Technology Lookup Table based Default Timing Assertions for Hierarchical Timing Closure

Ravi Ledalla, Debjit Sinha, Adil Bhanji, Chaobo Li, Gregory Schaeffer,
Hemlata Gupta, Jennifer Basile
IBM Systems (EDA), Poughkeepsie, USA
{ravc, adil, lich, gmschaef, guptah, basile}@us.ibm.com, debjitsinha@yahoo.com

***Abstract*—This paper presents an approach to dynamically generating representative external driving cell and external wire parasitic assertions for the ports of sub-blocks of a hierarchical design. The assertions are based on a technology lookup table and use attributes of the port and the hierarchical wire connected to the port as keys. A concept of reverse timing calculation at the input of the driving cell is described that facilitates the approach to drive efficient timing optimization of boundary paths of design sub-blocks. Experimental results in an industrial timing environment demonstrate significantly improved timing optimization accuracy when compared to prior work.**

Keywords—Timing analysis, assertions, hierarchical timing

I. INTRODUCTION

Microprocessor designs continually push the limits of design, technology, function and timing. Modern microprocessor designs now contain in excess of 9 billion transistors along with multiple processing cores operating at frequencies over 5 GHz. Challenges abound on all fronts including growing logic densities, shrinking physical geometries, increased operating frequencies, power usage constraints, reliability and the need to have on-time first-pass working silicon, necessitate a parallel approach to chip design and timing closure. The design complexity at hand has driven a paradigm shift from a flattened full-chip design methodology to one which is hierarchical [1]. By dividing the design into hierarchy, and then partitioning within the hierarchy, design work can proceed concurrently and be put together at the chip level for final analysis and optimization.

A hierarchical chip design typically includes partitions and multiple levels of hierarchy. The top *chip* level can contain a number of *cores*; where each core contains logic that is organized into *units* which in turn contain as many as a few hundred sub-blocks or *macros*. We refer to a partition inside the design (e.g., macro, unit, core) as a sub-block for rest of this paper for generality.

To facilitate parallel design, sub-blocks are designed and timing-optimized in isolation. In this environment, the sub-block is termed to be in its *out-of-context* or OOC state. Following optimization, the sub-block or its macro-model [2]-[3] is plugged into a parent level of hierarchy. The parent level of hierarchy for the instantiated sub-block is termed the *in-context* or IC state for that sub-block. It is critical to have consistent OOC- and IC-timing for any logic inside the sub-block to avoid incorrectly guided timing optimization at the OOC stage.

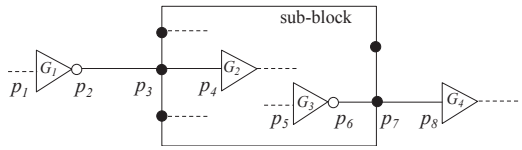

Fig. 1. Illustration of a driving gate feeding an input port of a sub-block.

Designers aim to minimize iterations of OOC and IC timing for a given sub-block by ensuring that the OOC timing accurately models the IC timing environment at the boundary ports (input and output pins). Traditionally, the boundary timing environment is obtained from an IC timing run and captured as timing *assertions* that are applied during OOC timing. These assertions describe the timing characteristic at each port of the sub-block and represent the timing values that are present at the IC environment.

Fig. 1 illustrates a small part of an IC timing environment with an instantiated sub-block having multiple ports that include an input port p_3 and an output port p_7. The port p_3 is part of a hierarchical wire that includes a segment outside the sub-block (between ports p_2 and p_3) and a segment inside the sub-block (between ports p_3 and p_4). Port p_3 is driven by a gate G_1, with input and output ports, p_1 and p_2, respectively. At each of the ports, there are timing values that represent the propagated signal arrival times (ATs), the signal (voltage) waveform shapes with corresponding transition times or slews (SLWs), and the required signal arrival times (RATs).

In an OOC run, timing values like AT and SLW at the input ports, and RAT and load at the output ports constitute the contract that allow a sub-block to be accurately optimized in isolation. Boundary assertions captured in these forms represent the signal characteristics from the IC run, and are the conditions that would allow the OOC run to model timing as though the sub-block was timed in the IC stage.

This paper presents an approach to generating representative external driving cell and external wire parasitic assertions for the ports of sub-blocks that enable accurate and robust early stage OOC timing optimization. The assertions are based on a technology lookup table. A concept of reverse timing calculation at the input of the driving cell is described that facilitates efficient timing optimization of boundary paths of design sub-blocks. Experimental results from an industrial timing environment demonstrate improved timing optimization accuracy when compared to prior work in [5].

II. Out-of-context input assertions

A. Fixed assertions

The accuracy of boundary path timing optimization in an OOC environment strongly depends on boundary assertions. In a traditional *fixed* style of boundary assertions, actual timing values (including ATs and SLWs) on macro ports (e.g., port p_3 in Fig.1) from the IC environment are captured and then applied on the corresponding ports as frozen timing values in the OOC timing environment.

Unfortunately, fixed assertions fail to accurately model timing on OOC boundary paths when compared to the IC timing during optimization. As an example, from Fig. 1, the arrival times and slews at p_3 are dependent on the RC (resistance and capacitance) parasitics of the hierarchical wire between p_2 and p_4. If either the *external* wire segment between p_2 and p_3 or the *internal* wire segment between p_3 and p_4 is changed during optimization, the arrival time and slew at p_3 would change relatively. This implies that the captured assertions at p_3 is accurate only if the internal wire and logic are unchanged, which is unlikely during OOC optimization.

During OOC timing, once the wire segment between p_3 and p_4 is optimized (e.g., re-routed or buffered), or the receiver gate G_2 is updated (e.g., re-sized), the assumption of a fixed port assertion does not hold, any timing data since then is not able to reflect the real timing. This is because the assertion is simply a surrogate for the timing from the true IC environment where the port is actually being driven by a gate that would react to any change internal to the sub-block design.

Optimizing a sub-block with fixed assertions may thus result in false timing closure at the OOC level and expose fails once the macro is instantiated at the IC level, thereby forcing multiple unproductive iterations of OOC optimization runs followed by new assertion generation from IC runs.

Fig. 2 and Fig. 3 illustrate the comparison of arrival times and slews, respectively, at the sink of an internal wire (e.g., pin p_4) during timing optimization of a sub-block being done in an OOC environment versus the same steps of timing optimization in an IC environment. The figures highlight significant miscorrelation in the timing values obtained between the two environments (up to 29% and 54% errors in arrival times and slew estimation, respectively), and motivates the need for assertions that can react to updates to the internal wire segment.

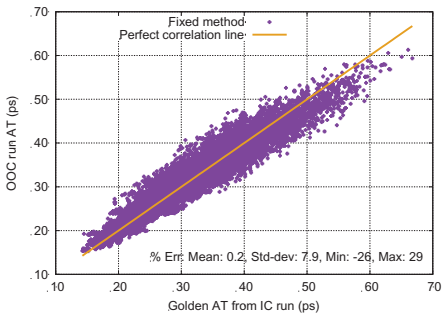

Fig. 2. Timing optimization accuracy correlation results for arrival time.

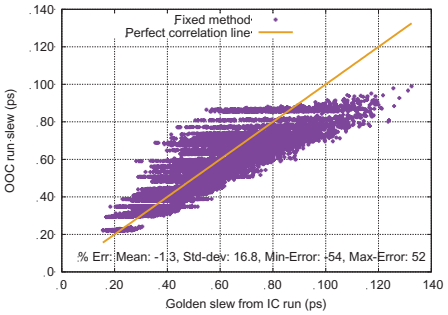

Fig. 3. Timing optimization accuracy correlation results for slew.

B. Assertions with driving cell and external wire parasitics

The unproductive iterations between the OOC and IC environments caused by timing miscorrelation when using fixed assertions can be overcome with an extension of the known prevailing idea of adding a driving cell [4] to input ports as part of assertions [5].

In this method, during an IC timing run, for each sub-block input port (e.g., p_3 in Fig. 1), relevant details (like cell type, cell power level, corresponding critical input and output pins) of the gate driving the port (e.g., G_1 in Fig. 1) are captured as part of the port assertions, so are the RC parasitics of the external wire connected to the port. These details are captured for all combinations of min and max timing, and individually for multiple clock domains.

Instead of capturing the timing values at the sub-block ports as assertions like in the *fixed* assertions method, the timing values at the driving gate's input (e.g., port p_1 in Fig. 1) are captured as fixed assertions. The goal is to capture sufficient detail from the IC environment for the OOC timing run so as to get perfect timing optimization accuracy for the boundary paths. During OOC timing, the captured external wire parasitics are dynamically *stitched* with the internal wire's current parasitics, and the captured driving cell is instantiated as the driver of the stitched wire segment. This is different from the industry standard approach of simply asserting a driving cell at an input port without any external wire.

The presented solution mimics the IC environment by having the driving cell drive the complete hierarchical wire. Any optimization change to the internal boundary wire segment forces a dynamic re-stitching of parasitics with the captured external wire's parasitics and results in dynamic recalculation of the timing to the receiver gate (e.g., G_2 in Fig. 1).

Fig. 4 and Fig. 5 illustrate the comparison of arrival times and slews, respectively, at the sink of an internal wire (e.g., pin p_4) during timing optimization of a sub-block being done in an OOC environment using assertions containing a driving cell with external wire parasitics versus the same steps of timing optimization in an IC environment (results from which are considered golden). As expected, the figures highlight perfect correlation in the timing values obtained between the two environments when using assertions with the driving cell and external parasitics.

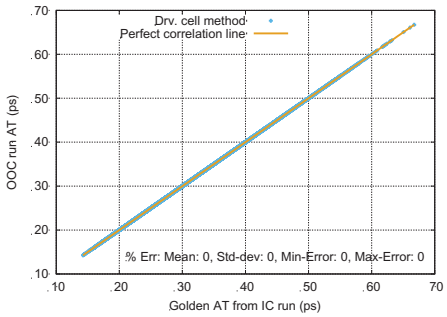

Fig. 4. Timing optimization accuracy correlation results for arrival time.

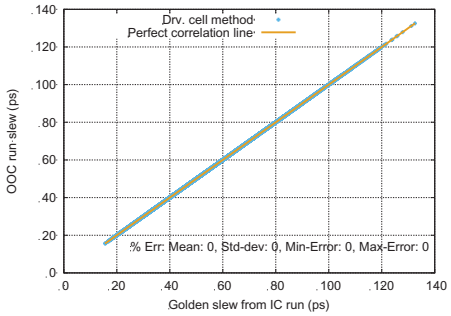

Fig. 5. Timing optimization accuracy correlation results for slew.

C. *Limitations of assertions with driving cell and external wire parasitics*

The method of capturing assertions in section II-B assumes that the IC design is mature enough to obtain driving cell and external wire parasitics that will meaningfully guide OOC boundary path optimization. On the contrary, external paths are estimated during early stages of design since full detailed routing is unnecessary before sub-blocks become mature. Capturing unrealistic parasitics will be incorrectly guiding OOC optimization until a future iteration of IC timing with a buffered external wire generates an updated set of assertions.

Fast time to market needs dictate concurrent OOC and IC design optimization for modern hierarchical chip designs. During early stages of chip design, coarse grain optimization at the IC level causes significant churn to fixed assertions. OOC timing optimization at this stage uses contract-based assertions instead of frozen timing values from the IC environment to reduce unnecessary timing iterations and to ensure stability. This implies that the accuracy advantages of the driving cell and external parasitics method cannot be applied in early stages of the design optimization flow, which can span more than 75% of the design life cycle.

There is naturally a need for an assertion method that improves from the accuracy of the fixed assertion method and is usable in the early stages of the design flow. In addition, the driving cell method in section II-B fails when the driver of the hierarchical wire is a hard-IP block, and forces defaulting to the fixed assertion approach. This motivates the need for representative driving cell and external wire parasitics based assertions that can be employed for higher accuracy in early design stage OOC timing till the IC design is mature enough to provide exact driving cell and parasitics based assertions.

TABLE I.
TECHNOLOGY LOOKUP TABLE ILLUSTRATION

WireCode	R	C	Cell	Input Cap	Reach Length	TOF	STOF
C5_W10_S11	14.47	0.22	BUF_X10	3.51	0.07	378.74	405.97
H1_W30_S15	0.09	0.38	BUF_X64	22.97	0.52	76.70	79.50
…							
M3_W10_S15	18.65	0.22	BUF_X8	3.51	0.07	405.38	431.70

III. DEFAULT DRIVING CELL AND WIRE BASED ASSERTIONS

A. *Technology lookup table*

A concurrent design flow paradigm for leading edge microprocessor designs is often coupled with a new device technology application. As part of technology exploration for design planning and routing estimation, a technology-based lookup table is generated to provide representative information about thousands of potential wiring and buffering strategies. Using Table I as an example, each entry represents a buffering strategy at certain wire layer, which contains layout wirecode (metal *Layer, Width* and *Spacing* combination), resistances and capacitances per unit length for that the wirecode choice, buffer (*Cell*) to be used to drive that wire with its optimized driving distance (*Reach Length*), time of flight (*TOF,* delay per unit length) & slew time of flight (*STOF,* slew change per unit length) along the wire and typical sink pin capacitance. With buffering solutions in lookup table, design team can evaluate the pin-to-pin delays/slews and routing resource usage before a real layout is implemented. The size of table depends on the technology, number of metal layers, wire width/spacing combinations and buffer variations in library. The example table listed above can have over 5K entries.

The technology lookup table is updated frequently to always sync up with technology refinements and design goals. Given a wirecode from a wire connected to a port of a sub-block, typical buffering solution can be obtained from the technology lookup table, and also the driving cell and external wire parasitics for default driving cell usage.

B. *Assertions with default driving cell and default external wire parasitics*

Given wirecode in the technology lookup table that are obtained from attributes found on a wire connected to a sub-block port, the recommended buffer value found (*Cell* in Table I) is used to model the *default* driving cell for that port during early stages of OOC timing optimization. In addition, the typical wire-length (l) and expected per unit length resistances and capacitances ($\frac{dR}{dL}, \frac{dC}{dL}$) from the table facilitate the creation of a *default* distributed RC parasitic model for the external wire. A N piece distributed model is assumed, where each piece i is modeled as a RC-pi section with parasitic values calculated as:

$$R_{\pi_i} = \frac{l}{N} \times \frac{dR}{dL} \times f, \text{ and} \quad (1)$$

$$C_{\pi_i}^{near} = C_{\pi_i}^{far} = \frac{l}{2N} \times \frac{dC}{dL} \times f, \quad (2)$$

where, f denotes a user or design specific factor that is used to scale the typical wire length l. In practice, a N piece model with *5* or more pieces is found to be accurate in comparison to an ideal distributed wire model. This allows the use of generated default driving cell and external wire parasitics as part of OOC assertions similar to that in Section II-B during

early OOC timing optimization. This approach can be used even in mature IC design stages where the exact driving cell in the IC environment is found to be a hard-IP block to obtain a default driving cell while still using the exact external wire parasitics.

C. Output port load assertions

Traditional OOC assertions at the output ports of a sub-block include load models that represent the external wire's RC parasitics. As an example, from Fig. 1, the load assertion for port p_7 represents the parasitics for the external wire that connects that port to gate G_4 in the IC environment. The load assertions are traditionally captured from an IC timing environment similar to the approach of capturing input assertions. These assertions may be captured as a single lumped capacitance or as a reduced order RC model (e.g., RC-pi model).

The accuracy of OOC timing optimization of an output boundary path (e.g., gate G_3 in Fig. 1) depends on the accuracy of the output port's load assertion and how closely it models the actual parasitics of the external output wire (between ports p_7 and p_8 in Fig. 1) from the IC environment. While the external wire parasitics can be accurately captured during mature stages of the IC timing environment, load assertions during early design stages are forced to be designer-specified lumped capacitances which are often inaccurate.

A *default* distributed output load model for the external wire can be generated from the technology lookup table in a manner similar to that described in Section III-B for the input side. Given wirecode in the technology lookup table that is obtained from attributes found on a sub-block output port or on the connected wire, the typical wire-length (l) and expected per unit length resistances and capacitances ($\frac{dR}{dL}, \frac{dC}{dL}$) from the table facilitate the creation of a *default* distributed RC parasitic model for the external wire. A N piece distributed model is assumed, where each piece is modeled as a RC-pi section with parasitic values calculated as described in (1) and (2). In addition, the receiver gate input capacitance C_r from the table is obtained to augment the distributed RC model by adding C_r to the $C_{\pi_N}^{far}$ value of the N^{th} RC-pi section. The distributed RC model is converted to a reduced RC-pi model and then used as the load model assertion in the OOC timing run.

This conversion is performed efficiently using closed form expressions without actually creating the distributed model. The total capacitance C_t and resistance R_t for the distributed RC wire (without considering C_r) is initially calculated as follows.

$$R_t = l \times \frac{dR}{dL} \times f, \text{ and} \quad (3)$$

$$C_t = l \times \frac{dC}{dL} \times f, \quad (4)$$

where, f denotes a user or design specific factor that is used to scale the typical wire length l. Based on [6], the parameters for the final RC-pi model including C_r are calculated as:

$$C_\pi^{far} = \frac{y_2^2}{y_3}, \quad (5)$$

$$R_\pi = -\frac{y_3^2}{y_2^3}, \text{ and} \quad (6)$$

$$C_\pi^{near} = y_1 - C_\pi^{far}, \text{ where} \quad (7)$$

$$y_1 \triangleq C_t + C_r, \quad (8)$$

$$y_2 \triangleq -R_t(C_r^2 + C_t C_r + 0.33C_t^2), \text{ and} \quad (9)$$

$$y_3 \triangleq R_t^2(C_r^3 + 1.33C_t C_r^2 + 0.67C_r C_t^2 + 0.133C_t^3). \quad (10)$$

This facilitates the efficient generation of *default* output load assertions for early stages of OOC timing optimization and significantly improves the accuracy over traditional lumped capacitance-based load assertions.

IV. Reverse timing calculation

Section III-B described an approach for obtaining default driving cell and default external wire parasitic assertions for a sub-block input port using the technology lookup table and wire attributes. While this is conceptually akin to the exact driving cell and external parasitics-based assertion approach described in Section II-B, the latter approach additionally captures the timing at the input pin of the driving cell from the IC environment as part of the input assertions. With the default assertion generation approach, there is no actual IC timing environment with that driving cell and the default external wire parasitics. Consequently, it is not possible to capture the timing at the input of the default driving cell from an existing IC timing environment. A method to determine the timing values at the input of the default driving cell is therefore needed.

Given desired arrival time and slew assertions from hierarchical timing contracts for a sub-block input port, reverse timing calculation involves applying the default driving cell and external parasitics at that port, performing wire parasitics stitching of the (default) external and the internal wires, and then computing a slew which when applied to the input of the default driving cell produces the desired slew at that port. As an example, referring to the OOC environment for the sub-block in Fig. 1, and assuming that gate G_1 is the default driving cell, the goal is to compute a slew at the input pin (p_1) of G_1 that when propagated downstream to the port p_3 results in a slew that matches the desired contract slew for p_3.

This analysis is performed in an iterative fashion using binary search between the slew limits (or thresholds) defined for that default driving cell. This reverse slew calculation happens only once per port at the time of default assertion processing. Once the slew computation at the input of the driving cell has converged to a final value, the signal delay through the default driving cell and the default external wire is calculated. This delay is then subtracted from the desired arrival time at the sub-block input port (obtained from hierarchical timing contracts) to obtain the arrival time at the input of the driving cell. The iterative approach is employed only for the slew calculation while the subsequent arrival time calculation does not involve any iterations. The final slew and arrival time obtained at the input of the default driving cell completes the full input assertion set and can be used for subsequent OOC timing optimization of the sub-block similar to that described in Section II-B.

In practice, it is observed that the reverse slew calculation requires between 3 and 8 iterations to converge to the desired value. Since this is done as part of early stage timing optimization, allowing a small error tolerance during iterations allows faster convergence without impacting accuracy.

V. Results

Experiments are performed in an industrial timing tools framework for multiple sub-blocks of a 14-nanometer technology mapped microprocessor design. For each sub-block, fixed assertions are initially obtained from the IC environment along with the exact driving cell and the exact external wire parasitics for the driving cell-based assertion approach. Two instances of the OOC environment are set up with fixed assertions and exact driving cell-based assertions, respectively. In addition, the IC environment is set as the golden reference run for comparisons. Thousands of timing optimization moves of input boundary paths are subsequently performed identically across the two OOC instances and the IC environment. The performed optimization moves include wire re-routing, buffering and receiver gate re-sizing.

The timing impact from each optimization move is captured by observing the arrival times and slews at the sink of the hierarchical input wire (like pin p_4 in Fig. 1) for all the three runs. Normalized arrival time and slew comparisons for the two OOC environments with respect to the golden IC environment were illustrated in Fig. 2-Fig. 5. The figures clearly highlight the near perfect accuracy of the exact driving cell assertion approach and large inaccuracies of the fixed assertions approach.

To quantify OOC timing optimization accuracy with a default driving cell, an additional instance of the OOC environment is set up with a default driving cell, but with exact external wire parasitics obtained from the IC environment. This mimics the case when the exact driving cell cannot be used for the OOC driving cell assertion (e.g., case of a hard-IP block as a driver). Timing optimization moves are performed as before.

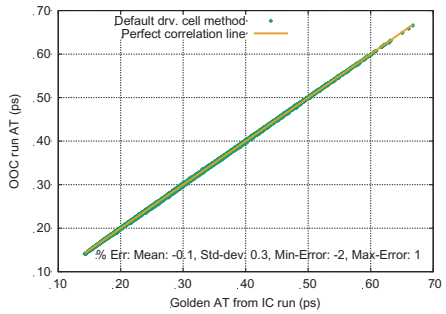

Fig. 6. Timing optimization accuracy correlation results for arrival time.

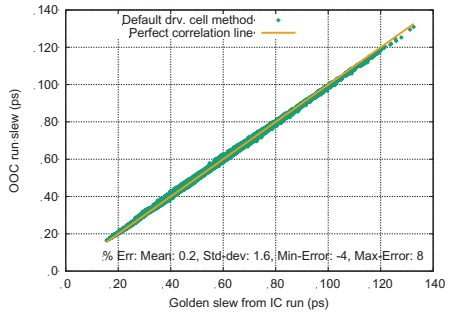

Fig. 7. Timing optimization accuracy correlation results for slew.

The impact of each optimization move on the arrival times and slews at the sink of the hierarchical wire are observed for the third OOC instance and compared with the IC environment in Fig. 6 and Fig. 7, respectively. It is observed that a default representative driving cell suffers from minimal accuracy loss with worst miscorrelations of less than 2% for the arrival time and less than 8% for the slew.

To test timing optimization accuracy with a default driving cell as well as default external wire parasitics, the exact wire parasitics from the aforementioned OOC environment are removed and replaced with representative default wire parasitics. Timing impact observations are made as before following multiple timing optimization moves and illustrated in Fig. 8 and Fig. 9. While the timing accuracy with default assertions is found to be worse than the exact driving cell based assertions, this is expected and is still significantly better than the fixed assertions approach which is the traditional approach for early stages of design optimization.

The statistics of the timing correlations for the arrival times and slews across the different OOC scenarios are summarized in Table II and Table III, respectively. Considering the timing from the IC environment as golden, the difference of arrival times and slews between each of the OOC instances and the golden is presented as a % of the golden value.

The fixed assertions approach has errors as high as 54%, while just the default driving cell approach reduces the worst-case error to 8%. The default driving cell and default external wire parasitics approach show errors of up to 17% which is reasonable given its scope of usage being early stages of timing optimization.

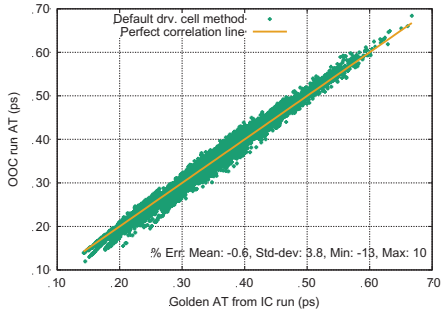

Fig. 8. Timing optimization accuracy correlation results for arrival time with default parasitics.

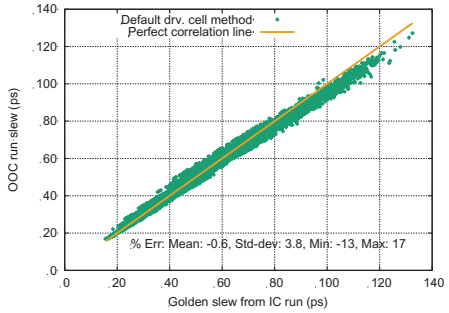

Fig. 9. Timing optimization accuracy correlation results for slew with default parasitics.

TABLE II.
TIMING OPTIMIZATION ARRIVAL TIME ACCURACY COMPARISON

	Error w.r.t in-context optimization (%)			
	Mean	Standard deviation	Min	Max
Fixed assertions	0.2	7.9	-26	29
Exact driving cell and exact parasitcs	0.0	0.0	0	0
Default driving cell and exact parasitics	-0.1	0.3	-2	1
Default driving cell and default parasitics	-0.6	3.8	-13	10

TABLE III.
TIMING OPTIMIZATION SLEW ACCURACY COMPARISON

	Error w.r.t in-context optimization (%)			
	Mean	Standard deviation	Min	Max
Fixed assertions	-1.3	16.8	-54	52
Exact driving cell and and exact parasitcs	0.0	0.0	0	0
Default driving cell and exact parasitics	0.2	1.6	-4	8
Default driving cell and and default parasitics	-0.6	3.8	-13	17

Applying driving cell method to sub-blocks OOC timing would increase accuracy with little extra cost. Simple cells are used to be stitched at outside of input ports, which adds little complexity to existing logic graph. Compared with internal complicated logic, driving cells would only add one more signal propagation at inputs to overall timing analysis. For an unfriendly scenario, a test circuit with four buffers and three input ports, driving cell method adds extra 2% runtime and 0.02% memory increase compared to regular timing analysis.

VI. CONCLUSIONS

This paper presents an approach to dynamically generating representative *default* driving cell and *default* external wire parasitic assertions for the input and output ports of sub-blocks of a hierarchical design. The assertions are based on a technology lookup table and use attributes of the port and the hierarchical wire connected to the port as keys. A concept of reverse timing calculation at the input of the driving cell is described that facilitates the approach to drive efficient timing optimization of boundary paths of design sub-blocks during early design stages.

The default driving cell-based assertion approach fills a major step gap between the fixed assertions and the exact driving cell-based assertions approach for early timing optimization which spans more than 75% of the design cycle. This method has been productized in a commercial timing tools environment successfully and shows significant turnaround time improvements of timing optimization across hierarchies and reduces unproductive iterations of timing closure between the IC and OOC levels. Experimental results demonstrate significantly improved timing optimization accuracy (worst errors reduced from 54% to 17%) when compared to a fixed assertions-based approach.

The presented method can be used in a hybrid fashion with the exact driving cell assertion approach in cases where the driving gate at the IC level is a hard-IP block. This method is also used when the OOC design introduces new input ports that are not yet visible at the IC design stage.

APPENDIX

A. Extensions for statistical timing

The general concept of sub-block input and output port assertion generation is the same for single corner timing as well as statistical timing [7]-[9]. In the latter case, timing values are captured as a statistical random variable with a known probability distribution instead of a single (deterministic) value. The approaches to capturing assertions involving exact and/or default driving cell and external parasitics are also unaltered.

The reverse calculation approach for default assertions as described in Section IV is more complex for statistical timing since obtaining a statistical slew at the input of the default driving cell that would produce the desired contract based statistical slew at the sub-block port is non-trivial. To overcome this problem, the desired port statistical slew is sampled at multiple corners of the variation space to obtain deterministic slews corresponding to each corner. For each corner, the reverse calculation approach from the prior section is employed to obtain a deterministic slew at the input of the default driving cell (corresponding to that corner). Finally, all the deterministic slews at the driving cell input are combined into a statistical parameterized form using standard concepts of statistical finite differencing. The statistical path delay through the default driving cell and the default external wire is then calculated and subtracted from the desired statistical arrival time at the sub-block port to obtain the statistical arrival time at the input of the default driving cell.

REFERENCES

[1] Hierarchical Timing Analysis: Pros, Cons, and a New Approach: *https://www.cadence.com › dam › documents › tools › digital-design-signoff*.

[2] C. W. Moon, H. Kriplani, and K. P. Belkhale, "Timing model extraction of hierarchical blocks by graph reduction," in *DAC*, 2002, pp. 152–157.

[3] S. Zhou, Y. Zhu, Y. Hu, R. Graham, M. Hutton, and C. Cheng, "Timing model reduction for hierarchical timing analysis," in *ICCAD*, 2006, pp. 415-422.

[4] Synopsys, "PrimeTime user guide".

[5] D. Sinha, R. Ledalla, C. Li, T. Ko, A. Bhanji, and H. Gupta, "Load-aware assertion generation for sub-blocks in hierarchical timing optimization and sign-off," in *DAC* 2018 [designer track].

[6] P. O'Brien and T. Savarino, "Modeling the driving-point characteristic of resistive interconnect for accurate delay estimation," in *ICCAD*, 1989, pp. 512-515.

[7] H. Chang and S. S. Sapatnekar, "Statistical timing analysis considering spatial correlations using a single PERT-like traversal," in *ICCAD*, 2003, pp. 621–625.

[8] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, and S. Narayan, "First-order incremental block-based statistical timing analysis," in DAC, 2004, pp. 331–336.

[9] V. Zolotov, D. Sinha, J. Hemmett, E. Foreman, C. Visweswariah, J. Xiong, J. Leitzen, and N. Venkateswaran, "Timing analysis with nonseparable statistical and deterministic variations," in *DAC*, 2012, pp. 1061–1066.