Managing Cross-talk Noise

Rajendran Panda
Motorola Inc., Austin, TX

Advanced Tools Organization

- Central in-house CAD tool development and support organization catering to the needs of all design teams across the semiconductor sector.
- Located in Austin, satellite activities in India, Russia, Australia, Israel, and Germany
- Four focus areas:
  - Circuit simulation (Spice) - Brian Mulvaney
  - Physical design - Patrick McGuinness
  - Functional verification - Mike Garcia
  - Analysis and Optimization – Rajendran Panda
    - Power grid and signal integrity
    - Circuit and interconnect reliability
    - Inductance analysis
    - Clock analysis
    - Leakage optimization
    - Fast circuit simulation
    - Process variation issues
Acknowledgements

- To the Signal Integrity team:
  - Chanhee Oh
  - Murat Becer
  - Amir Grinshpon
  - Rafi Levy
  - Ilan Algor
- And to the numerous designers who helped to drive this activity

Talk Outline

- Functional and Delay Noise
- Correlation between them
- SI Methodology and Experiences
  - Preventive Measures
  - Functional Noise Repair
  - Delay Noise Analysis/Repair challenges
  - Delay Noise Repair
- Summary
Introduction

- Crosstalk noise is an undesired change in the voltage waveform of a net due to signal activity in its neighboring nets which are capacitively coupled to it.
- Ratio of crosstalk capacitance to total capacitance is increasing.
- Faster slews result in increased injected noise
- More aggressive and less noise immune circuit structures are being used due to performance requirements.

Functional Noise

- Crosstalk causes voltage glitches on quiet nets, resulting in false logic states being captured in the registers, causing functional failures.
Noise on Delay

- Noise on delay changes the signal propagation on some of the nets, causing timing violations.

Noise Analysis

- Thevenin model for aggressor driver
- Holding resistance for victim driver
- Noise propagation table

Failure criteria:
* Max. noise at receiver input
* Max. noise at receiver output
Functional and Delay Noise Correlation

• Large functional noise also results in large delay noise

Functional and Delay Noise Correlation

• Fixing a functional violation often fixes many delay violations
Buffer Insertion

- Repair actions are effective for both functional and delay noise

Methodology Dilemma

- Design is SI clean only when all functional and delay violations due to noise are eliminated.
- Small glitches cause no problem functionally, but even small $\Delta$ delays of nets can add up to large path delays, suggesting delay problem is harder to deal with and so should be tackled early.
- On the other hand, delay noise analysis is inherently more expensive. Design groups tend to live with ‘guard-banding’ timing for noise and tackle only functional noise issues explicitly.
- There is a strong correlation in the occurrence and magnitudes of both these violations.
- Which violations should be tackled first? Delay violations or functional violations?
Why fix functional problems first?

- Since nets are shared by multiple timing paths, every path through a net failing functional noise criteria is likely to fail, even the non-critical ones.
  - Delay failure list is too large to manage efficiently, before functional noise fixes.
- Small glitches (which produce small delta delays) may not matter for many non-critical paths. So, after the large magnitude functional violators are eliminated, the number of violated paths decreases drastically.
- Easier to drive repair actions (buffering and sizing) using the noise magnitude metric (of functional noise), rather than the delta delay metric (of delay noise).
Case Study Details

- A wireless communication chip (SoC_Chip)
  - Integration with ~90K top level nets
  - 20 large SoC blocks/platforms
  - SoC blocks are delivered timing and SI clean
- A low-power IP platform (SoC_Platform)
  - 1 synthesized IP core, 11 synthesized modules, and 24 compiled memories
  - ~150K placed instances and ~160K nets
- Two functional blocks
  - ~45K nets SoC_Block_1
  - ~165K nets SoC_Block_2
- A high performance core (SoC_Core)
  - ~227K nets

All designs in 0.13um technology

Early Noise Prevention

- Preventive measures
  - Limit on parallel run length
  - Shielding of buses
  - Routing with extra spacing
  - Limit on slews
- Preventive actions do not require expensive noise analysis
### Prevention: Parallel Run Limits

**SoC Platform example:**

<table>
<thead>
<tr>
<th></th>
<th>Limit on parallel run length</th>
<th># delay violations</th>
<th>Worst delay slack</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>150um</td>
<td>300um</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-0.22ns</td>
<td>-0.57ns</td>
<td></td>
</tr>
</tbody>
</table>

**SoC Block-2 example:**

<table>
<thead>
<tr>
<th></th>
<th>Limit on parallel run length</th>
<th># func violations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>500um</td>
<td>497</td>
</tr>
<tr>
<td></td>
<td>No limit</td>
<td>1013</td>
</tr>
</tbody>
</table>

- Few trial routes required to obtain a reasonable number

---

### Prevention: Slew Constraints

**SoC Platform example:**

<table>
<thead>
<tr>
<th></th>
<th>Slowest transition time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.4ns</td>
</tr>
<tr>
<td># delay violations</td>
<td>2818</td>
</tr>
<tr>
<td>Worst delay slack</td>
<td>-0.22ns</td>
</tr>
<tr>
<td># functional violations</td>
<td>91</td>
</tr>
<tr>
<td># inserted buffers</td>
<td>3559</td>
</tr>
</tbody>
</table>

- Stronger victims overshadow the effect of sharper aggressor slews
- Increase in power consumption due to additional buffers is sub-linear due to reduction in short-circuit power

---
Prevention: Wide Spacing

SoC Platform example:

<table>
<thead>
<tr>
<th></th>
<th>Spacing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1x</td>
<td>2x</td>
</tr>
<tr>
<td>Delay noise on</td>
<td>0.22ns</td>
</tr>
<tr>
<td>memory bus</td>
<td></td>
</tr>
<tr>
<td>Timing slack on</td>
<td>-0.68ns</td>
</tr>
<tr>
<td>path</td>
<td></td>
</tr>
</tbody>
</table>

Functional Noise Analysis

- Beware of false violations and false fixes
  - Lot of effort can get consumed with no real benefit
- Timing windows
  - Activity windows (for aggressors)
  - Sensitivity windows (for victims)
- Logic constraints
  - Invert, same, imply, set_high/low/stable
  - One-hot, one-cold, one-switching

<table>
<thead>
<tr>
<th>For SoC Core Example</th>
<th># Violations</th>
</tr>
</thead>
<tbody>
<tr>
<td>No constraint</td>
<td>595</td>
</tr>
<tr>
<td>Logic constraints only</td>
<td>555</td>
</tr>
<tr>
<td>Activity windows only</td>
<td>187</td>
</tr>
<tr>
<td>Activity+Sensitivity windows</td>
<td>79</td>
</tr>
<tr>
<td>Logic + Timing constraints</td>
<td>50</td>
</tr>
</tbody>
</table>
Functional Noise Repair

Repair preferences:

- Sizing, buffering, and routing fixes suitable for block level noise repair
- Sizing not desirable at SoC integration stage
- Routing and buffering fixes preferred over sizing at SoC integration stage. (Assumes all SoC blocks are timing and SI clean.)

SI Convergence: Double Spacing

SoC Chip example:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>#violations remaining</th>
<th>#nets spaced</th>
<th>#violations fixed</th>
<th>Effectiveness (#fixed/#spaced)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>885</td>
<td>240</td>
<td>356</td>
<td>1.48</td>
</tr>
<tr>
<td>2</td>
<td>529</td>
<td>176</td>
<td>198</td>
<td>1.13</td>
</tr>
<tr>
<td>3</td>
<td>331</td>
<td>254</td>
<td>220</td>
<td>0.87</td>
</tr>
<tr>
<td>4</td>
<td>111</td>
<td>308</td>
<td>64</td>
<td>0.21</td>
</tr>
<tr>
<td>Total</td>
<td>47</td>
<td>1078</td>
<td>838</td>
<td>0.78</td>
</tr>
</tbody>
</table>

- Law of diminishing returns applies to wide spacing
### SI Convergence: Sizing and Buffering

#### SoC Block-1 example:

<table>
<thead>
<tr>
<th>Iteration</th>
<th># violations</th>
<th># gates sized</th>
<th># buffers inserted</th>
<th># violations fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1380</td>
<td>1170</td>
<td>100</td>
<td>1180</td>
</tr>
<tr>
<td>2</td>
<td>500</td>
<td>400</td>
<td>80</td>
<td>450</td>
</tr>
<tr>
<td>3</td>
<td>280</td>
<td>200</td>
<td>50</td>
<td>250</td>
</tr>
<tr>
<td>4</td>
<td>170</td>
<td>120</td>
<td>45</td>
<td>160</td>
</tr>
<tr>
<td>5</td>
<td>90</td>
<td>75</td>
<td>40</td>
<td>85</td>
</tr>
<tr>
<td>6</td>
<td>70</td>
<td>50</td>
<td>15</td>
<td>65</td>
</tr>
</tbody>
</table>

- Spacing not attempted because design was congested and only 4 metals were available.
Size First or Buffer First?

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Sizing before buffering</th>
<th>Buffering before sizing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Violations</td>
<td># Sized</td>
</tr>
<tr>
<td>1</td>
<td>208</td>
<td>197</td>
</tr>
<tr>
<td>2</td>
<td>51</td>
<td>48</td>
</tr>
<tr>
<td>3</td>
<td>23</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td>272</td>
<td>9</td>
</tr>
</tbody>
</table>

- Sizing is less intrusive, causes less disruption to routing, and is easily implemented with incremental routing.

Timing with Noise

SoC Platform example:

<table>
<thead>
<tr>
<th></th>
<th># Setup Violations</th>
<th>Timing Slack</th>
</tr>
</thead>
<tbody>
<tr>
<td>STA with 2.0X Coupling Multiplier</td>
<td>4865</td>
<td>-1300 ps</td>
</tr>
<tr>
<td>STA with Delay Noise</td>
<td>2936</td>
<td>-620 ps</td>
</tr>
<tr>
<td>STA with 1.5X Coupling Multiplier</td>
<td>1138</td>
<td>-560 ps</td>
</tr>
</tbody>
</table>

- STA with delay noise is slow due to the added noise analysis overhead and iterations for timing windows convergence.
- Assume infinite windows for first iteration and iterate over critical paths only, for faster analysis.
Excessive Pessimism in Delay Uncertainty Analysis

Assumptions
- All aggressors of all nets in victim path switch in the same direction.
- Switching of all aggressors of a net align well to cause maximum delay variation.
- Launch and capture clocks slow down/speed up independently of each other, and of the victim.

Result
- Excessive pessimism in setup and hold time analyses
- Noise induced by victim on clock may be double counted.

Difficulties in Reducing Pessimism

Need to consider:
- whether delay variations of clocks are independent or not, based on type of analysis (setup or hold), phases of clocks, etc.
- whether launch and capture clocks have shared paths
- logical constraints between aggressors, victims and clocks involved in the entire path – very expensive to find worst set of aggressors creating worst setup/hold violation for a given path.
Small delay changes – hard to fix

- Delay change on each net is only small (few pS), but the cumulative effect on entire path is significant.
- Need to fix some, but which ones?
- Repair methods are less effective if fixing too many small violations.
- Get caught in convergence issues – fix some, but create new violations.

Delay Noise Repair

1. STA with coupling and infinite windows
2. STA iterations with coupling and updated windows
3. For each failing path
   1. Select a net contributing most delta delay
      1. Resize driver to next higher size
      2. Incremental STA
      3. If new timing violations on other paths
         1. Revert back to original size
         2. Create router constraint
      4. Else if timing of path improves and no new timing violations
         1. Accept and legalize placement
         2. If path meets timing now, take up next path (Step 3)
         3. If not, take up next net in the path (Step 3.1)
Delay Noise Repair

- SoC Platform example:
  - Num. STA iterations with windows: 2
  - Initial violations: 88 (setup), 63 (hold)
  - Initial slack: -100ps (setup), -20ps (hold)
  - Num. Drivers sized: 76
  - Num. Nets spaced: 12

Summary

- Occurrence and magnitude of functional violations and delay violations in a design are highly correlated.
- It is efficient to tackle functional violations first, and then deal with delay violations.
- Preventive actions are very valuable. Moreover, they do not require detailed noise analysis.
- Sizing fixes are less intrusive than buffering, and give quicker SI convergence.