Synthesis of All-Digital Delay Lines

Alberto Moreno and Jordi Cortadella
Department of Computer Science, Universitat Politècnica de Catalunya, 08034 Barcelona, Spain.

***Abstract*—The synthesis of delay lines (DLs) is a core task during the generation of matched delays, ring oscillator clocks or delay monitors. The main figure of merit of a DL is the fidelity to track variability. Unfortunately, complex systems have a great diversity of timing paths that exhibit different sensitivities to static and dynamic variations. Designing DLs that capture this diversity is an ardous task. This paper proposes an algorithmic approach for the synthesis of DLs that can be integrated in a conventional design flow. The algorithm uses heuristics to perform a combinatorial search in a vast space of solutions that combine different types of gates and wire lengths. The synthesized DLs are (1) all digital, i.e., built of conventional standard cells, (2) accurate in tracking variability and (3) configurable at runtime. Experimental results with a commercial standard cell library confirm the quality of the DLs that only exhibit delay mismatches of about 1% on average over all PVT corners.**

I. Introduction

Delay lines (DLs) have been used in different contexts to track the increasing variability of integrated circuits as CMOS advances to smaller technology nodes. The main goal of a variability-tracking DL is to have a circuit that generates a delay highly correlated with the longest timing path of the system. DLs are often used for post-silicon tuning [1]–[4], thus enabling the reduction of guardband margins.

One of the potential uses for DLs is in bundled-data (BD) asynchronous circuits [5] where DLs are inserted in the paths of the handshake signals (req/ack) that synchronize different modules of the system. For a correct operation, delays need to be longer than the critical path yet as small as possible to prevent performance degradation.

The notion of Representative Critical Path (RCP) is used in [1] for the synthesis of a DL highly correlated with the circuit delay. Two algorithms are proposed for designing RCPs based statistical static timing models for variability rather than using the more conventional static timing analysis (STA).

Delay monitors, such as canary paths, are also built with DLs [6]. In [7], a comprehensive survey can be found. An algorithmic technique is also introduced for designing Ring Oscillators (RO) for circuit performance monitoring. The approach of [7] simplifies the design of DLs by considering only blocks of identical gates and specific interconnect lengths as the basic building element. This allows to ignore variations in slew propagation and capacitance between blocks. With this simplification, the problem can be modeled by an integer linear program, at the cost of losing flexibility and precision.

DLs can be used for the design of Ring Oscillator Clocks (ROCs) [8]. An ROC is a ring oscillator used as clock generator and with a variability highly correlated to the critical paths of the circuit. ROCs are proposed as a substitute of the classical Phase-Locked Loop (PLL). This allows a significant reduction of guardband margins. In particular, it provides a robust scheme for instantly adapting to voltage droops without the need of introducing large margins.

*This work has been partially supported by funds from the Spanish Ministry for Economy and Competitiveness and the European Union (FEDER funds) under grant TIN2013-46181-C2-1-R, the Generalitat de Catalunya (2014 SGR 1034 and FI-DGR 2015).

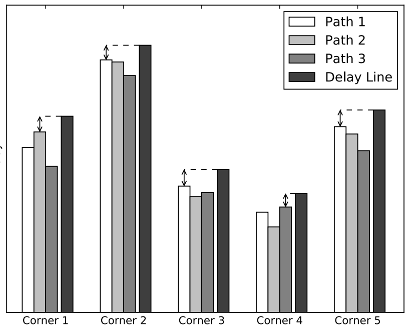

Fig. 1. Several timing paths and delay line at different PVT corners.

DLs can also benefit from post-silicon tuning to reduce margins after chip manufacturing by adjusting the delays. There are several ways of accomplishing this, including analog and digital techniques. On the analog side, voltage-controlled delay elements are typically used [9]–[11]. Digitally-controlled delay elements are also possible, for example, by interleaving multiplexers in the DL [4], [12].

All these schemes share the need to accurately match the delay of a DL with timing paths that exhibit PVT variability. Using the terminology of STA, we can say that different timing paths may have different criticality at different PVT corners. Therefore, designing a DL by simply replicating a timing path of the circuit is not always a good approach for delay matching.

A typical situation of time criticality is depicted in Fig. 1. The histogram shows the delay of three different paths (Path 1-3) at five different PVT corners (Corner 1-5). Due to the different sensitivities to PVT variations, none of them can be taken as a representative of the time criticality of the circuit.

In general, the number of critical paths (with small slack) tends to be extremely large. The main reason is that physical design tools amortize the available time slacks to reduce power by undersizing non-critical gates. In this context, synthesizing a DL that is, at the same time, reliable and accurate at all corners is a challenging problem. The figure also illustrates the desirable properties for a DL:

- It must be longer than the longest delay at any corner (within a certain guardband margin).
- It must be as short as possible to minimize performance degradation.

It is also desirable that DLs can be synthesized and analyzed using conventional standard cell libraries and design automation flows. In this way, the use of DLs can be leveraged in a broader spectrum of application domains.

All the previous requirements pose a challenge for the design of DLs that must address several aspects:

- How to extract the timing characteristics of a circuit at all PVT corners without enumerating all critical paths?
- How to build a chain of heterogeneous standard cells that mimic the timing behavior of the circuit under different PVT conditions?

978-1-5386-2749-5/17 $31.00 © 2017 IEEE
DOI 10.1109/ASYNC.2017.10

- How to take into account the variations introduced by the interconnect components (wires)?
- How to make the DL configurable?

This paper proposes algorithmic techniques for the synthesis of all-digital DLs with the following characteristics:

- The DLs only contain cells from a standard cell library. No custom cells or analog components are used.
- The timing of the DLs is analyzed by conventional STA tools using library corners and derating factors to model PVT variability.
- The design of DLs includes physical synthesis. In particular, an algorithm for cell placement and derivation of routing constraints for interconnects is proposed.
- The DLs include configurable delays for post-silicon tuning.

The area and power consumption of the DLs can be considered negligible when used for coarse-grain control, e.g., large clock domains or complex functional units.

A. Relevance of the problem

Fig. 2 illustrates the importance of designing DLs with a mixed combination of gates and wires to accurately track variability at different operating conditions. The algorithm proposed in this paper was used to generate DLs for the ITC99 benchmarks [13]. For the selection of the DL cells, three scenarios were considered: (1) only using one type of inverter (i.e. all the cells are identical), (2) using a mix of inverters of different size and (3) using a mix of combinational cells in the standard cell library. The algorithm tried to find the best match for each scenario.

A commercial 65nm library was used to map all reported circuits. Variability was modeled by considering 22 different PVT corners with temperatures in the interval $[-40^oC, 125^oC]$, power supply in the range $[0.9V, 1.32V]$ and process parameters including SS, TT and FF models for transistors. The $RC_{\min}$ and $RC_{\max}$ corners were used to model the variability of the interconnect layers.

The figure depicts the average discrepancy (mismatch %) of the DLs with regard to the delay of the ITC99 circuits [13] mapped onto the library. The average was calculated over the delays reported by STA (Synopsys PrimeTime [14]) at all available corners of the library (more details in Sect. VI).

It can be observed that matching delays with only one type of inverter may result in a large mismatch (e.g., 20% for b13). Using a mix of inverters with different size may mitigate the mismatch significantly (6% for b13). Finally, the use of a mix of gates with large diversity may contribute to have a good match at all corners (1% on average for b13).

Table I also reports the usage of each cell type in the DLs when any type of cell was used for synthesis. We can observe that more than half of the gates are not inverters. It is precisely this diversity what allows a better matching at different operating conditions. It is important to emphasize that the DLs do not only select a mix of gates, but also a mix of wire lengths between neighboring cells to account for interconnect variability. The details will be described in the paper.

Fig. 3 depicts an example of DL synthesized to match the delay of one of the experimental circuits (b05). The picture shows the diversity of gates and sizes used in the DL that contribute to mimic the delay of the circuit more accurately at different operating conditions[1].

[1]The numbers inside the gates indicate the size of the cells.

TABLE I
GATE TYPE USAGE IN DELAY LINES

Gate	Usage	Gate	Usage	Gate	Usage
INV	42.2%	CKND2	4.1%	AO221	0.4%
NAND3	18.6%	NAND4	1.9%	XNOR2	0.4%
NOR2B1	13.5%	NOR2	1.0%	OAI222	0.4%
CKINV	6.6%	NAND2B1	0.6%	OA211	0.3%
NAND2	4.9%	AOI21B20	0.5%	Others	4.6%

TABLE II
DELAY LINE STAGE PARAMETERS

c	Corner from the set of CORNERS
$d_{c,i}$	Delay of stage i at corner c
$C_{c,i}$	Output capacitance of stage i at corner c
$S_{c,i}$	Input slew of stage i at corner c
$w_{c,i}$	Wire delay of stage i at corner c

II. NOMENCLATURE, PROBLEM STATEMENT AND OVERVIEW

The problem we want to solve is the synthesis of a DL that matches the delay of a circuit under any potential operating conditions. In our context, variability is modeled using the same PVT corners and derating factors used during conventional STA to model global variability and on-chip variability (OCV)[2].

Using STA, the delay of the most critical path at each corner is obtained. However, any information about the particular critical path that generates the longest delay is disregarded, bearing in mind that each corner may exhibit different critical paths and the particular structure of each critical path is irrelevant. We will call Dmax_c the longest delay at corner c.

With this information, and the use of OCV derating factors, a set of target delays T is derived. This set contains, for each corner c, the ideal delay $\tau_c \in T$ of the DL for that corner. Formally:

$$\tau_c = \delta \cdot \mathit{Dmax}_c. \tag{1}$$

with $\delta > 1$ being the OCV derating factor[3].

Fig. 4 shows a representation of a DL, which is a sequence of gates and wires. Each pair gate/wire will be referred to as a stage of the DL. Each stage i has an output capacitance C_i, an input slew S_i and a delay d_i. For the sake of simplicity in the nomenclature and the description of the algorithm, we will not distinguish between falling and rising delays. However, they are considered in the actual algorithms and results reported in the paper.

Each stage i is characterized by the parameters defined in Table II, where c represents the PVT corner at which the parameters are measured. The delay for stage i is computed as the sum of the gate and wire delays. The gate delay and the output slew are functions of the input slew and output capacitance:

$$d_{c,i} = \mathit{GateDelay}_c(S_{c,i}, C_{c,i}) + w_{c,i}; \quad S_{c,i+1} = \mathit{Slew}_c(S_{c,i}, C_{c,i}).$$

The output capacitance for stage i is the sum of the input capacitance for stage $i+1$ and the wire capacitance of stage i.

[2]An in-depth discussion about derating factors for DLs can be found in [15].
[3]For simplicity, we assume a unique δ for all corners. However the proposed approach can be easily extended to different values of δ for each corner.

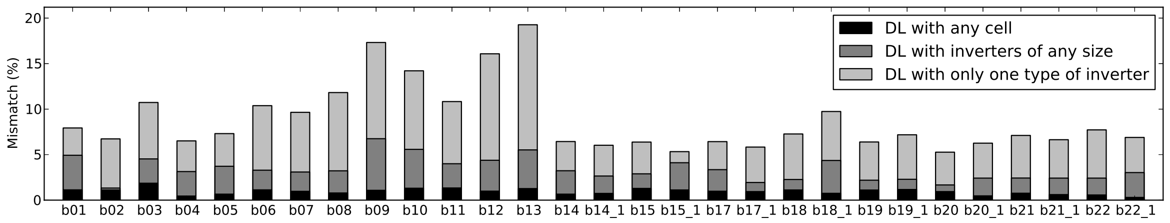

Fig. 2. Accuracy of a DL when using only inverters or any cell in the library. The Y-axis represents average mismatch.

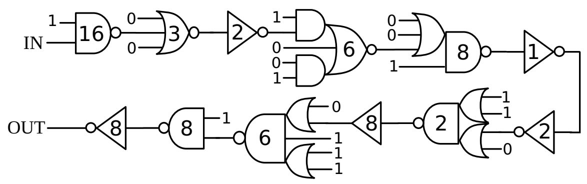

Fig. 3. DL obtained for matching the delay of b05.

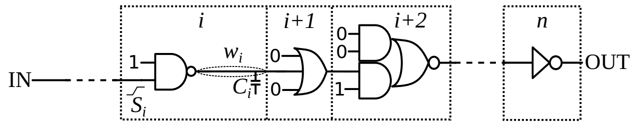

Fig. 4. Stages of a delay line.

The delay of a DL of n stages at corner c is obtained by adding the delays of all stages:

$$\mathit{delay}_c(DL) = \sum_{i=1}^{n} d_{c,i}$$

Given a set of target delays $\{\tau_c\}$, we can define the *delay mismatch* of a DL at each corner c:

$$\mathit{Mismatch}_c(DL) = \mathit{delay}_c(DL) - \tau_c.$$

It is important to notice that the mismatch is computed on a delay that has already been derated to take into account on-chip variability (equation (1)). For the algorithm, it is also convenient to define a normalized version of the mismtch:

$$\mathit{NormMismatch}_c(DL) = \frac{\mathit{Mismatch}_c(DL)}{\tau_c}. \tag{2}$$

Delay constraint: For a DL to be correct, it should be always longer than the target delay. Therefore, the following property must hold for any valid DL:

$$\forall c \in \textsc{Corners}: \quad \mathit{Mismatch}_c(DL) > 0. \tag{3}$$

Cost function: A cost function is needed to guide the exploration of the DL structure during the execution of the synthesis algorithm. The cost function is responsible for reducing the mismatch between the DL and the delay of the circuit at different corners. Depending on the context, various cost functions may be considered. Here we present a generalized formulation that can be customized for different application domains:

$$\mathit{Cost}(DL) = \sum_{c \in \textsc{Corners}} \omega_c \cdot \mathit{NormMismatch}_c(DL)^{\alpha} \tag{4}$$

with ω_c being a set of weights associated to each corner and α being a constant to control the mismatch diversity. For example, if the designer would prefer to minimize the mismatch at the typical corner, at the expense of having more mismatch at other corners, then the weight ω_{typ} should be increased. If α has a small value (e.g., $\alpha = 1$), then the cost function will guide the exploration towards minimizing the average mismatch over all corners. Instead, if a large value is used (e.g., $\alpha = 3$), the cost function will guide towards minimizing the maximum mismatch over all corners.

The algorithm presented in this paper is independent of the cost function used for optimization. Therefore, the designer can propose her/his own customized cost function.

Problem statement: The synthesis problem consists of finding a sequence of gates and wires to build a DL with the following goal:

$$\begin{aligned} \text{minimize:} \quad & \mathit{Cost}(DL) \\ \text{subject to:} \quad & \text{Constraint } (3) \end{aligned}$$

Exploration space: The space of potential configurations for a DL is determined by the number of gates in the library (G) and the set of wire configurations for each stage (W). Unfortunately, W is infinite: any sequence of segments of different length using different layers could be potentially used to connect two consecutive gates. To prune the search space, only a small subset of wire configurations is defined a priori to cover a reasonable spectrum of wire lengths.

As an example, the results presented in this paper have been obtained by considering wires with length 5, 12, 25, 50 and $100\mu m$ (the height of a standard cell is $1.8\mu m$). More details about the gate and wire delay models will be given in Sect. III-A.

Still, with G and W being finite, the possible set of configurations of a DL with N stages is $(|G| \times |W|)^N$, which makes an exhaustive exploration impractical, bearing in mind that N is unknown and can potentially be a large number (e.g., $N > 50$ in some of the examples reported in Sect. VI).

Overview of the DL synthesis flow: The algorithmic strategy to generate a DL is decomposed into four steps:

1) Selection of gates and wire lengths that will constitute the DL (algorithm presented in Sect. III).
2) Physical placement of the gates (Sect. IV).
3) Routing of wires using conventional EDA tools.
4) Timing sign-off with STA tools. If some timing violation is produced, the target delay is slightly adjusted and steps 1-4 are executed again until no violation occurs.

Steps 1 and 2, described later in this paper, use simplified delay models to synthesized the DLs. Step 4 ensures that DLs will always meet constraint (3) using the same timing models as the STA tools.

III. Algorithm for gate and wire selection

The synthesis of a DL is a combinatorial optimization problem. In this paper we present a heuristic algorithm based on the *Beam Search* paradigm [16]. Beam Search is based on a constant parameter β (beam width) and explores a search tree by keeping β partial

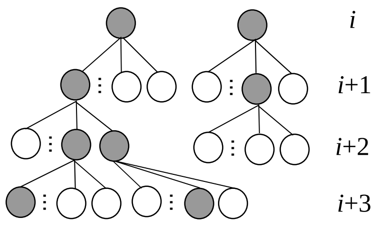

Fig. 5. Beam Search with $\beta = 2$ showing the search levels $i \ldots i+3$. The selected candidates are shadowed.

solutions at each level selected from all the solutions generated from the previous level. A heuristic cost function is used to select the β best solutions. Fig. 5 shows a search example with $\beta = 2$.

For the synthesis of DLs, each tree level i stores partial solutions with i gates. When all the generated solutions meet constraint (3), the search is aborted and the best solution is delivered.

For the details of the algorithm, it is important to define two new concepts:

- **Partial delay line** (PDL): any DL with zero or more stages.
- **Final delay line** (FDL): any PDL that meets constraint (3).

Algorithm 1 shows the main loop of the synthesis algorithm. Initially, the set of PDLs is initialized with a 0-stage DL (level 0 of the search tree) and the set of FDLs is empty. At each iteration of the main loop, each element in PDL is extended by one stage and the β best solutions are stored, according to the cost function described later in Algorithm 3. The extension is performed by the function EXTENDDELAYLINES, described in detail by Algorithm 2.

Algorithm 1: BEAMSEARCH(β)

```
begin
    dl = DL with 0 stages
    FDL = ∅                                   // Set of FDLs
    PDL = {dl}                                // Set of PDLs
    while not Empty(PDL) do
        // Generate next level of DLs
        PDL, FDL = EXTENDDELAYLINES(PDL, FDL)
        PDL = select the $\beta$ best DLs from PDL
    return the best DL in FDL
```

The function EXTENDDELAYLINES generates the next level of the search tree by adding a new gate g and a wire w to the PDLs generated in the previous level. *Wires* contains a discrete variety of wire lengths. The number of new solutions is $|\text{PDL}| \times |Gates| \times |Wires|$, from which the Beam Search algorithm will select the β best solutions. If any of the new solutions meets constraint (3), it is stored in the set of final solutions (FDL).

Finally, Algorithm 3 shows the function that computes the cost of each PDL. The function estimates the accuracy of a PDL if the current delays would be scaled linearly to meet constraint (3). First, a scaling factor s is calculated that corresponds to the smallest factor required to meet constraint (3) at each corner. Next, the normalized mismatch is computed for each corner using the scaled delays. Finally, the cost of the DL is estimated using the scaled mismatches and the cost function (4).

A. Gate and wire delay models

The models used during the synthesis of DLs are identical to the ones used for STA. Each library uses one or more delay models (e.g.,

Algorithm 2: EXTENDDELAYLINES(PDL, FDL)

```
input : A set of PDLs and FDLs stored in PDL and FDL,
        respectively
begin
    newPDL= ∅ // Stores next level of the tree
    foreach $dl \in$ PDL do
        foreach $g \in Gates$ do
            foreach $w \in Wires$ do
                dl' = addStage(dl, g, w)
                if dl' meets constraint (3) then
                    $FDL = FDL \cup \{dl'\}$
                else
                    $newPDL = newPDL \cup dl'$
    return newPDL, FDL
```

Algorithm 3: COST(*dl*)

```
begin
    // $s'$ is a vector of scaling factors
    foreach $c \in$ CORNERS do
        $s'[c] = \tau_c / delay_c(dl)$
    $s = \max(s')$                     // scale factor
    // Vector of scaled normalized mismatches
    foreach $c \in$ CORNERS do
        $NormMismatch[c] = (s \cdot delay_c(dl) - \tau_c)/\tau_c$
    // Apply the cost function (4)
    return CostFunction(NormMismatch)
```

NLDM, CCS, ECSM). One of the simplest is NLDM, which is the one used in this paper for the experiments. However, the delay model is only used in the evaluation of the cost function and the heuristic exploration can easily adopt any other model.

For NLDM, each timing arc defines, for each transition direction, a transition time (slew) and a delay table. These tables are indexed by the output capacitance and input slew. The delay and output slew are calculated by a bilinear interpolation.

Libraries also include wire models. The main parameters that affect wire delays are capacitance, resistance and crosstalk. For a set of technological parameters (e.g., resistance/capacitance per unit length), resistance mainly depends on wire length, whereas capacitance and crosstalk are heavily influenced by surrounding wires.

DLs have three interesting properties that simplify delay analysis: (1) the nets do not have glitches, (2) the time windows of the nets do not overlap, and (3) all nets have single fanout[4]. In this way, simple delay models can be used and crosstalk can be ignored by simply isolating or shielding the DL.

In order to simplify the analysis of interconnect delays, the following routing constraints for the DLs are defined:

- Only a small set of metal layers is used. This limits the range of resistivity coefficients and increases the correlation between delay and wire length, regardless the layers used during routing. In our experiments, only three layers were used.
- All the wires must have the same width.

[4]Property (3) is not fully complied when synthesizing configurable DLs with muxes (see section V).

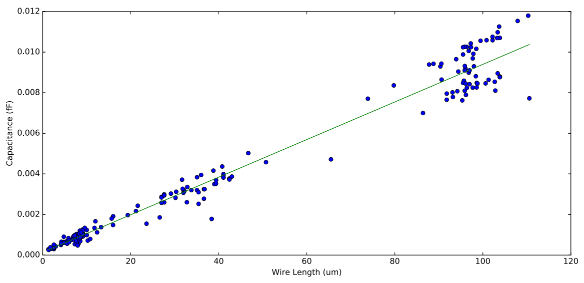

Fig. 6. Linear regression to estimate capacitance as a function of wire length.

- Large spacing rules between wires are defined. This dramatically reduces coupling capacitance.
- The DL must be isolated from the rest of the circuit, preventing crosstalk.
- The routing algorithm must minimize length. This is important for predicting wire length during placement.

With the previous constraints, wire delays mostly depends on wire length. Thus, simple delay models can be generated by randomly synthesizing DLs and learning a simple statistical prediction model. Fig. 6 shows a linear regression to estimate capacitance from a set of wires extracted from synthesized DLs, where each point represents a net. A high correlation between capacitance and wire length can be observed ($R^2 = 0.98$). A similar correlation is observed for wire delay predictions.

B. Implementation details

In the previous sections, it was assumed that the gate delay of a stage only depends on the input slew and output capacitance. In a real scenario, delay also depends on the transition direction (rising or falling). The previous algorithm can be easily extended to take into account the delays in both directions and select the most convenient.

Each combinational gate may also have multiple input pins and each one may be eligible for the connection with the previous stage. Each input pin and transition direction corresponds to a different timing arc in the gate with different characteristics in slew, capacitance and delay.

The search algorithm can be easily extended to explore any input pin of each combinational gate with both transitions, rising and falling. In fact, any library gate could be considered as a family of gates in which a different pin and transition is selected for the exploration.

The non-selected input pins must be connected to constant values in such a way that the selected input pin is sensitized (e.g., the remaining pins of a NAND gate must be connected to 1).

The DL is treated as a black box during physical design. Therefore, space for the DL must reserved a priori and used for placing its cells, as explained in Section IV.

Finally, the algorithm for DL synthesis assumes that the driver of the first gate and the output capacitance of the last gate are known in advance. For example, if the DL implements a delay monitor, there will be flip-flops at the input/output of the DL. In handshake circuits, there might be C-elements.

IV. Cell placement

The last step for the synthesis of DLs is physical synthesis (placement and routing). Routing is delegated to the existing routing tool in the design flow, but imposing the constraints described in Sect. III-A.

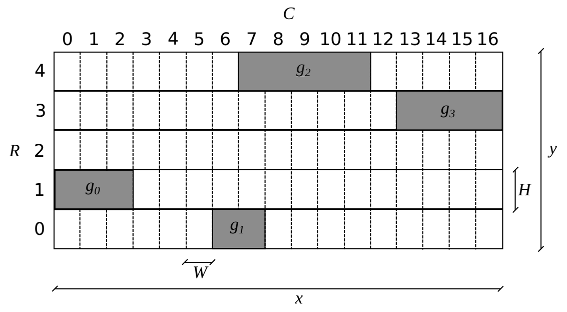

Fig. 7. Placement area for a delay line discretized into a grid.

This section proposes a SAT formulation for the placement step. The SAT formula is guided by the wire lengths of each stage selected during the synthesis step (see Algorithm 2).

Given the routing constraints defined for the wires, that push for the minimization of wire length, it is reasonable to assume that the nets will have a length close the half-perimeter of their bounding boxes. Therefore, the half-perimeter wire length (HPWL) model can be used as a good estimator.

The input of the placement formulation is a DL:

$$g_1 \xrightarrow{l_1} g_2 \xrightarrow{l_2} \cdots \xrightarrow{l_{i-1}} g_i \xrightarrow{l_i} g_{i+1} \cdots \xrightarrow{l_{n-1}} g_n \qquad (5)$$

where g_i represents the gate at stage i and l_i represents the required wire length from g_i to g_{i+1}.

The gates must be placed in an pre-defined area of the circuit. Fig. 7 depicts a placement area with width x and height y, divided in R rows and C columns. The height of each row is H and corresponds to the height of the standard cells. The width of each column is W and must be a multiple of the minimum routing granularity specified in the cell library. Hence,

$$R = y/H \qquad C = x/W.$$

Placement problem statement: Given a DL as defined in (5), place the gates $g_1 \dots g_n$ in a gridded area such that:

$$\forall i \in \{1, \dots, n-1\}: \quad |\textsc{Manh}(g_i, g_{i+1}) - l_i| < m \qquad (6)$$

where $\textsc{Manh}(g_i, g_{i+1})$ represents the Manhattan distance between g_i and g_{i+1}, and m is a tolerance factor between the actual distances and the required distances (ideally, m should be small).

Given that the number of gates is relatively small (few dozens at most), finding an optimal solution may be affordable. We first propose an iterative approximation based on the fact that a SAT formulation can be built for a given value m. The SAT formula is satisfied for all placement solutions for which (6) holds.

Main algorithm:

1) A small margin m is defined.
2) A SAT formulation is generated for m.
3) The formula is solved by a SAT solver.
4) If not satisfiable, increase m and go to 2)

The model that satisfies the SAT formula determines the location of each gate.

A. SAT formulation of the placement problem

We next define the set of variables and clauses of the SAT formula. We assume that each gate g occupies a set of adjacent slots in the grid. We call *size*(g) the number of slots occupied by g (for example, gate g_2 occupies 5 slots in Fig. 7).

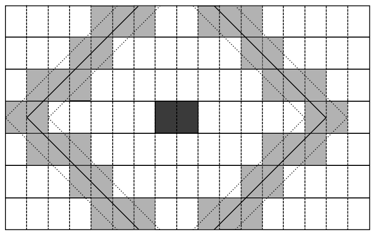

Fig. 8. Valid positions for a gate connected to the one in the middle, as represented by the shadowed boxes.

Variables: For every gate g, every row r and every column c, the variable $P^g_{r,c}$ indicates the fact that the leftmost slot of gate g is placed at the grid location (r, c).

Clauses: For simplicity in the representation, a number of definitions follow before describing the clauses.

- The function $\mathit{Overlap}(g, c)$ returns, for gate g and column c, the set of columns occupied by g if placed at column c. More specifically:

$$\mathit{Overlap}(g, c) = \{c' : c \leq c' < c + \mathit{size}(g)\}$$

- The function $\textsc{Manh}(r_1, c_1, r_2, c_2)$ returns the Manhattan distance between the grid cells (r_1, c_1) and (r_2, c_2).
- The predicate $\mathit{validDist}(l, r_1, c_1, r_2, c_2)$ is true when

$$|\textsc{Manh}(r_1, c_1, r_2, c_2) - l| < m$$

This predicate is useful to describe all the grid cells that are at a certain distance from another cell. As an example, the darkest cell in the center of Fig. 8 represents the location of a gate g_i. The shadowed halo around it represents the set of valid locations for gate g_{i+1} assuming that the required wire length is l_i. The width of the halo is determined by the tolerance factor m.

We next describe the set of clauses of the SAT formula:

- **Every gate must be placed:** A clause for each gate g with the disjunction of all the possible grid locations, ensuring that it is placed at least in one of them:

$$\forall g : \quad \bigvee_{r,c} P^g_{r,c}.$$

- **Every gate can only be placed in one location at most:**

$$\forall g, r_1, c_1, r_2, c_2 \; s.t. \; (r_1, c_1) \neq (r_2, c_2) : \quad P^g_{r_1,c_1} \Rightarrow \neg P^g_{r_2,c_2}$$

- **Gates cannot overlap:**

$$\forall g, g', r, c, c' \; s.t. \; g \neq g', c' \in \mathit{Overlap(g,c)} : P^g_{r,c} \Rightarrow \neg P^{g'}_{r,c'}$$

- **Valid distance for consecutive gates:** For any pair of consecutive gates, g_i and g_{i+1}, the Manhattan distance between them must be close to l_i (within the tolerance factor m), i.e.,

$$\forall g_i, g_{i+1}, r, c, r', c' \; s.t. \; \neg\mathit{validDist}(l_i, r, c, r', c') :$$
$$P^{g_i}_{r,c} \Rightarrow \neg P^{g_{i+1}}_{r',c'}$$

It is interesting to realize that all clauses have two literals except those that enforce every gate to be placed. The proliferation of 2-literal clauses implies that a lot of decisions are taken without branching (unit propagation). This aspect makes SAT solving more computationally efficient.

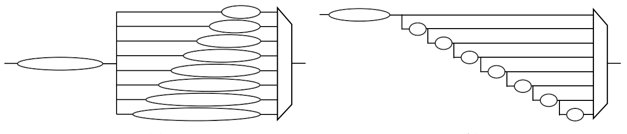

Fig. 9. Mux-based configurable RO architectures.

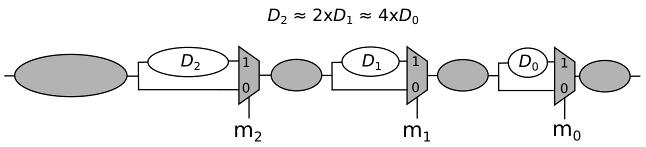

Fig. 10. Distribution of delays in a configurable DL with 3 muxes.

V. Configurable Delay Lines

Delay models are just *approximations* of the reality used during synthesis and verification. But reality is only known after manufacturing. Therefore, post-silicon calibration is essential to adjust DLs to the actual delays of the circuit.

Various techniques exist for calibration such as current starved inverters or voltage-controlled delay elements. In our work we propose all-digital solutions that use multiplexers (muxes) that can be found in the cell library. Calibration is performed by a set of *codewords* that control the muxes. It is desirable that the different configurable delays are uniformly distributed across codewords.

Fig. 9 depicts two possible schemes for configurable DLs. Each of them has a minimum delay shared by all possible configurations. The one in Fig. 9b is more area efficient but gives less flexibility in synthesizing the delay for each configuration. Another interesting and area-efficient solution commonly used for delay lines is shown in Fig. 10 (e.g., [4]). For N codewords, this scheme requires $M = \lceil \log_2 N \rceil$ 2-input muxes.

For the synthesis of configurable DLs, two new parameters are introduced:

- The number of codewords (N), usually a power of two.
- The configuration interval, $\mathit{CI} = (\mathit{CI}_{\min}, \mathit{CI}_{\max})$, that defines the range of configurable delays as coefficients over the target delay τ_c at each corner c. For example, $\mathit{CI} = (0.9, 1.1)$ indicates that N different delays must be configured in the interval $(0.9 \cdot \tau_c, 1.1 \cdot \tau_c)$.

In this paper we will focus on the scheme shown in Fig. 10 as it is the smallest of the three schemes. The synthesis for other schemes requires simple modifications with regard to this one.

The configuration step Δ of the DL is the expected delay difference between two adjacent codewords for a uniform delay distribution. Hence,

$$\Delta_c = \frac{\tau_c \cdot (\mathit{CI}_{\max} - \mathit{CI}_{\min})}{N - 1}, \qquad \text{for each } c \in \textsc{Corners}$$

and the delay D_i associated to each mux with control signal m_i is:

$$D_{i,c} = \Delta_c \cdot 2^i, \qquad \text{for } i \in \{0, \ldots, M-1\}, c \in \textsc{Corners}$$

The process of synthesizing a configurable DL is as follows:

- Synthesize a regular DL with target delay $\mathit{CI}_{\min} \cdot \tau_c$, for each corner c, in which M cells are enforced to be 2-input muxes. To mitigate the impact of slew propagation, it is also enforced that there are at least 5 gates between muxes (see the discusison about slew problem at the end of this section). This DL is represented by the shadowed components in Fig. 10. After this step, D_0, D_1 and D_2 are simply wires.

- The two inputs of each mux cell are connected to the output of the previous cell. One of the inputs will be selected to implement the delay D_i, whereas the other will remain intact.
- Implement each delay D_i as a DL using the same algorithm for a conventional DL. Insert the delay in front of one of the inputs of the mux.

The synthesis of configurable DLs requires small modifications of the SAT formulation of the placement problem that will not be discussed in the paper.

The slew problem. Using muxes introduces a new problem in the synthesis of DLs. The output slew of a mux depends on which input is selected. This effect is multiplicative, as the number of potential slew values at the output of a chain of muxes grows exponentially with the number of muxes.

This problem can be solved using the following property: for a sufficiently long path of gates, the output slew at the last gate is independent from the input slew at the first gate. Typically, and for reasonable slew values, a chain of 5 gates is sufficient to make the output slew virtually independent from the input slew [7].

The synthesis algorithm for configurable DLs guarantees that a minimum number of gates is inserted between two adjacent mux stages, as shown in Fig. 10. The delay of these gates is accounted within the minimum delay of the DL.

VI. Experimental Results for Ring Oscillators

DLs have multiple uses, including matched delays for bundled-data asynchronous circuits, canary paths or Ring Oscillators (ROs). This section will focus on using DLs to implement ROs, which implies some particular modifications on the algorithms previously described. A direct application of ROs is in the generation of Ring Oscillator Clocks (ROCs) [8].

An RO is a DL connected in a feedback loop. Few aspects must be considered for the synthesis of an RO:

- A new constraint for the DL algorithm is needed to ensure an odd number of inversions.
- The RO period consists of two oscillations, one for the rising and another for the falling transition. Thus, the period is the sum of the rising and falling delays at each stage.
- The output capacitance of the last cell is the input capacitance of the first cell. Similarly, the input slew of the first cell is the output slew of the last cell.

The experiments have been performed by synthesizing ROCs for several circuits. All the circuits have been implemented in a 65nm commercial library with 22 corners: 11 PVT corners $\times$ 2 interconnect corners (RC_{max} and RC_{min}). Timing results have been obtained by Synopsys PrimeTime [14].

The ITC99 benchmark suite [13] has been selected for the experiments. Circuits have been divided into two categories: small circuits (`b01-b13`), with size up to a thousand gates, and processors (`b14-b22`) with size up to a few hundred thousand gates [13].

The methodology for the experiments has been as follows:

- Layout synthesis has been performed using Synopsys EDA flow.
- PrimeTime has been used to calculate the target period (τ_c) at each corner.
- ROCs have been generated by running the synthesis algorithms for DLs presented in this paper.
- The reported results have been obtained after layout synthesis using PrimeTime.

TABLE III
Ring Oscillator delay mismatch (%), no muxes.

Circuit	Size	Max	Avg	Typ	Circuit	Size	Max	Avg	Typ
b01	5	2.70	1.13	1.01	b15	27	3.90	1.29	1.65
b02	5	2.23	1.11	1.09	b15_1	26	3.56	1.12	0.85
b03	5	4.50	1.86	0.75	b17	33	2.68	0.98	0.32
b04	20	0.98	0.45	0.40	b17_1	32	2.21	0.94	0.92
b05	12	1.37	0.66	0.70	b18	49	2.77	1.12	0.55
b06	5	2.00	1.13	1.51	b18_1	54	1.69	0.75	0.98
b07	8	1.71	0.97	0.64	b19	79	2.02	1.11	0.92
b08	9	1.22	0.79	0.78	b19_1	65	2.51	1.17	0.62
b09	6	1.86	1.08	0.97	b20	44	1.63	0.94	0.54
b10	6	2.38	1.31	1.81	b20_1	64	0.95	0.47	0.31
b11	13	2.69	1.34	0.88	b21	47	1.62	0.76	0.54
b12	13	2.61	0.99	0.86	b21_1	56	1.04	0.61	0.97
b13	8	1.86	1.27	1.27	b22	46	1.24	0.57	0.32
b14	41	1.60	0.66	0.66	b22_1	59	0.58	0.30	0.40
b14_1	49	1.95	0.74	0.39	**Aver**	**30.55**	**2.07**	**0.95**	**0.81**

The values reported at the tables and charts in this section correspond to the normalized mismatch (in percentage) of the ROC with regard to the target delay of the circuit at each corner (τ_c), as defined in equality (2). In the case of configurable ROCs, the mismatch has been calculated for each possible configuration of the delay.

Table III shows the results for ROCs without muxes. The column *Size* indicates the number of gates of the ROC. Column *Max* reports the maximum mismatch for all corners, whereas *Avg* reports the average mismatch across the 22 corners. *Typ* shows the mismatch at the typical PVT corner, bearing in mind that most dies will fall around this corner after manufacturing. The method guarantees that the mismatch is never negative.

The maximum mismatch is usually below 3% while the average mismatch is around 1% in most cases. This shows that a single DL can track circuit variability very accurately.

Fig. 11 gives more detailed information about the one shown in Fig. 2. It can be observed that, when restricting the set of gates used in the DLs, the capability of tracking variability is highly degraded. When only using one type of inverter, the average and maximum mistmatches can go up to 20% and 30%, respectively (see b09, b12 and b13). The inverter used in this experiment corresponds to the most used cell in all synthesized DLs. Even when using all inverters in the library, the mismatch is still substantially larger than when allowing all cells.

Table IV reports results for configurable ROCs with 1, 2 and 3 muxes (M), respectively. In this case, the maximum mismatch corresponds to the one achieved with any of the possible configurations. The average mismatch is the one over all configurations and corners. The mismatch at typical is the average over all the configurations at the typical PVT corner. Only circuits with DLs longer than 25 gates have been synthesized for this case. Small circuits are not appropriate for configurability given that the delay of a single gate is often longer than the minimum configuration step Δ. The configuration intervals used in the experiments were as follows:

	$CI_{\min}$	$CI_{\max}$
M=1	0.975	1.025
M=2	0.925	1.075
M=3	0.825	1.175

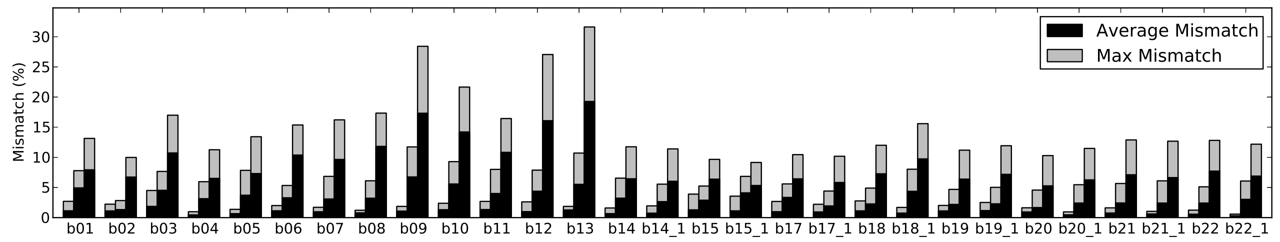

Fig. 11. Accuracy of DLs synthesized with any cell in the library (left bar), inverters of any size (middle bar) and inverters of one size (right bar).

TABLE IV
Ring Oscillator delay mismatch (%) with 1, 2 and 3 muxes.

	Max mismatch			Avg mismatch			Mismatch @typ		
Circuit	M=1	M=2	M=3	M=1	M=2	M=3	M=1	M=2	M=3
b14	2.06	2.46	3.44	1.04	1.19	2.03	1.00	1.14	1.99
b14_1	2.38	2.03	2.93	1.23	0.92	1.20	0.72	0.56	0.88
b15	3.27	4.82	6.30	1.56	2.50	3.38	1.19	2.02	3.37
b15_1	3.58	4.89	6.85	1.10	1.76	2.96	0.73	1.42	2.53
b17	2.46	4.32	4.94	0.88	2.31	1.97	0.40	1.92	1.78
b17_1	2.77	2.91	2.73	1.46	1.55	1.27	1.40	1.41	1.15
b18	3.73	3.05	4.55	1.81	1.22	1.80	1.42	0.73	1.20
b18_1	1.87	2.26	3.04	0.96	1.02	1.48	1.15	1.09	1.69
b19	2.90	3.65	3.93	1.54	2.25	2.18	1.01	1.80	1.48
b19_1	2.45	2.92	3.53	1.05	1.24	1.78	0.63	0.77	1.29
b20	1.39	1.73	2.04	0.75	0.85	1.08	0.44	0.50	0.77
b20_1	1.55	1.88	2.35	0.70	1.00	1.17	0.39	0.72	0.90
b21	2.14	3.31	3.04	0.96	1.51	1.47	0.81	1.29	1.33
b21_1	1.40	2.09	2.93	0.65	1.13	1.85	0.87	1.25	2.18
b22	2.01	2.54	3.05	1.09	1.49	1.92	0.78	1.04	1.55
b22_1	1.88	2.33	3.60	1.03	1.02	1.71	1.03	0.84	1.64
Aver	**2.36**	**2.97**	**3.70**	**1.11**	**1.44**	**1.83**	**0.87**	**1.16**	**1.61**

The results are reported in Table IV. As expected, the mismatch increases with the addition of muxes, since the requirement for introducing muxes reduces the flexibility to find gates that properly track the variability for all configurations. Still, the average mismatch is maintained around 1-2% in most cases, which is a remarkable achievement. This confirms the effectiveness of the synthesis algorithms to find very accurate mixtures of gates even with a large number of configurations.

As an example, Fig. 3 shows the DL generated for b05 according to the results shown in Table III. In this particular case, an ROC was constructed by connecting the input and the output of the DL.

VII. Conclusions

The synthesis of DLs for tracking variability is one of the emergent topics as technologies move towards nanometric dimensions. For a widespread use of DLs, it is necessary to provide design automation and schemes that can use the components of the cell libraries.

This paper has presented algorithmic techniques to tackle the synthesis of DLs, both at the logic and physical level. Using a variety of gates and wires in the same DL has proved to be essential for an accurate tracking of delays under the presence of variability.

We expect the incorporation of DLs, either playing the role of sensors or clock generators, to be a growing trend in the future. DLs can be used to monitor the potential fluctuations of delays at runtime and adapt the circuit to the varying operation conditions without requiring conservative guardband margins.

References

[1] Q. Liu and S. S. Sapatnekar, "Synthesizing a representative critical path for post-silicon delay prediction," in *Proc. International Symposium on Physical Design (ISPD)*, Apr. 2009, pp. 183–190.

[2] G. D. Carpenter, A. J. Drake, H. S. Deogun, M. S. Floyd, N. K. James, R. M. Senger *et al.*, "Circuit for dynamic circuit timing synthesis and monitoring of critical paths and environmental conditions of an integrated circuit," US Patent 7,576,569, Aug., 2009.

[3] L. Xie and A. Davoodi, "Representative path selection for post-silicon timing prediction under variability," in *Proc. ACM/IEEE Design Automation Conference*, 2010, pp. 386–391.

[4] A. Singhvi, M. T. Moreira, R. N. Tadros, N. L. V. Calazans, and P. A. Beerel, "A fine-grained, uniform, energy-efficient delay element for FD-SOI technologies," in *2015 IEEE Computer Society Annual Symposium on VLSI*, Jul. 2015, pp. 27–32.

[5] G. Heck, L. S. Heck, A. Singhvi, M. T. Moreira, P. A. Beerel, and N. L. V. Calazans, "Analysis and optimization of programmable delay elements for 2-phase bundled-data circuits." in *VLSI Design*, 2015, pp. 321–326.

[6] M. Bhushan, A. Gattiker, M. B. Ketchen, and K. K. Das, "Ring oscillators for CMOS process tuning and variability control," *IEEE Transactions on Semiconductor Manufacturing*, vol. 19, no. 1, pp. 10–18, Feb. 2006.

[7] T. B. Chan, P. Gupta, A. B. Kahng, and L. Lai, "DDRO: A novel performance monitoring methodology based on design-dependent ring oscillators," in *Thirteenth International Symposium on Quality Electronic Design (ISQED)*, Mar. 2012, pp. 633–640.

[8] J. Cortadella, L. Lavagno, P. López, M. Lupon, A. Moreno, A. Roca, and S. S. Sapatnekar, "Reactive clocks with variability-tracking jitter," in *Proc. International Conf. Computer Design (ICCD)*, Oct. 2015, pp. 540–547.

[9] M. Maymandi-Nejad and M. Sachdev, "A digitally programmable delay element: design and analysis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 5, pp. 871–878, Oct. 2003.

[10] W. Hua, R. N. Tadros, and P. Beerel, "2 ps resolution, fine-grained delay element in 28 nm FDSOI," *Electronics Letters*, vol. 51, no. 23, pp. 1848–1850, 2015.

[11] N. R. Mahapatra, S. V. Garimella, and A. Tareen, "An empirical and analytical comparison of delay elements and a new delay element design," in *IEEE Computer Society Workshop on VLSI, 2000. Proceedings*, 2000, pp. 81–86.

[12] J. Tschanz, K. Bowman, S. Walstra, M. Agostinelli, T. Karnik, and V. De, "Tunable replica circuits and adaptive voltage-frequency techniques for dynamic voltage, temperature, and aging variation tolerance," in *2009 Symposium on VLSI Circuits*, Jun. 2009, pp. 112–113.

[13] F. Corno, M. S. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *IEEE Design Test of Computers*, vol. 17, no. 3, pp. 44–53, Jul. 2000.

[14] Synopsys, "Synopsys PrimeTime," http://www.synopsys.com/Tools/Implementation/SignOff/Pages/PrimeTime.aspx.

[15] J. Cortadella, M. Lupon, A. Moreno, A. Roca, and S. S. Sapatnekar, "Ring oscillator clocks and margins," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 2016.

[16] R. Bisiani, "Beam search," in *Encyclopedia of Artificial Intelligence*, S. Shapiro, Ed. John Wiley & Sons, 1987, pp. 56–58.