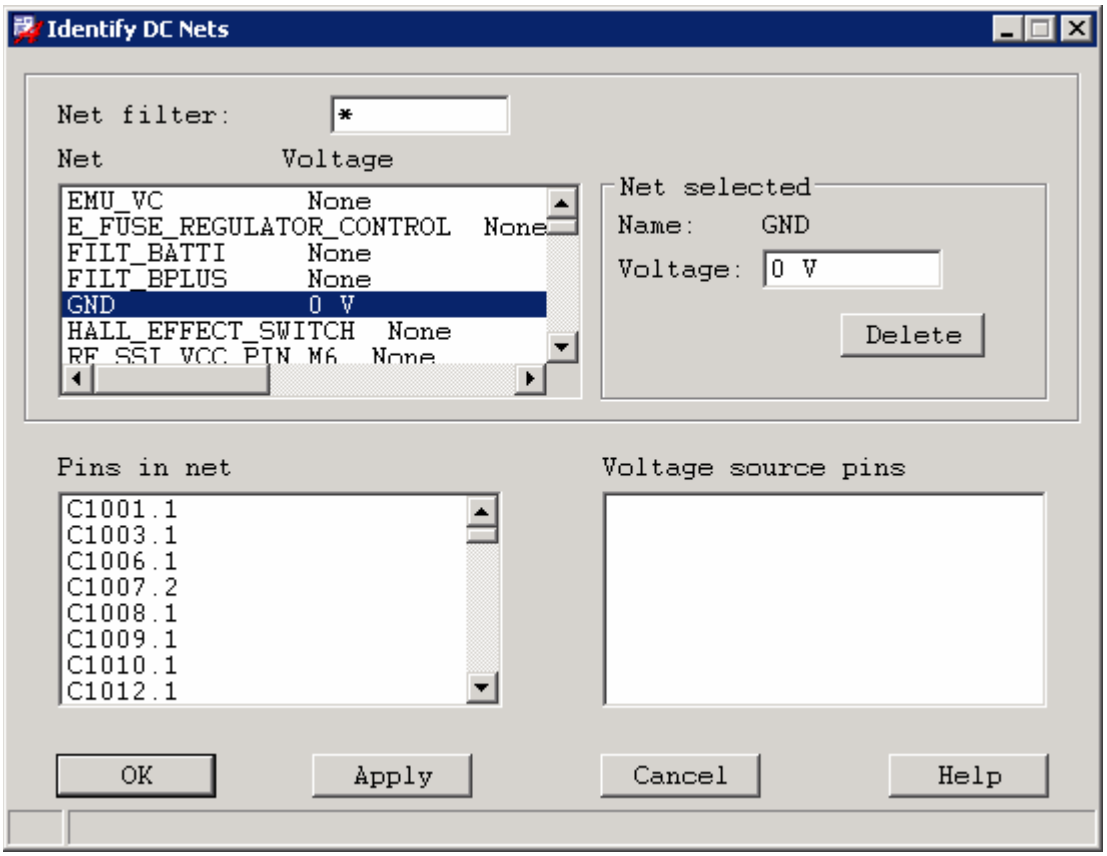
By definition, SI analysis involves the modeling of interconnect parasitics. In order to do this accurately, the tool needs to know the properties and characteristics of the materials used in the PCB stack-up. This information is defined in the Cross Section form, as shown below.



It is crucial to get this data correct, as it will be fed to the 2D field solver to model interconnect parasitics during the extraction process. The best source for this detailed information is generally from the PCB fabricator. Layer thickness, dielectric constant, and loss tangent are all critical parameters for the cross section definition.

In order for circuit extraction to be done properly, the tool needs to know about DC nets in the design, and what their associated voltage levels are. This accomplishes two main things in the setup; a) enables voltage sources to be injected properly in the extracted circuits, and b) avoids having the tool needlessly trying to extract extremely large DC nets, and hanging up the analysis process. Take the example of a parallel resistor termination. Allegro SI will encounter the resistor as it walks the signal net to be extracted. The tool will look up the SI model assigned to this resistor, splice in the resistor subcircuit, and continue extracting whatever is on the other side of the resistor. If this is a large DC net (ex. VTT), the desire is for the tool to put a voltage source at the 2nd resistor pin, complete the circuit, and simulate the signal. To do this properly, the tool relies on a VOLTAGE property to exist on the DC net, with a numeric value defined. In the absence of the VOLTAGE property, the tool will simply continue to extract, which in the case of a 2000 pin ground net, would be a large waste of computational time.

To identify DC nets, clicking “Logic > Identify DC Nets” will spawn the following form.



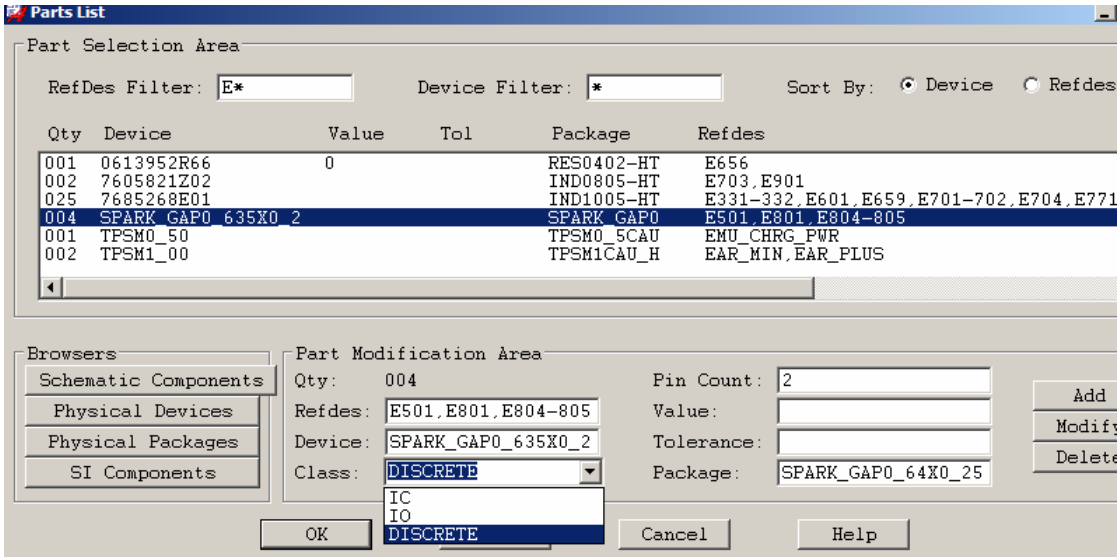
All DC nets in the design should be identified, to fully optimize SI analysis. These can be identified up front in the schematic, as well as in the physical layout as shown here.

The next step in the design set-up process is to verify that the logical “CLASS” and “PINUSE” attributes for the devices in the design are defined appropriately. These attributes originate from the schematic symbol libraries and are passed into the Allegro physical layout environment. In an ideal methodology, these libraries would be defined properly and would require no edits. However, this is not always the case, and as these attributes have a bearing on the behavior of the SI analysis, it is worth mention here.

The “CLASS” attribute is used to distinguish between different types of components in the PCB design. Legal values of “CLASS” are listed below:

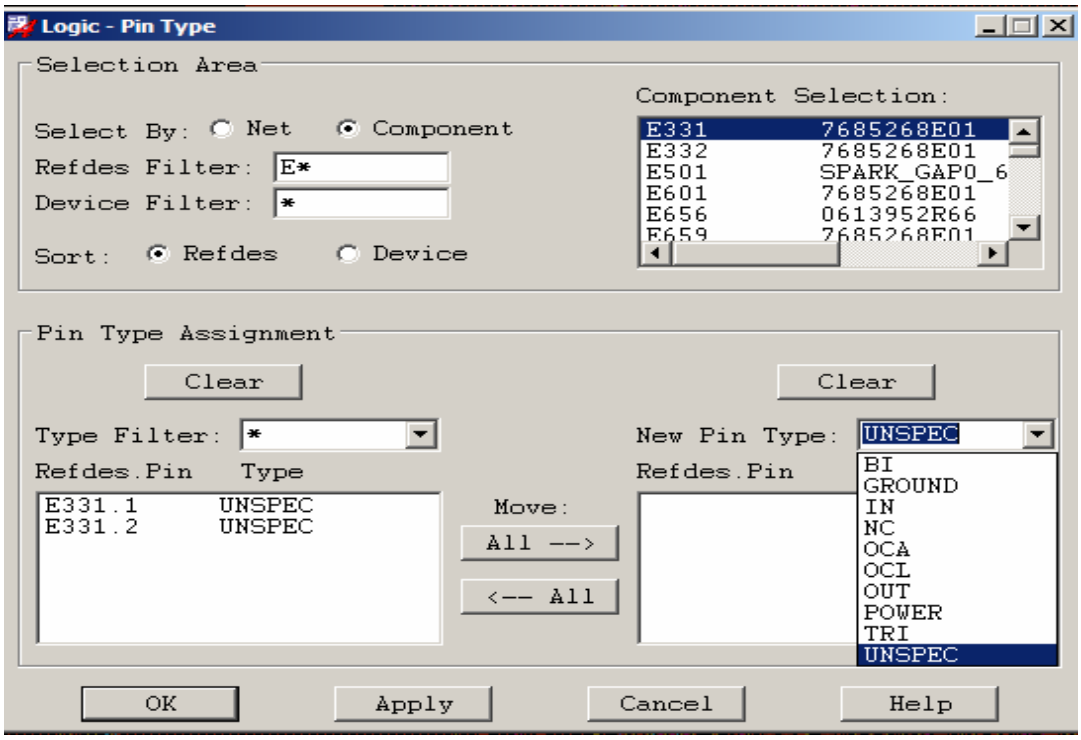
- IC – This is used for digital integrated circuits, which contain drivers and/or receivers. These types of components are modeled with an SI model of the type “IbisDevice”. When the automated circuit building algorithms in Allegro PCB SI encounter a model of this type, it looks up the buffer model (driver, receiver, or bidirectional) assigned to the pin in question, and inserts it into the circuit along with its associated package parasitics.
- IO – A component with CLASS = IO is intended for components that connect off-card to other physical layout designs, such as connectors. These components can be associated with a “DesignLink”, which provides netlisting to other physical designs and enables multi-board SI analysis. So circuit building algorithms expect to jump from a device of CLASS=IO to a similar device on a different physical layout.
- DISCRETE – For devices of this class, circuit building algorithms expect to traverse “through” the component, from one pin to another, inserting a subcircuit in-between. A good example of this would be a series resistor.

If CLASS attributes are not set up properly in the source schematic libraries, they can be edited in the physical layout database for analysis by using the form shown below, launched from the “Logic > Parts List” menu pick.

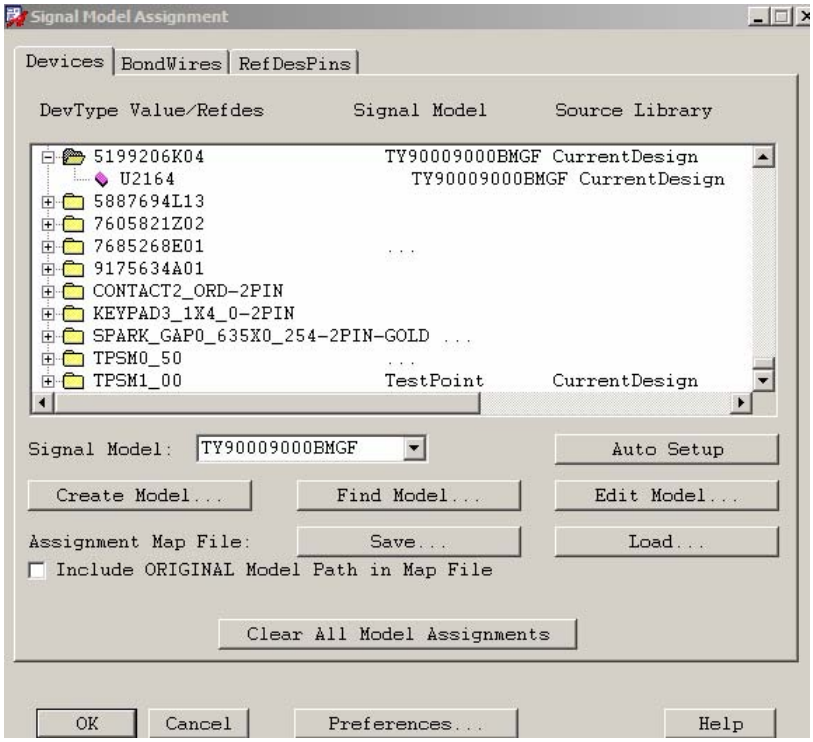


The “PINUSE” attribute also impacts the behavior of the SI analysis, as the tool uses this information to determine if a pin is a driver, receiver, bidirectional, or passive pin. As with the “CLASS” attribute, in an ideal methodology this is defined properly in the schematic libraries, and no editing is required in physical layout.

“PINUSE” can be modified in two main ways for SI purposes. The most straightforward way is to ensure that the IOCell models used in the IbisDevice models assigned to components have the appropriate Model Type for the signals they are associated to. When SI models are assigned to components, the tool will check for conflicts between the model and the PINUSE it finds for the component in the design, and will use the SI model to automatically override the PINUSE found in the drawing. So if the correct pin types are found in the SI models, the layout will automatically inherit those settings. For components not explicitly modeled, their PINUSE can be set using the form shown below, launched from the “Logic > Pin Type” menu pick.



Signal Integrity (SI) models can be assigned using the “Signal Model Assignment” form, shown below.



Upon clicking “OK” the selected models will be assigned to the components and saved directly in the layout database. As mentioned previously, “PINUSE” attributes will be synced up, with the SI models superseding attributes in the original layout drawing.

Pre-Route SI Analysis

Performing pre-route analysis is a key part of the high-speed design process. Once critical component placement has been done, Manhattan distances can be used to estimate trace lengths, and can provide a realistic picture of how routed interconnect will potentially perform.

Before simulations are run for critical signals, the timing of the interface must be well understood. To accomplish this, we will first sketch timing diagrams for each signal group and then extract a representative signal for analysis. Next, we will explore Z_0, layer assignments, drive strength, route lengths, spacing, and terminations for these nets.

To sketch the timing diagrams, we first analyze the memory interface. The memory interface consists of both DDR and NAND signals and has around seventy nets. To simplify the analysis of the interface, we first divide these nets based on function and then simulate one net from each group. Accordingly, we select one signal from each of the following groups — clock_ddr, strobe_ddr , data_ddr, control_ddr, address_ddr, control_nand, and data_nand — for our pre-route simulations.

To understand the timing relations in the interface, we should look at the following operations between the memory device and the processor — *read*, *write*, *address write*, and *control operations*. Next, we identify the nets involved and the clocking reference signal for each of these operations. We then calculate the worst case slack available from the setup and hold numbers available in the data sheets. In particular, we adopted the worst case numbers across four different memory vendors, to ensure robustness of the manfactured system in the field..

1. Read

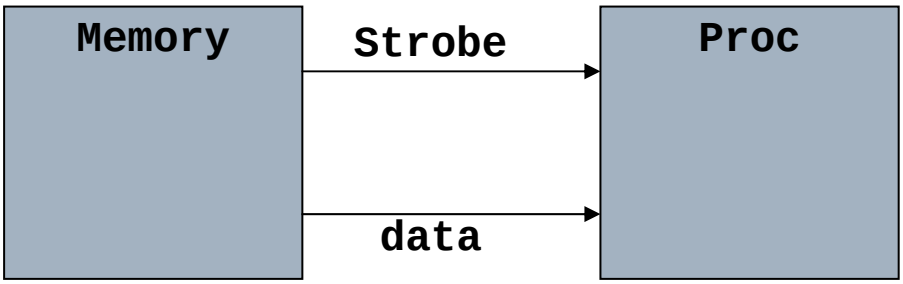


Figure 6. Read operation at memory interface.

During the read operation, the memory drives the data and DQS lines. The processor has a delay line (a series of buffers which can be tapped at different points), which is used to delay the DQS signal so that it samples the data at quarter of the cycle. The processor also offers programming options that allow us to apply an offset to the quarter cycle, enabling us to meet our setup and hold times. Hence, the processor self-corrects for strobe/data skew using this delay line. The granularity of this delay line is 30 ps; that is, each of the buffers of the delay line contributes 30 ps of delay. The data lines 0-7 are clocked with respect to the DQS0 strobe signal, and the data lines 8-15 are clocked with respect to DQS1. Data and strobe lines should be clustered, with the matching constraints determined by the write cycle.

2. Write

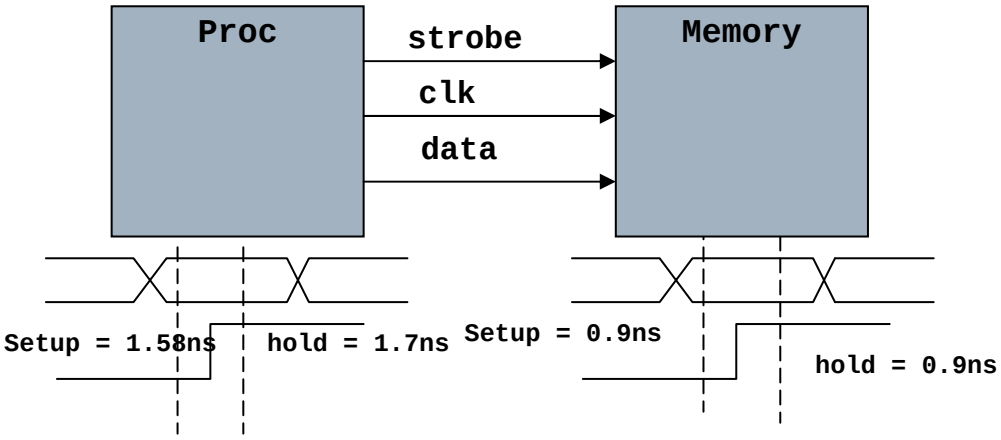


Figure 7. Write operation at memory interface.

During the write operation, both data and DQS are driven by the processor. Data is latched at both the positive and the negative edges of the DQS signals. Here again, data bits 0-7 are clocked by DQS0 and data bits 8-15 are clocked by DQS1. The setup and hold times available as these signals come out of the DDR controller are 1.58ns and 1.7ns respectively and the corresponding times required at the memory to ensure correct operation is 0.9ns. Hence, the slack available for routing is the lesser of 1.58ns – 0.9ns or 1.7ns – 0.9ns, which comes out to be 0.68ns. This amounts to an allowable ~85mm mismatch between the data lines. In addition, we need to make sure that length of the DQS lines is around the average of all the data lines. The data mask signals DQM0 and DQM1 also come into play during the write operation and we should group them along with the respective data lines.

3. **Address bus**

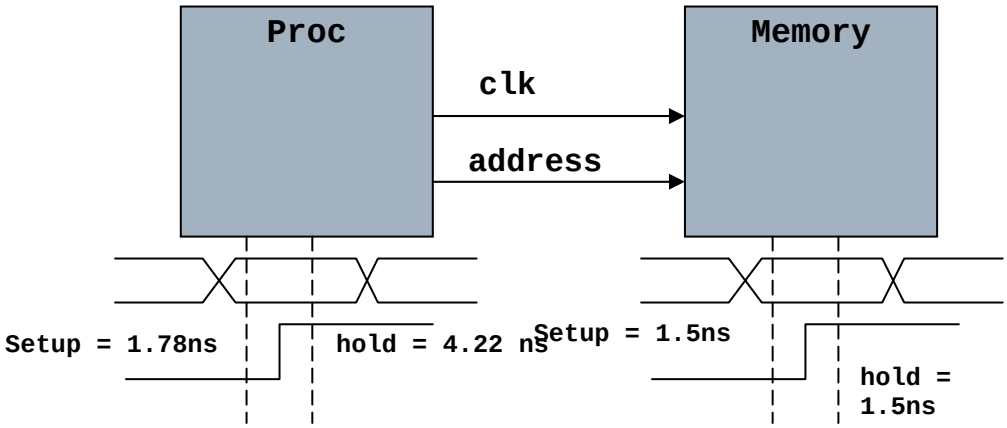


Figure 8. Address bus operation at memory interface.

Both address and clock lines are driven by the processor. The address bits 0-12 are clocked by the differential clock and latched at the positive edge of the clock. The setup and hold times available for these signals from the DDR controller are 1.78ns and 4.22ns respectively and the corresponding times required at the memory to ensure correct operation is 1.5ns for both. Hence the worst case slack for routing is 0.28ns and we have to try to match our signals to meet these numbers. The 0.28ns slack amounts to ~14mm mismatch between the address lines and the clock.

4. Control lines

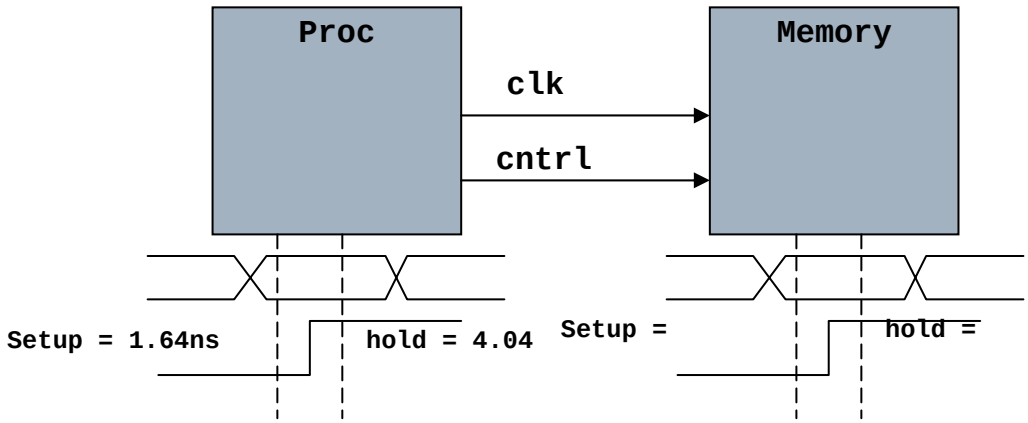


Figure 9. Control lines at memory interface.

The control signals are clocked by the differential clock and latched at the positive edge of the clock. The setup and hold times coming out of the DDR controller are 1.64ns and 4.04ns respectively. The setup and hold times required at the memory to ensure correct operation is 1.5ns. Hence, the worst case slack for routing is 0.14ns and we have to try and match our signals to meet these numbers. The 0.14 ns slack amounts to ~7mm mismatch between the control lines and the clock.

In addition, CLK to DQS skew is around 600 ps. With regards to the NAND lines, setup and hold numbers are in the order of tens of ns and hence routing them as short as possible based on their Manhattan lengths would suffice.

To complete pre-route analysis, SigXplorer must be setup for these tasks:

a. Extract a topology file for single net analysis. To bring up the net in SigXplorer, it is essential that the models are assigned, as described in Section 2, to each of the drivers, receivers, and components in the signal path.
b. Set up parameters for extraction and simulate using SigXplorer.
c. Perform measurements using SigWave

The following screenshots of SigXplorer show this process in detail.

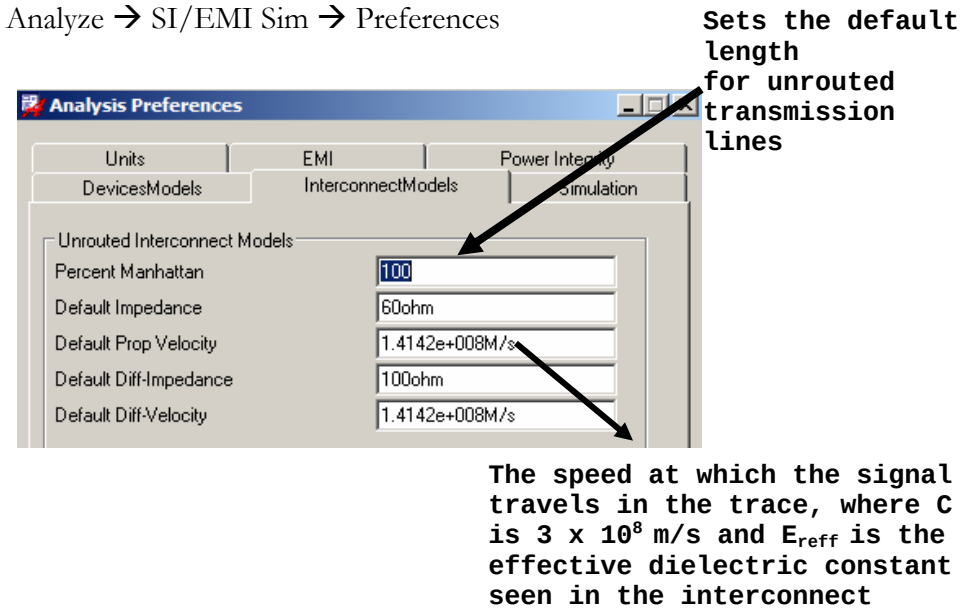
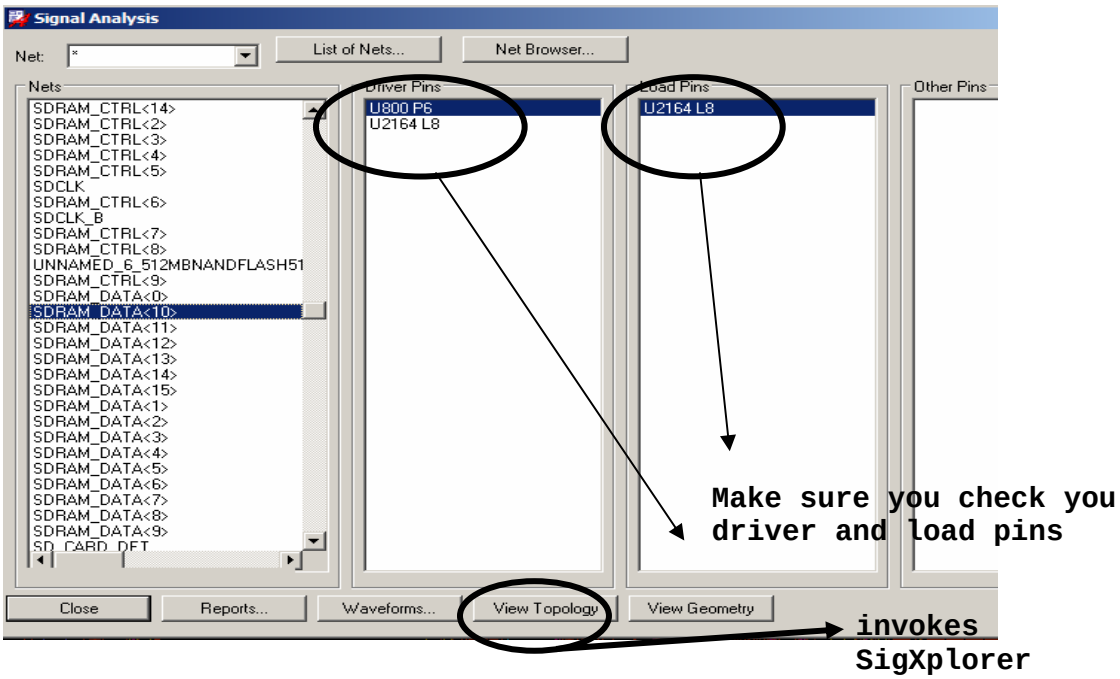


Figure 10. SigXplorer screenshots.

Since at this point none of the nets in the design are routed we need to set the percent Manhattan section for unrouted interconnect models. We should then select the net, as shown in the next screenshot, for analysis.

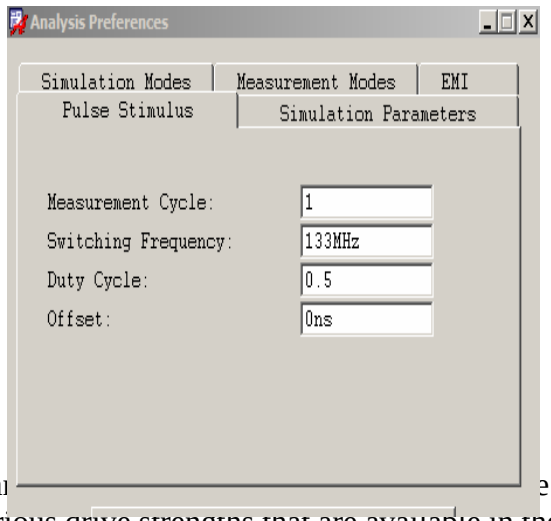
Analyze→ SI/EMI Sim →probe



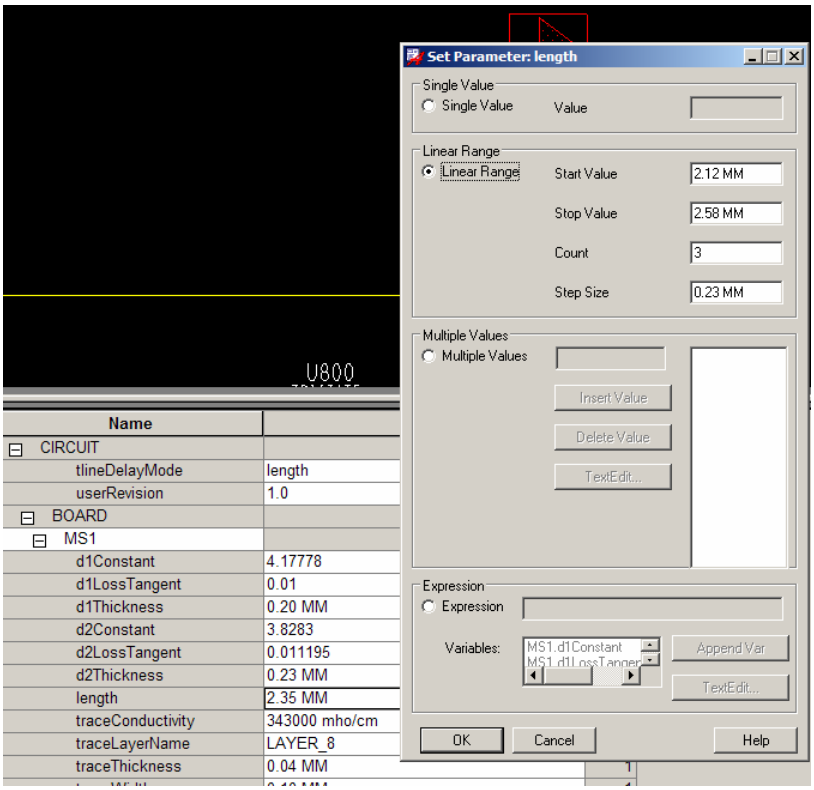
At this point, it is important to check if your driver and receiver pins are set correctly. The net chosen in the above example is a data net, it is bi-directional, hence it can be driven both by the memory device as well as the processor. The view topology icon can be clicked to export this net in SigXplorer.

The tool extracts the net along with drivers, receivers and strip lines on various layers of the board. Before you start the simulation, you must set the stimulus frequency, pulse step offset, and cycle count. This can be set in the following GUI.

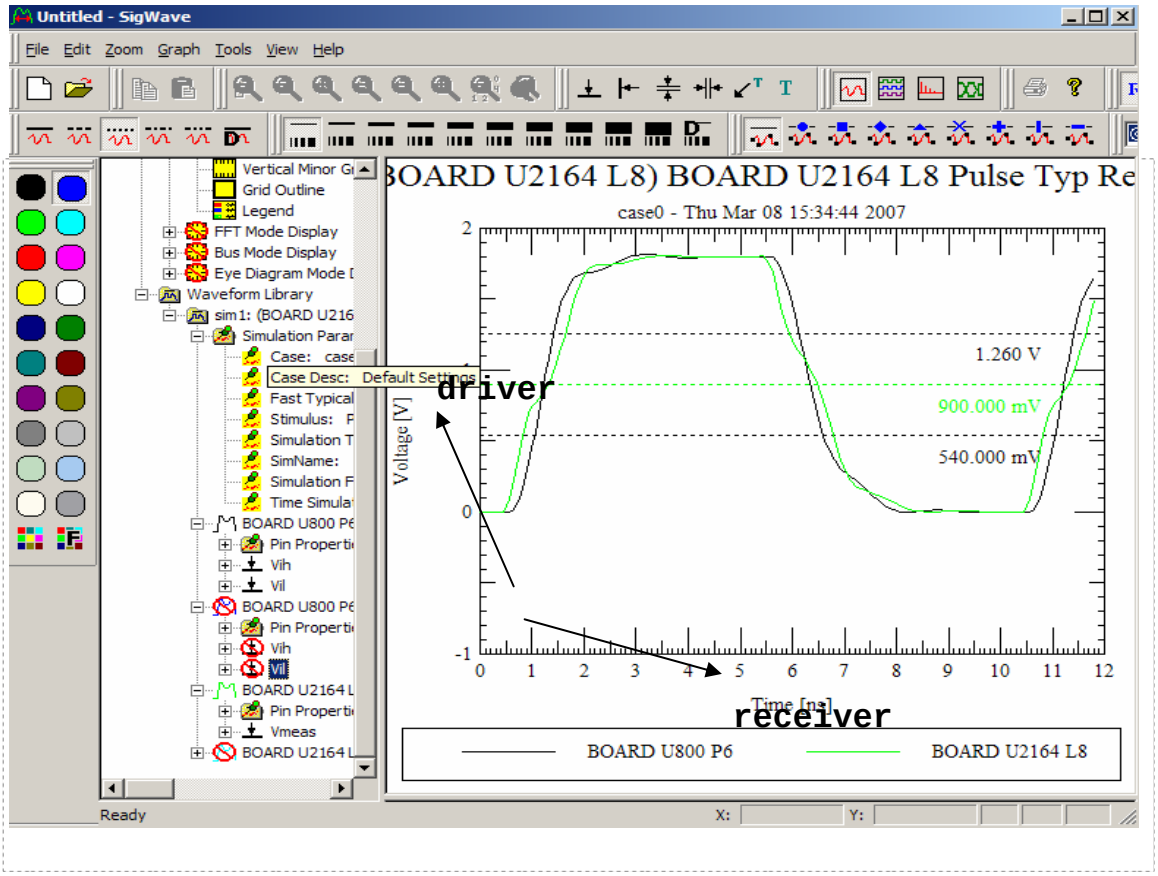
Analyze → Preferences



Both the memory device a e strengths. The buffer model can be changed to pick up the various drive strengths that are available in the dml models of the devices till we observe satisfactory waveforms in SigWave.



SigXplorer allows you to sweep any of the parameters such as the thickness, length, drive strengths and displays corresponding settle/switch delays, monotonicity, and glitch tolerance for the corresponding simulation. It also allows adding components such as resistors and capacitors and let's us sweep their values. We added a resistor in series with our clock in or to get rid of ringing in the rising edge. The tool let us determine what values were suitable for this resistor. As shown in the next figure the waveform corresponding to our simulation can be brought up on SigWave.



You can observe the rise/fall times, look for noise margins, overshoot/undershoot of the receiver waveform.

The constraints we develop in the pre-route simulation will be used by the routing tool to ensure correct first time results. This leads to our next section; Constraint-driven routing.

Constraint-driven routing

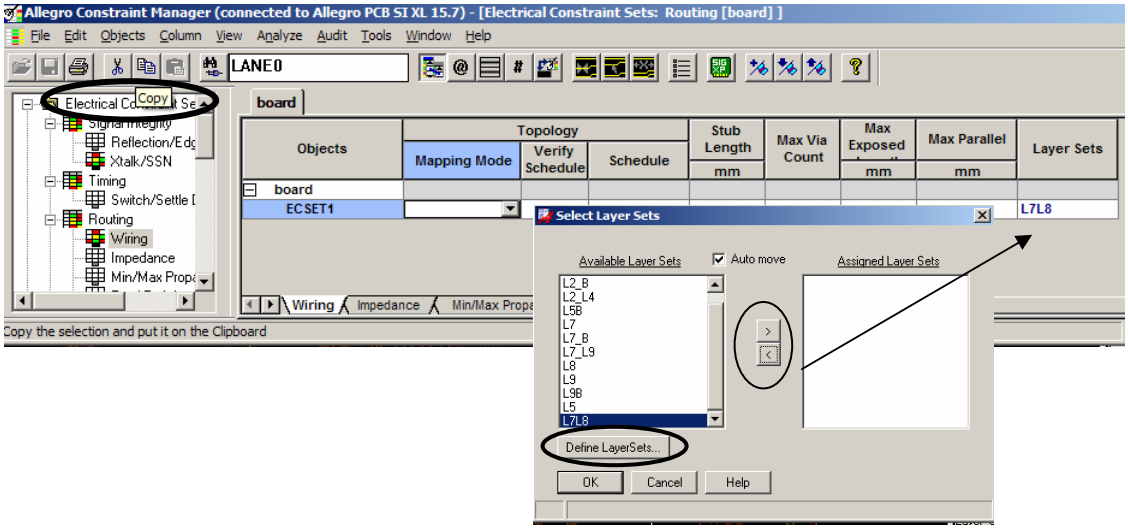
Once pre-route analysis has been done, and trade-offs have been examined, signal wiring constraints need to be developed to drive the constraint-driven routing process. With the DDR interface being point-to-point between the processor and memory, we translated our timing requirements into length constraints to make the routing as straightforward as possible. We also assigned layer constraints for our DDR signals. Both the length and the layer constraints can be directly applied to the constraint manager before the routing process starts.

For our particular design, we determined the following layer assignments from the results of the pre-route simulations, taking into account the layer's characteristic impedance per our stack-up:

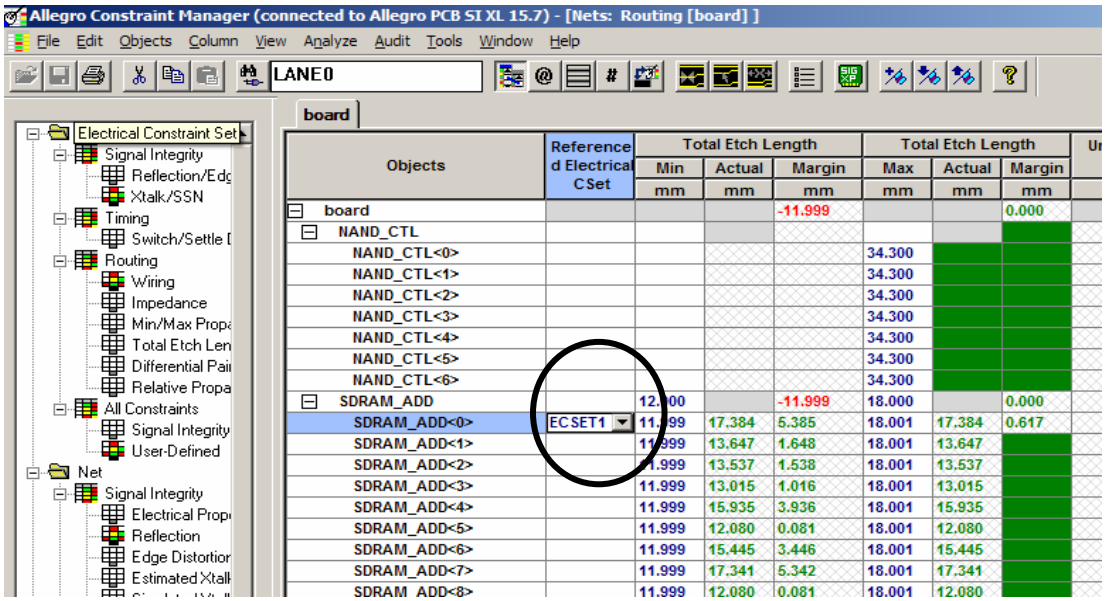
Layer 6 → ground plane
Layer 7 → clock, add, ctrl
Layer 8 → data, strobe
Layer 9 → NAND interface

Before we set up our design for auto-routing, we routed the differential clock lines manually on the layers closest to the ground plane. For the rest of the nets, the layer constraints can be created as shown in the following snapshots of the constraint manager.

Electrical Constraint Set → Wiring
Right click on board → Create new constraint
Name the constraint (ex. ECSET1)



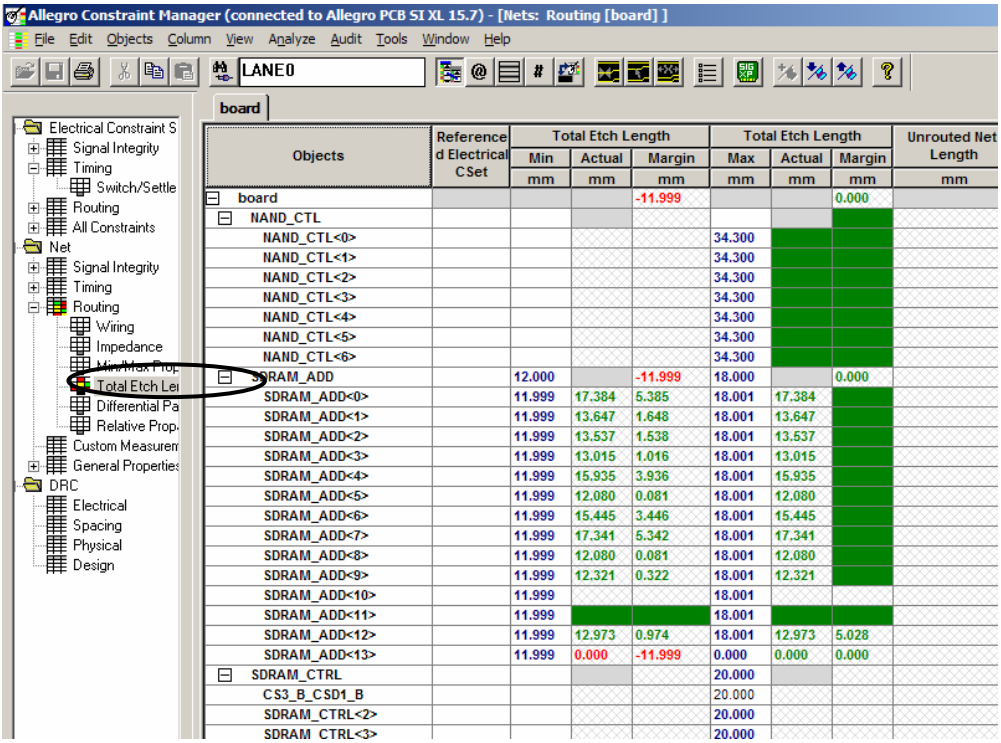
We choose one layer with horizontal orientation and one with vertical for each of our layer sets. You can form groups from the available layer sets and create a new constraint. This constraint, which we define as ECSET1, can be easily read back in the constraint manager and applied to the relevant net group, as shown in the following snapshot.



We determined from pre-route analysis the slack available for each of our net groups; however, before we translate these into length constraints it is important to get a report of the Manhattan lengths of each of these signals. To illustrate this, we will focus on the address signals. The Manhattan report of the address lines showed that the shortest lines were 6mm and the longest were 17mm. Accordingly, the minimum length constraint must be longer than 6mm and the maximum length constraint must be longer than 17mm. Additionally, from our timing diagrams, we determined that the maximum spread can be no more than 14mm. Following these restrictions, we set the minimum and maximum length limits for the address line are 11.99 mm

to 18.99 mm (shown in the constraint editor window below). Based on the layout designer's recommendations, we were able to constrain a bit tighter (7mm margin) and produce better margins.

To enter the length constraint, we open the Net → Routing →Total etch length section of the constrain manager. We followed this procedure for all the other net groups. The snapshot that follows shows length constraints associated with the address lines. Here, the key is to not to over-constrain your design, but at the same time have enough constraints so the timing and signal integrity parameters are met. Over-constraining the design severely inhibits the auto-router and may leave large portions of the design (as much as 90%) un-routed.



Objects	Referenced Electrical CSet	Total Etch Length Min (mm)	Total Etch Length Actual (mm)	Total Etch Length Margin (mm)	Total Etch Length Max (mm)	Total Etch Length Actual (mm)	Total Etch Length Margin (mm)	Unrouted Net Length (mm)
board				-11.999			0.000	
NAND_CTL								
NAND_CTL<0>					34.300			
NAND_CTL<1>					34.300			
NAND_CTL<2>					34.300			
NAND_CTL<3>					34.300			
NAND_CTL<4>					34.300			
NAND_CTL<5>					34.300			
NAND_CTL<6>					34.300			
SDRAM_ADD		12.000		-11.999	18.000		0.000	
SDRAM_ADD<0>		11.999	17.384	5.385	18.001	17.384		
SDRAM_ADD<1>		11.999	13.647	1.648	18.001	13.647		
SDRAM_ADD<2>		11.999	13.537	1.538	18.001	13.537		
SDRAM_ADD<3>		11.999	13.015	1.016	18.001	13.015		
SDRAM_ADD<4>		11.999	15.935	3.936	18.001	15.935		
SDRAM_ADD<5>		11.999	12.080	0.081	18.001	12.080		
SDRAM_ADD<6>		11.999	15.445	3.446	18.001	15.445		
SDRAM_ADD<7>		11.999	17.341	5.342	18.001	17.341		
SDRAM_ADD<8>		11.999	12.080	0.081	18.001	12.080		
SDRAM_ADD<9>		11.999	12.321	0.322	18.001	12.321		
SDRAM_ADD<10>		11.999			18.001			
SDRAM_ADD<11>		11.999			18.001			
SDRAM_ADD<12>		11.999	12.973	0.974	18.001	12.973	5.028	
SDRAM_ADD<13>		11.999	0.000	-11.999	0.000	0.000	0.000	
SDRAM_CTRL					20.000			
CS3_B_CSD1_B					20.000			
SDRAM_CTRL<2>					20.000			
SDRAM_CTRL<3>					20.000			

Post-Route SI Analysis

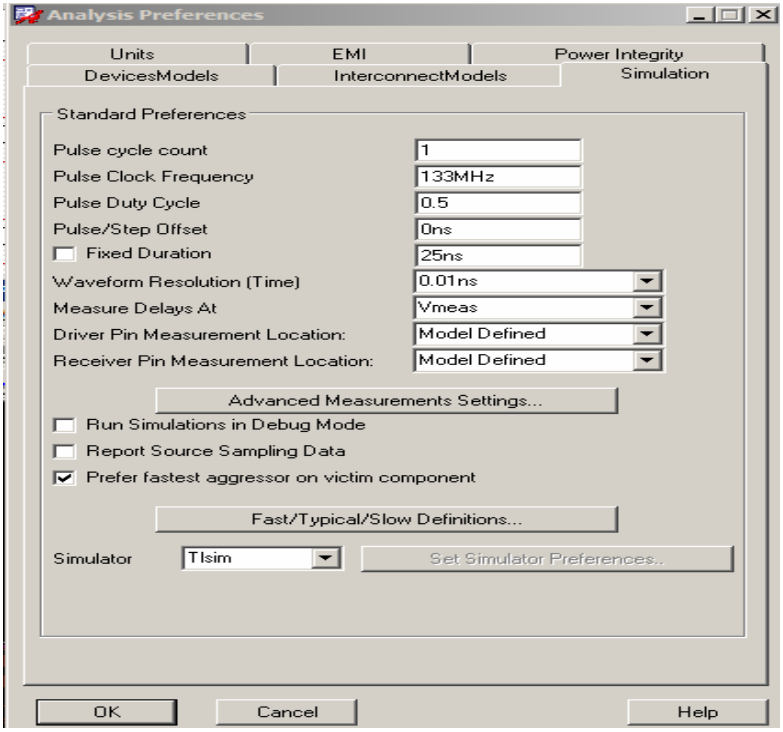
Once the design is fully routed, detailed simulations can be run for post-route verification. The goal at this phase is to determine final margins over all corners, and find and correct any SI or timing-related issues before the board is released for fabrication. Before starting simulation, it is important to verify that the design is properly routed and that it meets the specifications/constraints. In particular, it is essential to verify that the design does not include dangling and partially-routed/un-routed nets. We must also verify that all the nets meet the length constraints assigned to them. The Constraint Manager window helps identify nets that are in violation (shown in red) and nets that are in compliance (in green). For convenience and clarity, the Constraint Manager also reports the actual route length and the Manhattan lengths for each net.

The next step is to bring up the physical layout and visually inspect the nets to ensure that each net is routed in its appropriate layer, or run DRCs if the signals were explicitly limited to specific layers in Physical Constraint Sets. When test points are associated with a net, we must manually verify that the points are in line with the nets (and are not stubs hanging off the nets). Note that when using the simpler Total_Etch_Length constraint, the auto-router can meet routing length constraints for the net, even when there are stubs in the design. These stubs can produce undesirable effects such as reflections and hence this step is important. If there are too many

critical signals to check manually on larger designs, this check can be automated by using an explicit topology and stub length constraints. After manual inspection, we begin post–route simulation and generate reports to analyze the design. We then export the reports to an Excel spreadsheet to facilitate analysis.

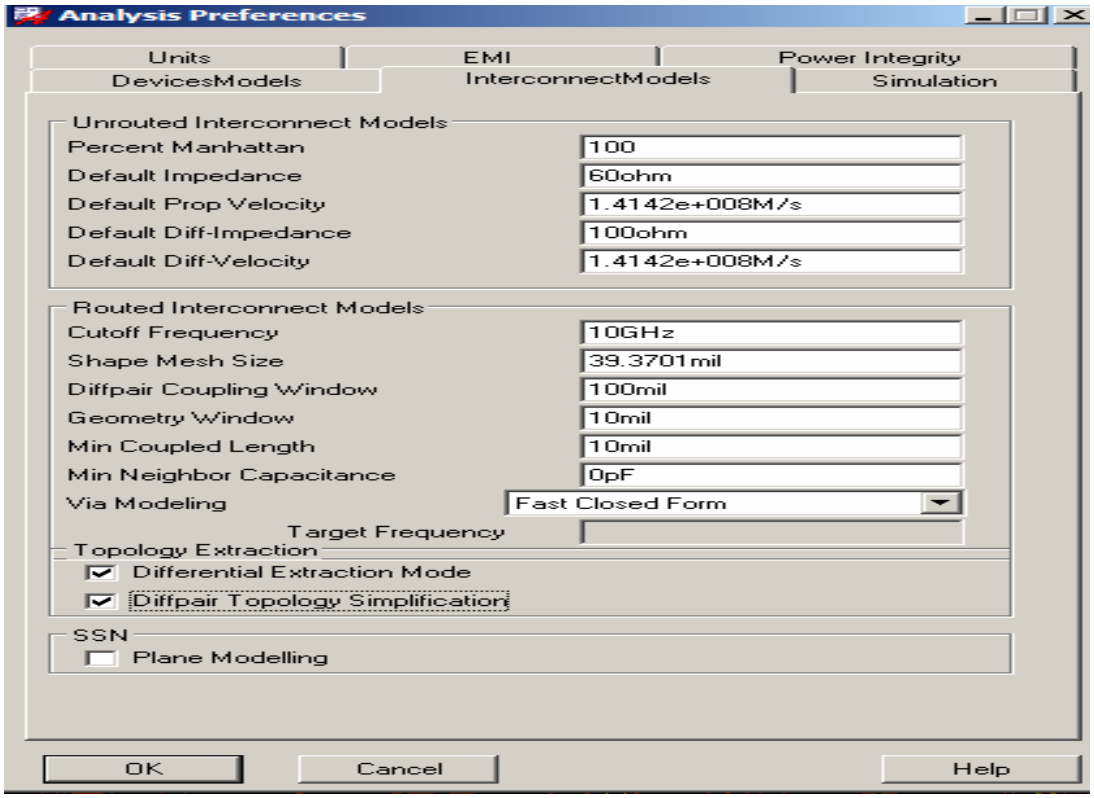
We generated both delay and reflection reports. The delay report provides information on timing parameters such as propagation delay, switch and settle rise and fall times. The reflection report presents data on signal integrity parameters such as overshoot, undershoot, noise margin, monotonicity, and glitch. Preparing the design for post-route simulation involves the selection of various options in the SI\EMI Sim preferences list. The following screen display describes this process.

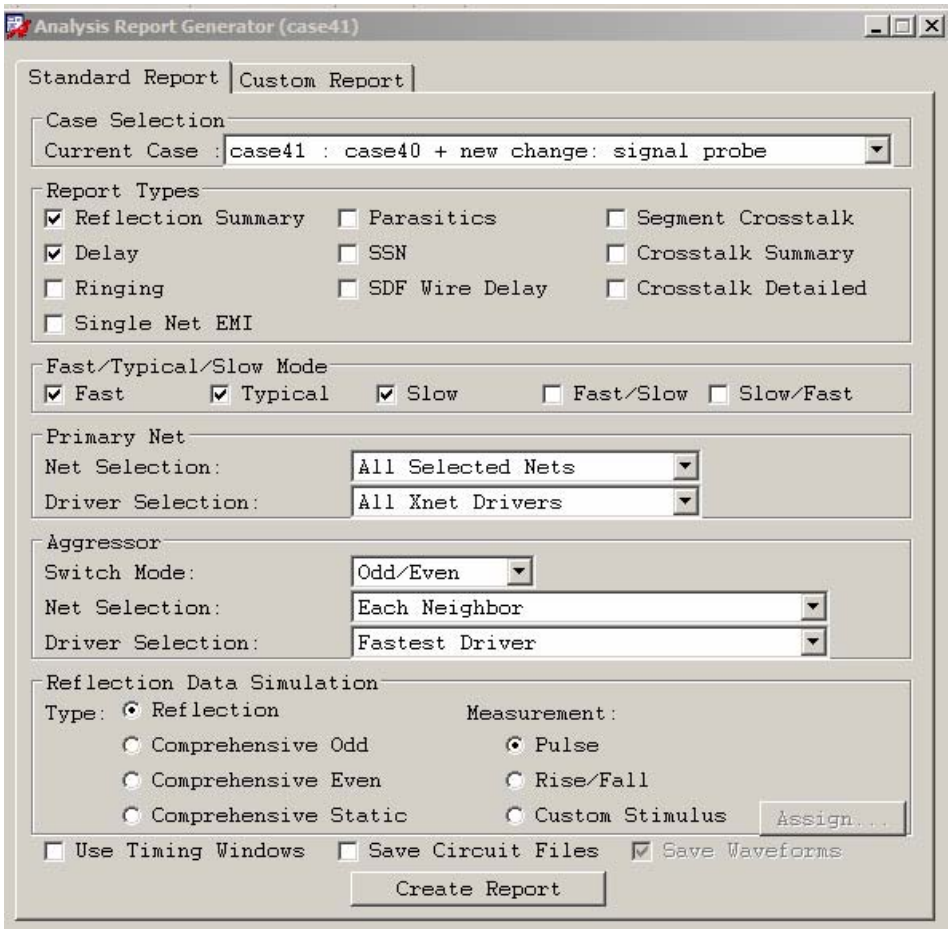
Analyze → SI/EMI Sim → preferences



In the form above, we set up the frequency of the stimulus and the duty cycle. We also set up V_{meas} as the reference for delay calculations. Choosing the reference as V_{meas}, rather than V_{IH} and V_{IL}, makes analysis much easier and is in accordance with the memory datasheet. We chose V_{meas} as 0.9V which is half of the peak-to-peak voltage swing (1.8V).

Now that the design is routed, we need to set the parameters for routed interconnects. Here you can specify the minimum coupling distance for nets for the tool to recognize it as a differential pair. This can be done by invoking Analyze → SI →Pref →Interconnect Models.


Analysis Preferences
Units
EMI
Power Integrity
DevicesModels
InterconnectModels
Simulation
Unrouted Interconnect Models
Percent Manhattan
100
Default Impedance
60ohm
Default Prop Velocity
1.4142e+008M/s
Default Diff-Impedance
100ohm
Default Diff-Velocity
1.4142e+008M/s
Routed Interconnect Models
Cutoff Frequency
10GHz
Shape Mesh Size
39.3701mil
Diffpair Coupling Window
100mil
Geometry Window
10mil
Min Coupled Length
10mil
Min Neighbor Capacitance
0pF
Via Modeling
Fast Closed Form
Target Frequency
Topology Extraction
Differential Extraction Mode
Diffpair Topology Simplification
SSN
Plane Modelling
OK
Cancel
Help


Analysis Report Generator (case41)
Standard Report
Custom Report
Case Selection
Current Case :
case41 : case40 + new change: signal probe
Report Types
Reflection Summary
Parasitics
Segment Crosstalk
Delay
SSN
Crosstalk Summary
Ringing
SDF Wire Delay
Crosstalk Detailed
Single Net EMI
Fast/Typical/Slow Mode
Fast
Typical
Slow
Fast/Slow
Slow/Fast
Primary Net
Net Selection:
All Selected Nets
Driver Selection:
All Xnet Drivers
Aggressor
Switch Mode:
Odd/Even
Net Selection:
Each Neighbor
Driver Selection:
Fastest Driver
Reflection Data Simulation
Type:
Reflection
Comprehensive Odd
Comprehensive Even
Comprehensive Static
Measurement:
Pulse
Rise/Fall
Custom Stimulus
Assign...
Use Timing Windows
Save Circuit Files
Save Waveforms
Create Report

The preceding screenshot shows the option that allows us to select the delay and reflection reports. In this form, we also choose all three simulation modes — fast, typical, and slow — to cover all corner cases. In our experience, running typical mode simulations were not enough to determine final timing margins over process, voltage, and temperature. So, we exported the reports to an Excel spread sheet and analyzed the results. Reflection and delay reports simulate only a primary net and none of its neighbors. As a result, these reports do not take into consideration the parasitics of the power and ground pins.

Timing > Control **typ**

Note: All timings in ns unless labelled otherwise.

Component Timing

	driving	to	**Memory**	
Tsetup	1.64		Tsetup	1.5
Thold	4.04		Thold	1.5

Skew_max = 1.64 - 1.5 = 140ps between clock and control
Skew_max= 0.14

Clock/Strobe Relationships
Sdram_Ctrl<6:7> is differential clock

Interconnect Timing

XNet	Drvr	Rcvr	PropDly	SettleRise	SettleFall	AvgSettle
SDRAM_CTRL<6>	U800	V2_L U2164 C7_U216	0.142029	1.13851	1.20538	1.172

XNet	Drvr	Rcvr	PropDly	SettleRise	SettleFall	MinSettle	MaxSettle	MinSettleSkew	MaxSettleSkew	MaxSkew	Margin
SDRAM_CTRL<0>	U800	U2164	0.1118	1.191	1.235	1.104	1.235	0.068	0.063	0.068	0.072
SDRAM_CTRL<10>	U800	U2164	0.1254	1.165	1.207						
SDRAM_CTRL<11>	U800	U2164	0.1114	1.141	1.187						
SDRAM_CTRL<12>	U800	U2164	0.1217	1.178	1.221						
SDRAM_CTRL<13>	U800	U2164	0.1067	1.114	1.153						
SDRAM_CTRL<14>	U800	U2164	0.09823	1.104	1.143						
SDRAM_CTRL<2>	U800	U2164	0.1274	1.163	1.205						
SDRAM_CTRL<3>	U800	U2164	0.09163	1.108	1.153						
SDRAM_CTRL<8>	U800	U2164	0.1081	1.137	1.182						
SDRAM_CTRL<4>	U800	U2164	0.06959	1.143	1.247						
SDRAM_CTRL<5>	U800	U2164	0.0862	1.169	1.285						

The preceding spreadsheet was created with data from delay reports and was used to analyze the control lines with respect to the clock. The clock signal in our design is called SDRAM_CTRL<6>. The sheet also lists the driver (U800, the processor), receiver (U2164, memory device), propagation delay (0.142029 ns), settle rise (1.13851 ns), and settle fall (1.20538 ns) values. The average settle delay (1.172 ns) is calculated by averaging the settle rise and settle fall numbers.

The control nets SDRAM<0> to SDRAM_CTRL <14> are listed next to the corresponding drivers, receivers, propagation delays, settle rise and settle fall delays. We then look for the minimum and maximum delays of all the settle rise and settle fall delays. These are listed under *maximum settle delay* (1.235 ns) and *minimum settle delay* (1.104 ns) respectively. Using these numbers, we calculate the *maximum settle skew* (0.063 ns), which is the difference between the maximum settle delay (1.235ns) and the average settle time (1.172 ns) of the clock signal. We also calculate the *minimum settle skew* (0.063 ns), which is the difference between the minimum settle delay (1.104ns) and the average settle time (1.172 ns) of the clock signal. Subtracting the maximum of these two skews, which in our case is 0.068 ns, from the total skew available (0.140 ns) gives the *margin* (0.072 ns) for these nets.

We repeated this analysis for both the fast and slow modes and calculated the available margins. In this design, we identified two control nets that met length constraints, but did not have positive margins in the slow mode. We worked with the PCB engineers to reroute these nets and made them shorter to meet the skew margins.

	A	B	C	D	E	F	G	H	I
46									
47	NAND SIGNALS - TYP MODE								
48									
49	XNet	Drvr	Rcvr	NMHigh	NMLow	OShootHigh	OShootLow	Monotonic	Glitch
50	NAND_CTL<0>	U800 L13	U2164 K3	346.9	349.7	1911	-147.2	PASS	PASS
51	NAND_CTL<1>	U800 B13	U2164 L3	345.5	357.3	1858	-91.98	PASS	PASS
52	NAND_CTL<2>	U800 G13	U2164 G3	330	331.8	1923	-186.2	PASS	PASS
53	NAND_CTL<3>	U800 B12	U2164 F3	319.5	313.4	1947	-170.2	PASS	PASS
54	NAND_CTL<4>	U800 H12	U2164 M3	304.2	288.3	1985	-236.1	PASS	PASS
55	NAND_CTL<5>	U800 C12	U2164 N3	297.7	278.5	2019	-242.4	PASS	PASS
56	WEMI_DATA<0>	U800 L12	U2164 P10	1201	497.3	2461	-254.3	PASS	PASS
57	WEMI_DATA<0>	U2164 P10	U800 L12	308.3	319.5	2055	-194.2	PASS	PASS
58	WEMI_DATA<1>	U800 C11	U2164 N10	1200	507.9	2460	-216.4	PASS	PASS
59	WEMI_DATA<1>	U2164 N10	U800 C11	322.1	347.5	2005	-139.5	PASS	PASS
60	WEMI_DATA<10>	U800 H10	U2164 L11	1200	535.4	2460	-185.7	PASS	PASS
61	WEMI_DATA<10>	U2164 L11	U800 H10	360	373.1	1939	-76.79	PASS	PASS
62	WEMI_DATA<11>	U800 B9	U2164 K11	1201	541.6	2461	-123.4	PASS	PASS
63	WEMI_DATA<11>	U2164 K11	U800 B9	342.6	369.4	1916	-71.52	PASS	PASS
64	WEMI_DATA<12>	U800 C9	U2164 G11	1201	554.9	2461	-80.01	PASS	PASS
65	WEMI_DATA<12>	U2164 G11	U800 C9	356.5	379.6	1842	-18.11	PASS	PASS
66	WEMI_DATA<13>	U800 B8	U2164 F11	1201	531.6	2461	-138.4	PASS	PASS
67	WEMI_DATA<13>	U2164 F11	U800 B8	337.3	362.5	1928	-89.06	PASS	PASS
68	WEMI_DATA<14>	U800 C8	U2164 E11	1200	554.2	2461	-97.18	PASS	PASS
69	WEMI_DATA<14>	U2164 E11	U800 C8	370.9	388.3	1835	-21.24	PASS	PASS
70	WEMI_DATA<15>	U800 B7	U2164 D11	1201	552.3	2461	-117.6	PASS	PASS
71	WEMI_DATA<15>	U2164 D11	U800 B7	364.5	383.5	1870	-49.3	PASS	PASS
72	WEMI_DATA<2>	U800 F11	U2164 M10	1200	521.4	2460	-219.6	PASS	PASS
73	WEMI_DATA<2>	U2164 M10	U800 F11	354.5	363.7	1959	-101.4	PASS	PASS
74	WEMI_DATA<3>	U800 B11	U2164 L10	1200	500.5	2460	-286.9	PASS	PASS
75	WEMI_DATA<3>	U2164 L10	U800 B11	330.1	345.6	2007	-130.9	PASS	PASS
76	WEMI_DATA<4>	U800 C10	U2164 F10	1200	534.5	2460	-149.9	PASS	PASS
77	WEMI_DATA<4>	U2164 F10	U800 C10	326.7	362.4	1936	-61.69	PASS	PASS
78	WEMI_DATA<5>	U800 A10	U2164 E10	1200	530.2	2461	-166	PASS	PASS
79	WEMI_DATA<5>	U2164 E10	U800 A10	337.9	362.5	1933	-98.84	PASS	PASS
80	WEMI_DATA<6>	U800 G11	U2164 D10	1201	531.5	2461	-174.3	PASS	PASS
81	WEMI_DATA<6>	U2164 D10	U800 G11	342.2	368.4	1942	-95.78	PASS	PASS
82	WEMI_DATA<7>	U800 B10	U2164 C10	1201	533.1	2461	-186.1	PASS	PASS
83	WEMI_DATA<7>	U2164 C10	U800 B10	349.9	371.3	1918	-94.22	PASS	PASS
84	WEMI_DATA<8>	U800 F10	U2164 N11	1200	528	2460	-175.4	PASS	PASS
85	WEMI_DATA<8>	U2164 N11	U800 F10	338.1	359.7	1983	-83.1	PASS	PASS
86	WEMI_DATA<9>	U2164 M11	U800 A9	1200	517.8	2460	-227.4	PASS	PASS
87	WEMI_DATA<9>	U800 A9	U2164 M11	345	360.4	1981	-109.8	PASS	PASS

The preceding spreadsheet was generated from the reflection report. Noise margins were calculated from the overshoot and undershoot values. The spread sheet is formatted to indicate all the passing signals in green (as long as overshoot and undershoot is less than 0.7V the signal passes). The report also checks for monotonicity and glitches. This check was completed on all the other net groups in the design.

In addition to looking at the single-line (i.e. uncoupled) reflections and resulting interconnect delays for the memory interface signals, it is also to be able to quantify the crosstalk these signals are subjected to, and the impact of that coupling on signal integrity and timing. First, the impacts on timing were examined. A topology was built in SigXplorer for 3 coupled data signals, with conductor spacing set to the minimum value, and lengths representing 150% Manhattan distance between the controller and memory chips. Stimuli in this

topology were varied to quantify the “push out” and “pull in” seen on the victim data signal due to the activity on the coupled neighbors. Even mode stimulus (neighbors switching in the same direction as the victim) tends to exhibit push out, while odd mode stimulus tends to do the opposite. This general effect is illustrated below.

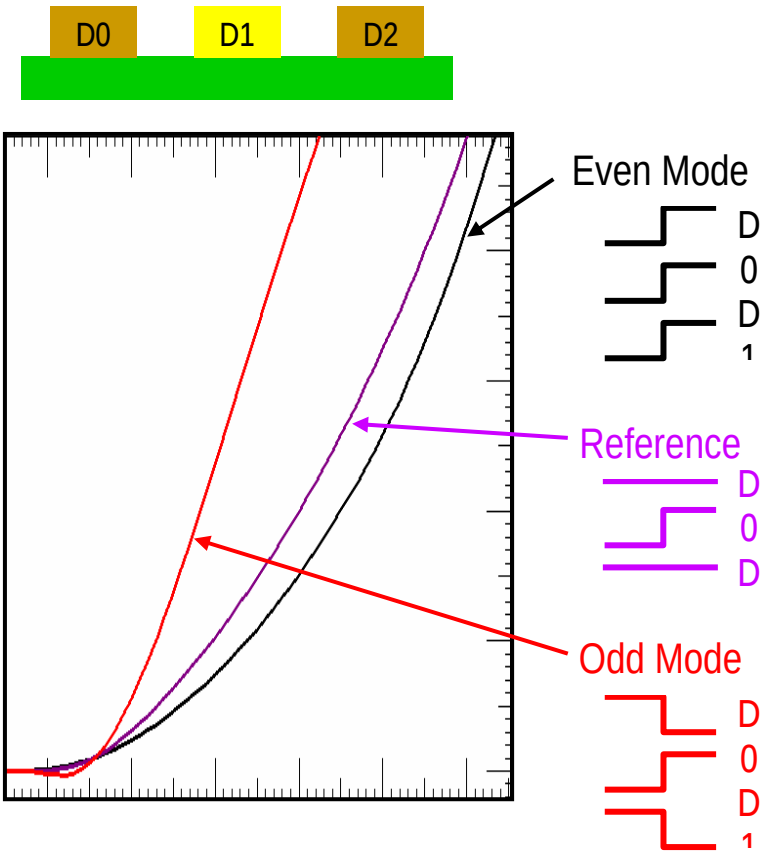
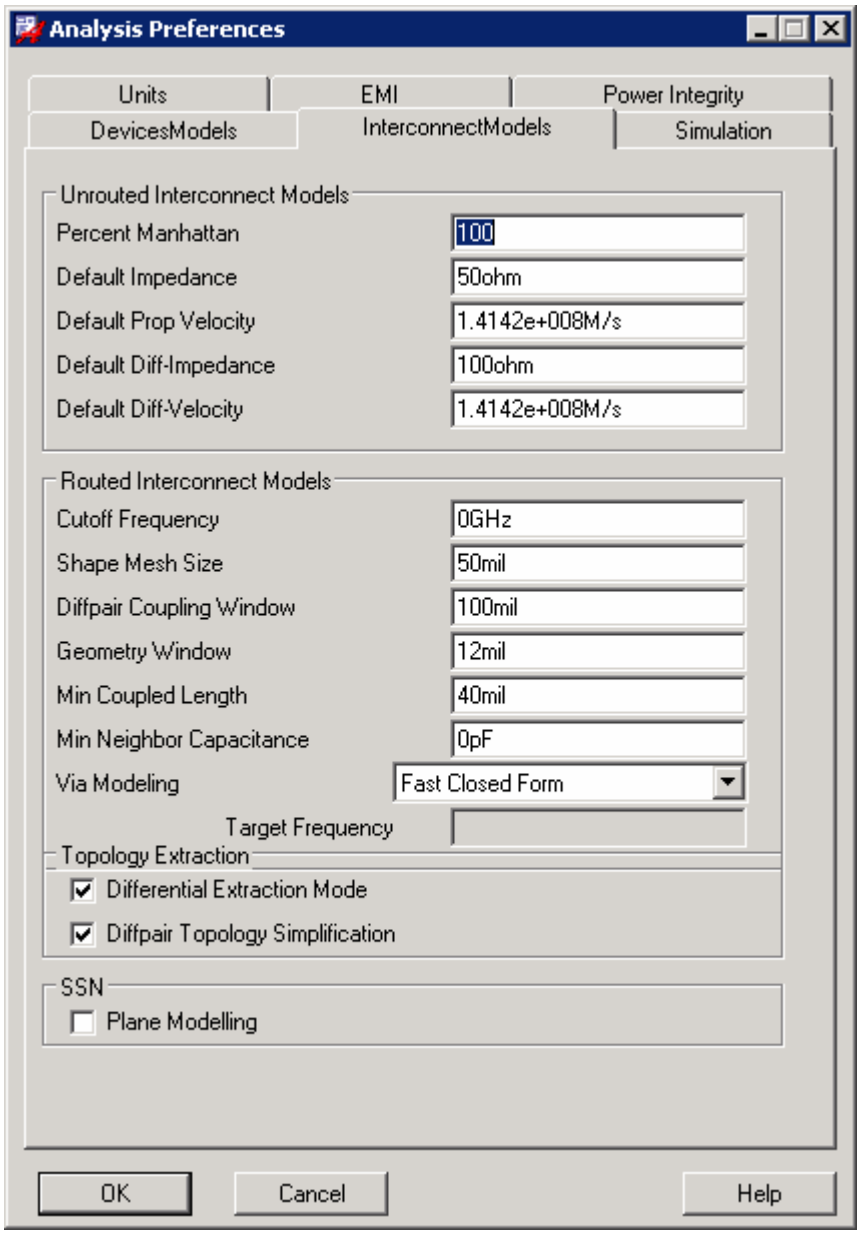


Figure 11. Even and odd mode crosstalk

Through these exploratory simulations, it was determined that on the small PCB being designed, neighbor coupling had a very small effect on the interconnect delays. While this is not generally true in all cases, it was found that the relatively short interconnect paths on this particular design caused minimal push out and pull in effect. This being determined, it was decided that intra-bus coupling could safely be disregarded on this design, which helped to avoid over-constraining the layout. Bus routing could be done using the minimum line-to-line spacing for manufacturability.

The next coupling effect that was addressed was that of inter-bus crosstalk. Here the main concern was whether the voltage level of a steady-state signal could be pushed out of its desired voltage level during sampling, due to coupling from aggressors outside the bus. Assuming the design is setup properly, the first step for crosstalk analysis is to set up the Analysis Preferences appropriately. The Interconnect Models tab of this form is shown below.

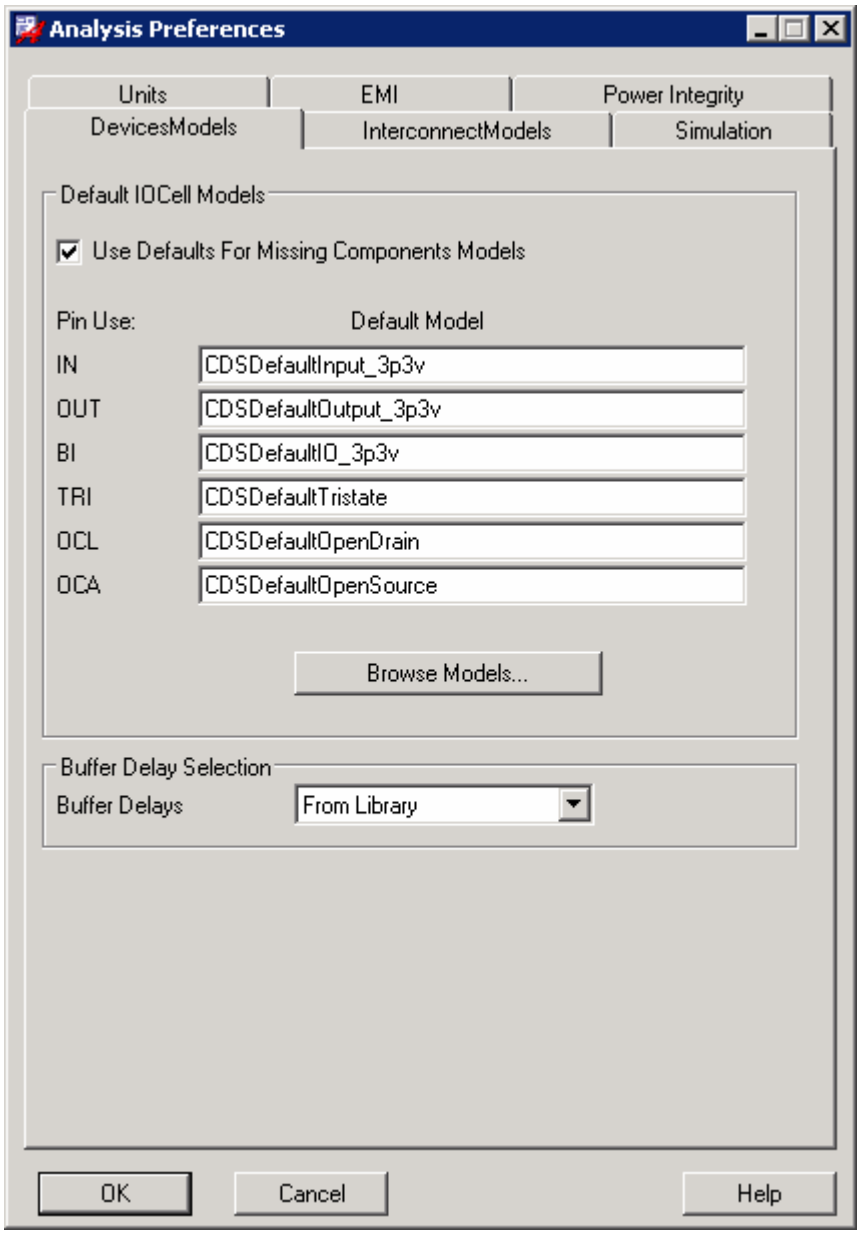


There is some judgment to apply here. The goal is to screen out any significant xtalk issues in the design, without spending excessive computational time. The key items to consider here from a crosstalk standpoint are:

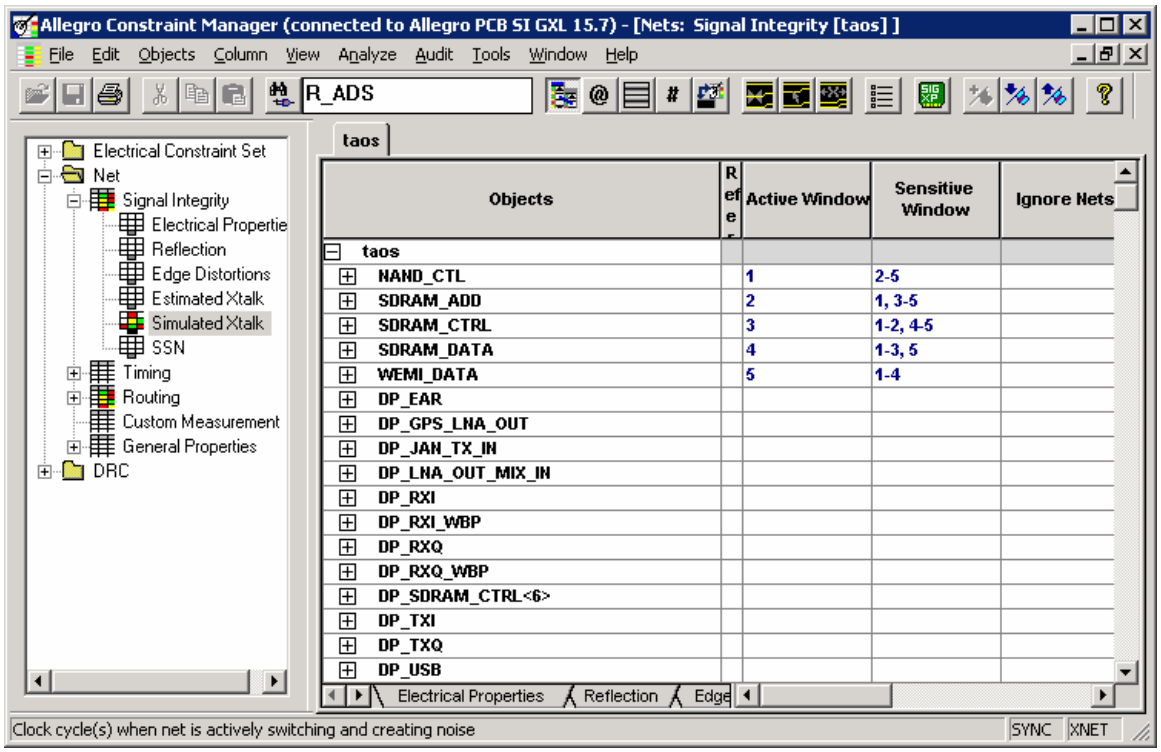
- Geometry Window - how far to look for aggressors
- Min Coupled Length – minimum run you model as coupled; segments shorter than this are modeled as uncoupled

For the initial run, the Geometry Window was set to pull in the nearest neighbor on each side of the victim net, and to ignore coupled segments smaller than 40 mils in length. Once a baseline is established, the user can expand these quantities to increase the crosstalk coverage in the design.

It is also important to set up default models for any driver/receiver pins that are not explicitly modeled. As this design used primarily 3.3V logic, 3.3V default driver models were specified for any pins not explicitly modeled, as shown below in the “Device Models” tab.



The next step is to set up Crosstalk Timing Windows to ignore intra-bus xtalk, focusing the analysis on inter-bus xtalk effects. An example of how to set this up in Constraint Manager as shown below for the 5 buses found in this design.



Crosstalk timing windows enable intelligent summing to be applied to crosstalk DRCs, and intelligent stimuli to be applied to neighboring XNets for crosstalk simulations. With no crosstalk timing windows defined, Allegro SI will assume that the victim is always sensitive to crosstalk and aggressors are always active, i.e. crosstalk is always generated by all present neighbor XNets. This can be overly pessimistic in many cases, and can cause very conservative layout, sacrificing density. The following crosstalk timing windows parameters can be defined to control which neighbors are to be considered as aggressors for crosstalk analysis:

- Active Window – time slots in which aggressor XNets can switch, and cause crosstalk (ex. 1)
- Sensitive Window – time slots in which victim XNets are sensitive to crosstalk from aggressors (ex. 1-2, 4-5)
- Ignore Nets – nets for a victim to ignore as sources of crosstalk (can specify nets or Electrical Constraint Sets)

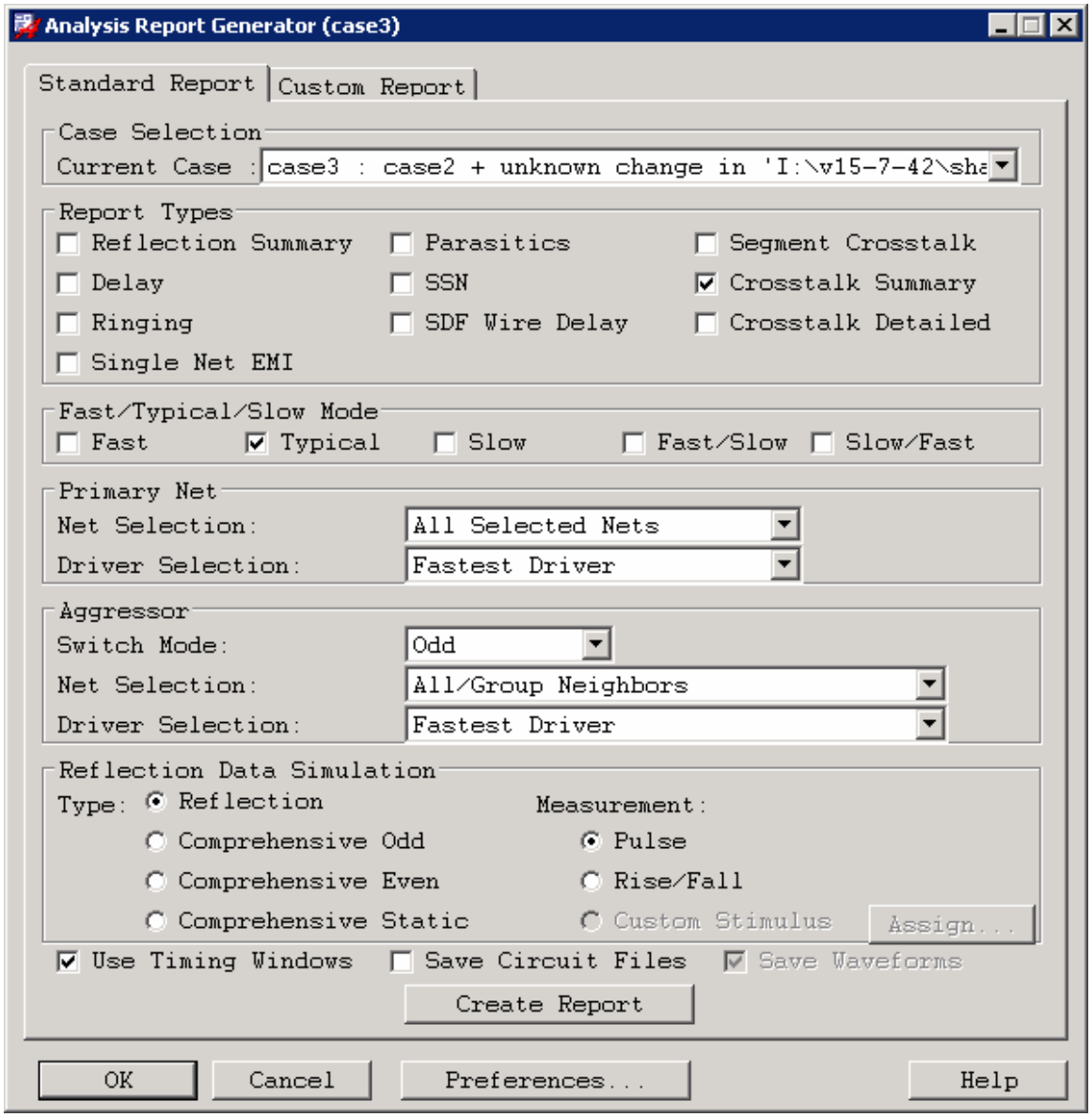
As an example, consider the following case:

- victim XNet “SDRAM_ADD1” has the attributes:
  - Active Window = 2, Sensitive Window = 1, 3-5
- aggressor XNet “SDRAM_ADD2” has the attributes:
  - Active Window = 2, Sensitive Window = 1, 3-5
- aggressor XNet “WEMI_DATA5” has the attributes:
  - Active Window = 5, Sensitive Window = 1-4

When crosstalk analysis is run for the victim SDRAM_ADD1, its ***sensitive*** window (1, 3-5) will be compared with the ***active*** window for aggressors SDRAM_ADD2 (2) and WEMI_DATA5 (5). Since SDRAM_ADD2’s active window does NOT overlap with the victim’s sensitive window, its crosstalk contributions will not be considered during analysis. But since WEMI_DATA5’s active window DOES overlap with the victim’s sensitive window (5 overlaps with 1, 3-5), its crosstalk contributions WILL be considered during analysis.

So Crosstalk Timing Windows give a simple means for controlling whether or not a signal (or group of signals) is a valid source of crosstalk for another signal (or group of signals). Using this control wisely can significantly decrease the amount of analysis required, eliminate false crosstalk DRCs, and improve routing density on the PCB.

Once Crosstalk Timing Windows were defined in the Constraint Manager, simulations were run using the "Probe" functionality. An example of the setup for the crosstalk analysis run is shown below.



A couple of key points should be made for this initial "screening pass":

- Specify "Fastest Driver" for both Primary and Aggressor initially. This will typically produce worst case results, and will minimize the number of simulations that will be run.
- "All/Group Neighbors" will stimulate all valid aggressor signals simultaneously, again for worst case.
- Turn on "Use Timing Windows", to skip intra-bus xtalk cases (which also saves computational time).

Once the initial crosstalk report has been generated, it can be desirable to determine, for the worst signals, which are the main culprits are so that they can be addressed at layout. To do this, these "outliers" can be re-analyzed with slightly different settings. The key item here is to set "Net Selection" to "Each Neighbor" in the "Aggressor" section of the form. As opposed to the previous analysis, where all aggressors to a net were stimulated simultaneously, in this run each aggressor neighbor will be stimulated individually, so that you get a breakdown of the contributions from each individual aggressor, which can be used to guide layout edits.

Results

The following results were noted from the SI work performed on this card:

1. All the signals in the DDR memory interface met timing margins in typical, slow and fast cases in simulation
2. All the nets met overshoot and undershoot specifications under fast, typical and slow cases in simulation
3. Once hardware came into the lab, the interface passed functional, signal integrity and timing specifications under hot (85^{o}C), cold (-30^{o}C) and room temperature in the actual design.
4. The interface was robust across different software loads and mechanical life tests.

In summary, as a result of the SI simulation work done on this design, we were able to achieve first pass results with our hardware. No re-spins or re-work were required due to timing or SI issues. Eliminating unnecessary re-spins saves significant time and money in terms of materials, fabrication, assembly, test, verification, and manpower. It also enables us to get to market faster, with higher quality product for our customers.

Recommendations

For new users of the Allegro SI tool, the authors have a number of recommendations that can help to make their initial experiences run smoothly.

- Central SI library – Establish a central SI library that the entire engineering community in your organization can use. Having multiple pockets of redundant models not only wastes time and resources, it produces inconsistent results. This is a must if SI analysis is to be an efficient part of your flow.
- Invest in setup – Taking the time to set up the layout database properly up front pays back many times during the design and analysis process. Get the stack-up completely defined in the master layout database, so any versions of the database that get analyzed later all benefit. The same goes for voltage properties, pinuse, and SI model assignments. If you can get this all set up from the beginning, it saves time throughout the rest of the process and eliminates mistakes.
- Signal naming conventions – When working with groups of signals in a design, such as those found in a memory interface, it is much easier to digest reports, set properties and constraints, and review feedback in the Constraint Manager when the signal names are intuitive. Take care in naming all the relevant signals in your design to make them easy to work with. Using the same naming conventions over again on subsequent designs will speed up the analysis process.
- Partner with the layout designer – For critical signals with tight timing margins, it is beneficial to get them routed as a first priority. Work with your layout designer to make sure the optimum route priority is identified. And there will almost always be some give-and-take from a constraint standpoint. The layout designer is often in the best position to determine if the routing constraints you are requesting are realistic or not, especially on dense designs. Work with the layout designer to make the design

physically realizable, and agree to re-simulate signals that can't be routed exactly as specified. Let the timing margins be the final judge as to whether or not a signal is acceptably routed.

- Fast and slow corners – When timing margins are tight, it is critical to simulate the Fast and Slow corners as well as Typical. Often Fast or Slow results turn out to produce the worst case margins, and the design needs to be robust enough to meet requirements under these tough conditions.
- When comparing results for the same signals in the SigXplorer (topology) and SpecctraQuest (physical layout) environments, be aware that some simplifying assumptions are made in the case of SigXplorer from a usability standpoint. There will always be more detailed input directly from the physical layout, so when minor differences appear, let the results from simulations in the layout take precedence.

Future Work

Some of the future work expected to be done on the high speed flow at Motorola is expected to include the following:

- Automated bus-level SI analysis (introduced in 15.7) – This capability automates a number of things for source synchronous bus analysis, including custom bit patterns, automated handling of on-die termination, and automated setup and hold margin measurements. These items and others are expected to be requirements for DDR2 designs with higher data rates.
- Crosstalk-driven routing – It is possible to enable crosstalk DRCs during automatic and interactive routing. This would enable the layout designer to get immediate feedback on coupling during the routing process, and facilitate crosstalk avoidance.
- IR drop analysis (introduced in 16.0) – This is a critical item for many portable and/or battery-operated designs. Without this automation, calculating IR drop in the design is a very tedious and error-prone process.
- Decoupling cap analysis (Power Integrity option) – With density, real estate, cost, and shrinking voltage ripple specifications, this becomes an important part of the design process.

About the Authors

Prithi Ramakishnan is a Senior electrical engineer with Motorola's iDEN subscriber group specializing in digital design. She has her bachelor's degree in Electrical Engineering from V.J.T.I, Mumbai, India and her Masters degree in Electrical Engineering from Pennsylvania State University, State College, PA.

Ken Willis is a Sr. Staff Services Application Engineer for Cadence Design Systems, providing high-speed design and consulting services in the Silicon-Package-Board Services group. Ken was the original Signal Integrity (SI) Applications Engineer with Cadence, where he also served as Technical Marketing Director and Engineering Director for SI product development for over 8 years. Ken then worked for Sirocco Systems/Sycamore Networks, where he was a lead SI Engineer and used Cadence SI tools on many high speed PCB designs. Prior to joining Cadence, Ken worked as a Process Engineer in PCB fabrication with the Tyco Printed Circuit Group, and as an SI Engineer with Compaq Computer Corporation. Ken received a BSEE from Worcester Polytechnic Institute in 1988.