Variable-Latency Design by Function Speculation

D. Bañeres
Universitat Oberta de Catalunya
Barcelona, Spain

J. Cortadella
Universitat Politècnica de Catalunya
Barcelona, Spain

M. Kishinevsky
Strategic CAD Lab, Intel Corp.
Hillsboro, OR USA

***Abstract*—Variable-latency designs may improve the performance of those circuits in which the worst-case delay paths are infrequently activated. *Telescopic units* emerged as a scheme to automatically synthesize variable-latency circuits. In this paper, a novel approach is proposed that brings three main contributions with regard to the methods used for telescopic units: first, no multi-cycle timing analysis is required to ensure the correctness of the circuit; second, the method can be applied to large circuits; third, the circuit can be optimized for the most frequent input patterns. The approach is based on finding approximations of critical nodes in the netlist that substitute the exact behavior. Two cycles are required when the approximations are not correct. These approximations can be obtained by the simulation of traces applied to the circuit.**

Experimental results on selected examples show a tangible speed-up (15%) with a small area overhead (3%).

I. Introduction

The performance optimization of combinational circuits is usually accomplished by reducing the delay of the critical paths after an accurate timing analysis. This reduction is achieved by applying different transformations on the circuit, such as logic restructuring or gate sizing, that usually result in area and power overheads.

There is an interesting property that can be exploited in many designs: *the critical paths are infrequently activated*. Instead of defining the cycle time by the worst-case delay, a shorter cycle time that covers a significant amount of input stimuli can be chosen. The worst-case operations may not be accommodated in this clock period and, therefore, more cycles may be required to complete them. These *variable-latency* circuits are commonly used in long data-paths, such as arithmetic circuits [1], to improve the performance.

Telescopic units [2]–[4] is a paradigm to automatically build variable-latency circuits. An *error detection function*, referred as *hold* function in [3], is computed to inform the environment at which cycle the correct result is available at the outputs. This function externally controls the clock period by holding the values on the registers [5] or by adapting the clock frequency [6]. The variable-latency circuit can be also used in an elastic design [7].

A combinational block is constructed to compute the error detection function that identifies those input patterns that require more than one cycle to complete the execution. The function is not always exact. A high computational cost is usually required to synthesize the exact function that covers all these input patterns. The complexity is equivalent to solving the *false path* problem, which is NP-complete [8]. The calculation of the error detection function requires an individual analysis of each input pattern. The proposed methods usually resort to symbolic methods (e.g. BDDs) that simultaneously analyze all input patterns. However, these strategies are often limited to small or medium-size circuits.

Finally, the synthesis of telescopic units in conventional design flows requires the definition of multi-cycle constraints that complicates the design and validation flows. Moreover, they are not easy to represent, since they are exercised by a complex set of input patterns.

This paper proposes an alternative and practical method to synthesize variable-latency units. The critical paths are substituted by non-critical signals that approximate their functionality. The error detection function checks the correctness of each approximation with regard the substituted signal. If the error detection function is not activated, the cycle time is reduced due to the utilization of faster signals in the original critical paths. If the error detection function is activated, the exact value of the substituted signal is supplied in the next cycle to amend the error.

A similar technique has been previously applied to some arithmetic circuits [9]–[11]. An adder is a circuit with a long critical path (the carry signal) that can be easily approximated with near-zero effect on the correctness of the result. In this paper, we generalize the technique for the automatic synthesis of any circuit.

The rest of the paper is organized as follows. Section II gives an overview on telescopic units and presents the main contributions of this paper. Section III introduces the basic terminology that will be used in the paper. The details of the variable-latency scheme are explained in Section IV, the technique is discussed in Section V and the algorithm to optimize the cycle time is presented in Section VI. Finally, Section VII explains how a variable-latency circuit is built and Section VIII reports the experimental results.

II. Overview and Contributions

This section introduces the basics of the scheme for the design of variable-latency units and the main differences with the existing approaches for telescopic units.

Figure 1(a) shows an example of the computation of the error detection function (F_{err}) for a telescopic unit. The example implements a 6-bit ripple carry adder in which each box represents a full adder. Assuming that the delay of each full adder is 1 unit, the critical path is 6 units. There are theoretical studies [11] that demonstrate delays larger than 4 units are rarely activated. A possible error detection function for this cycle time is $F_{err} = (A_4 \oplus B_4)(A_3 \oplus B_3)$, which describes the condition when the carry c_2 is 1 and the 3^{th} and 4^{th} full adders propagate a carry. If the error detection function is activated, the sum operation requires two cycles. Note that this speculation function is not exact since it does not consider the carry propagation at the least-significant bits of the addition. Assuming a uniform distribution of the inputs, the probability of F_{err} is 0.25, while the probability of an exact error function would be 0.1875.

The scheme proposed in this paper is based on the speculation of some values that approximate signals in the critical paths. Two examples of the computation of the error detection function by speculation are presented in Fig. 1(b)-(c). Assume that the carry c_2 is selected as a speculation point. The objective is to find a simple function that approximates the behavior of the carry signal c_2 with high probability. Figure 1(b) shows the selection of the constant zero. In terms of area and delay, it is a good function because there is no overhead, but the carry c_2 is 1 with probability 0.375. Figure 1(c) shows another approximation. The selected function is $A_2 \cdot B_2$ which decreases the probability of error down to 0.125. The error detection function will detect the errors by comparing the approximation with the exact function.

978-3-9810801-5-5/DATE09 © 2009 EDAA

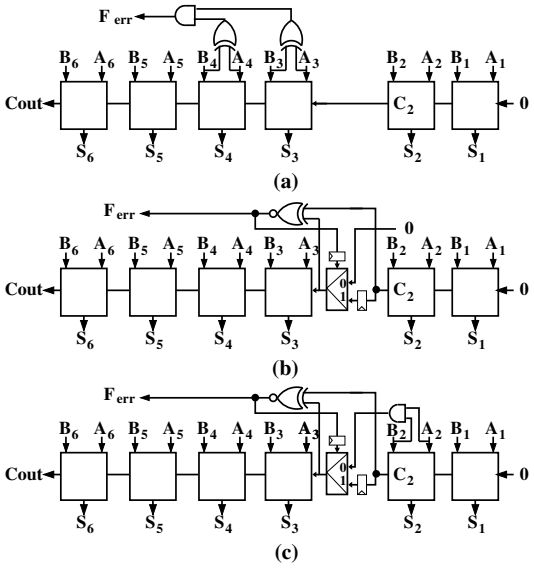

Fig. 1. The example of computing the error detection function for a ripple carry adder (a) by using [3] and (b),(c) by function approximation.

In Fig. 1(b)-(c), extra logic is also depicted. This logic, explained in Section IV, supplies the correct value of the carry c_2 when the error detection function is activated. The new design may operate in one or two cycles, but no combinational path is longer than one cycle. Thus, no multi-cycle timing analysis is required.

The contributions of the presented approach are next summarized:

- The speculation techniques for arithmetic circuits [10], [11] are generalized and a method for automatic synthesis is proposed.
- The approach presented in this paper does not require multi-cycle timing analysis. The original circuit is transformed into a sequential circuit with two cycles. In telescopic units, it is difficult to use conventional EDA flows because very intricate multi-cycle constraints must be applied.
- The approach based on speculation is specially suitable for large circuits. The choice of speculation points based on simulation makes the approach scalable. The use of real traces may lead to an optimization oriented to the most frequent input patterns.
- There is a negligible overhead in area on the variable-latency designs because the approximation functions are extracted from the internal logic. On average, a performance gain of 15.1% and an area overhead of 3.4% is reported in the experimental results.

III. BACKGROUND

This section introduces some definitions required for the paper.

Each gate g in a netlist has an *arrival time* ($AT(g)$) based on the worst-case delay needed to obtain the correct value. The *required time* ($RT(g)$) is defined as the time when the correct value is expected at the output of the gate. A gate is *critical* if its *slack* ($SL(g)$), computed as $RT(g) - AT(g)$, is negative. A *critical path* is a path from primary inputs to primary outputs where all the gates are critical. The *maximum arrival time* for a netlist is computed as the greatest arrival time of the outputs. Assuming that a sequential circuit has many combinational blocks between the registers, the *cycle time* is defined as the maximum arrival time of all the combinational blocks.

The performance of the circuit is analyzed with the *effective cycle time* (or *effective delay*). Assuming that the error detection signal F_{err} is activated with a probability P_{err} and the cycle time of the variable-latency design is C_v, the effective delay is computed as follows:

$$D_{eff} = C_v(1 + P_{err})$$

In the presented method, the set of gates selected to be approximated are called *speculation points*. On each point, the function of the gate and the chosen approximation are referred as *exact* and *approximation*, respectively. The candidates for the approximation of a speculation point are defined as *potential candidates*.

In this approach, the candidates to substitute a speculation point are the constants 0 and 1, and gates selected from the same netlist. Note that, new functions not included in the design [12] can be also used. However, we observed that the area considerably increases without an important improvement on delay.

Given two functions $F(X)$ and $G(X)$ where $X = (x_1, \ldots, x_n)$ and a set of k input assignments $\{\mathbb{X}_1, \ldots, \mathbb{X}_k\}$, the *equivalence probability* between F and G is defined as

$$P_{eq}(F,G)_{\{\mathbb{X}_1,\ldots,\mathbb{X}_k\}} = \sum_{\forall \mathbb{X}_i \in \{\mathbb{X}_1,\ldots,\mathbb{X}_k\}} p(\mathbb{X}_i) \cdot (F(\mathbb{X}_i) = G(\mathbb{X}_i))$$

where $p(\mathbb{X}_i)$ is the probability of appearance of the input assignment $\mathbb{X}_i$ and the second term is one when both functions have equal value and zero, otherwise.

The equivalence probability models the similarity between two functions. This probability can be estimated in many ways. In this paper, simulation with randomly generated traces has been chosen. This strategy makes the method scalable for large circuits. In the particular case that all input patterns are generated and they are equiprobable, the exact equivalence probability can be calculated as follows:

$$P_{eq}(F,G)_{\mathbb{B}^n} = \frac{1}{2^n} \sum_{\forall \mathbb{X}_i \in \mathbb{B}^n} (F(\mathbb{X}_i) = G(\mathbb{X}_i))$$

However, most circuits may have input don't care conditions determined by the environment or the sequential behavior of the circuit and the input vectors may not be equiprobable either. In this cases, the use of real traces may result in an estimation of the equivalence probability based on the real behavior of the circuit.

The *global equivalence probability* P_{geq} computes the ratio of correctness on the set of speculation points. This probability can be naturally extended from the previous definition of equivalence probability assuming that the functions F and G are multi-output functions and each output corresponds to a speculation point. This probability is also used to check the equivalence between a netlist and its respective variable-latency circuit assuming that each output corresponds to a primary output. Note that the probability of the error detection function is the complement of the global equivalence probability ($P_{err} = 1 - P_{geq}$).

IV. VARIABLE-LATENCY DESIGN

In this section, we discuss the structure of the variable-latency units. As an example let us consider how to transform a piece of a combinational logic into a variable-latency design with one speculation point.

Figure 2(a) shows an example of a combinational netlist. Every node in the graph corresponds to a combinational gate. Assume that we want to construct a variable-latency design. Let select gate k, that belongs to the critical paths, as a speculation point, and gate d as its approximation. Figure 2(b) depicts the resulting netlist. The new design has five new components:

- a primary output F_{err} that indicates whether there is an error in the approximation;
- a multiplexor that controls the output of the speculated function and selects between the exact function (gate k) and its approximation (gate d);

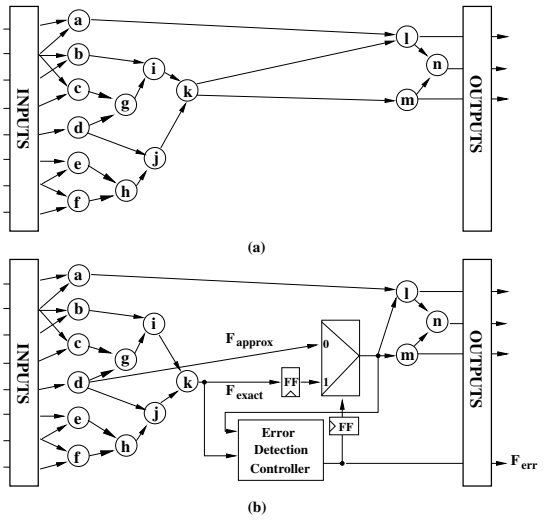

Fig. 2. Transformation of a circuit to a variable-latency design.

- an error detection controller that checks if the exact and the approximation function produce the same value;
- a flip-flop that stores the value of the error detection signal F_{err} for the next cycle for the controlling signal of the multiplexor;
- a flip-flop that stores the output of the exact function F_{exact}.

Note that the original combinational circuit is transformed to a sequential circuit with two cycles. Most of the time only one cycle pass is exercised (from gate d via the multiplexer to the primary outputs). In rare cases a two cycle sequential path is exercised. Unlike the telescopic units approach, the resulting circuit has no multi-cycle combinational paths and hence does not require multi-cycle analysis.

For some input patterns the approximate function F_{approx} may produce some erroneous values. These errors can be detected by comparing the result of the approximate function F_{approx} with the exact one F_{exact}. The comparison is performed by the *error detection controller* which checks whether the output of both functions is identical. In case of an error, the error detection signal F_{err} is emitted and, at the beginning of the second cycle, this signal forces the multiplexor to select the exact output F_{exact} with a correct value. In presence of an error the correct output value is produced at the end of the second cycle.

In the example of Fig. 2(b), the effective delay minimization is obtained using only one speculation point that cuts all the critical paths. In general, a set of speculation cut points may be needed to cut all critical paths. Therefore, a variable-latency design is constructed searching for a set of speculation points that cut all the critical paths. The details on how to find the best set and the best approximation for each point are explained in the following section.

Figure 3 shows the error detection controller for a variable-latency circuit with n speculation points. The controller checks whether an error is triggered at any of the speculation points. An error activates the error detection signal that selects the exact value in all the multiplexors.

V. Speculation Points

This section describes the method to find the set of speculation points. Given a targeted cycle time C_v for the variable-latency unit, the presented method consists of:

- Identifying the critical nodes of the netlist using conventional static timing analysis.
- Finding a list of *fast* approximation functions for each critical node. These functions, called *approximation candidates*, can be either constants or other nodes from the same netlist.

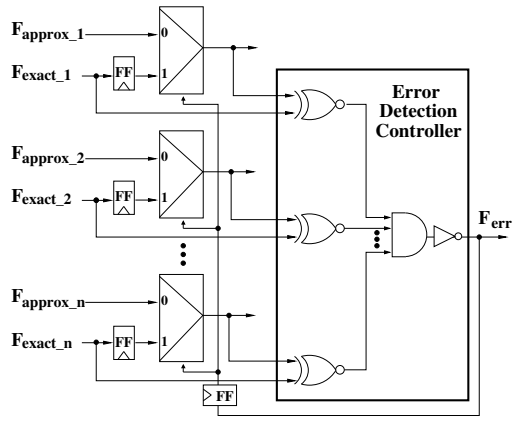

Fig. 3. Description of the error detection controller.

- Finding a set of critical nodes, called *speculation points* that *cut* all critical paths and their substitution by the approximations meets the target cycle time C_v.

Since the goal of the algorithm is to minimize the effective delay, the approximation functions should be similar to the functions of the speculation points to increase the probability of producing a valid output in one cycle.

A. Approximation candidates

This section describes how the approximation candidates are selected.

The similarity relationship between gates is computed using the equivalence probability. Given two gates f and g with their respective Boolean functions F and G, the *similarity* between these gates is defined as $Sim(f,g) = max(P_{eq}(F,G), 1-P_{eq}(F,G))$

where Sim is a number in the interval $[0.5,1]$. The fact that $Sim(f,g) \approx 1$ indicates that f and g are similar, either with positive or negative polarity. If $Sim(f,g) = P_{eq}(F,G)$, then G is a better approximation, otherwise $\overline{G}$ is better.

A large set of candidates is generated for each gate. Only the gates not too close to critical primary inputs or outputs are explored. As we describe in the next section, replacing the gates close to inputs or outputs would not improve the delay.

A Pareto curve, commonly used in multi-criteria optimization, is used to only store the best candidates. Given the approximation candidate g for the gate f identified with the pair $C_{(f,g)} = (Sim(f,g), AT(g))$, the candidate is *Pareto optimal* if and only if there is no candidate $C_{(f,h)}$ such that $Sim(f,h) \geq Sim(f,g)$ and $AT(h) \leq AT(g)$, with at least one strict inequality. I.e., we identify candidates with the smallest arrival time and the highest similarity.

Figure 4 shows a gate window in the middle of a circuit that we use as an example for illustrating a candidate selection for gate i. Out of all gates in this window, $\{a,b,c,d,e,f,g,h\}$, only 5 are stored as Pareto points. Given gate f with a set of approximation candidates $\{g_1,\ldots,g_n\}$, a *potential candidate* for the targeted cycle time C_v is a gate $g_i \in \{g_1,\ldots,g_n\}$ such that

$$SL(f) > AT(g_i) - AT(f) + \delta(MUX)$$

where $SL(f)$ is the slack of gate f with regard to the targeted cycle time C_v and $\delta(MUX)$ is the extra delay introduced by the multiplexor.

B. Cut of speculation points

This section describes how the cut of speculation points is selected. A netlist is a directed acyclic graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents

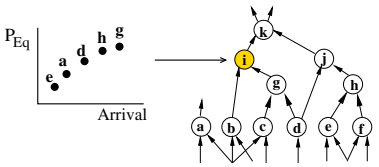

Fig. 4. Approximation candidates for gate *i*.

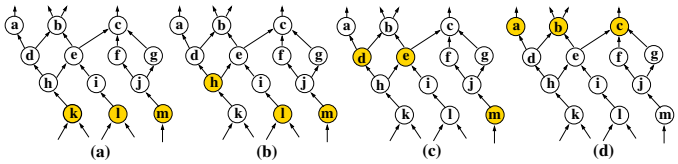

Fig. 5. Forward exploration. (a) Starting point. Exploration of (b) gate *h*, (c) gates *d* and *e*, and . (d) gates *a*, *b* and *c*.

Algorithm 1 Speculation point exploration

Input: The netlist N, the required cycle time C_v
Output: Cut of speculation points Cut_{sp}
1: $Cut_{sp} \Leftarrow$ Obtain initial cut in netlist N with cycle time C_v
{Refine cut of speculation points}
2: **repeat**
3: Speculation point: $Worst_{sp} \Leftarrow$ Select gate Cut_{sp} with smaller similarity
4: Cut of spec. points: $NewCut_{fwd} \Leftarrow$ Explore forward from $Worst_{sp}$
5: Cut of spec. points: $NewCut_{bwd} \Leftarrow$ Explore backward form $Worst_{sp}$
6: $Cut_{sp} \Leftarrow$ Select cut with higher global equivalence probability (Cut_{sp}, $NewCut_{fwd}$, $NewCut_{bwd}$)
7: **until** there are no more changes in Cut_{sp} and all speculation points in Cut_{sp} have been processed
8: **return** Cut_{sp}

the gates of the netlist and $\mathcal{E}$ represents the set of wires. A *critical cut* in the graph G is a subset of nodes $Cut = \{f_1, \cdots, f_n\} \subset \mathcal{V}$, such that every critical path from primary inputs to primary outputs contains some node $f_i \in Cut$.

Given a critical cut $Cut = \{f_1, \ldots, f_n\}$, the cut is a *minimal critical cut* if for each node $f_i \in Cut$, there is at least one path towards the primary outputs that does not contain a node $f_j \in Cut \setminus \{f_i\}$.

Not all minimal critical cuts are a valid cut of speculation points. Let C_f be the cycle time of the original fix latency circuit and C_v be the targeted clock cycle for the new variable latency circuit. Given a minimal critical cut $Cut = \{f_1, \ldots, f_n\}$ with their respective approximation gates $App_{Cut} = \{g_1, \ldots, g_n\}$ selected from their set of potential candidates, the cut is a *cut of speculation points* $Cut_{sp} = \{(f_1, g_1), \ldots, (f_n, g_n)\}$ if the next properties are satisfied:

- $\frac{C_f}{C_v} > 1 + P_{err}$
- $AT(F_{err}) = max_{\forall f_i \in Cut}(AT(f_i)) + \delta(F_{err\_comp}) < C_v$.
- $SL(f_i) > AT(g_i) - AT(f_i) + \delta(MUX)$.

The first property, derived from the definition of efficient cycle time guarantees that the new design has better effective delay. The remaining properties ensure that the cut satisfies required timing constraints. The second property checks that the error computation fits into the required cycle time C_v. To estimate the delay of the error detection function we use the following upper bound from representing this function as a binary tree of 2-input AND gates: $\delta(F_{err\_comp}) = \delta(MUX) + \delta(XNOR) + \log_2(\#Cut + 1) \cdot \delta(AND) + \delta(NOT)$. The last property ensures that the approximate computation fits into C_v.

The Algorithm 1 explains our method. The algorithm starts from an initial cut of speculation points and refines the cut until a solution with the smallest approximation error is found.

A greedy selection is performed to produce the initial cut (Line 1). Given a critical path $\pi = \{f_1, \ldots, f_n\}$ with the respective selected approximation candidates $App_\pi = \{g_1, \ldots, g_n\}$ such that $Sim(f_i, g_i) = max_{\forall g_j \in \{pot\_candidates\_f_i\}}(Sim(f_i, g_j))$, the gate $f_i \in \pi$ selected to cut the critical path has the best approximation candidate, i.e. $Sim(f_i, g_i) = max_{\forall j \in \{1,\ldots,n\}}(Sim(f_j, g_j))$. This process is performed for all the critical gates until a valid cut of speculation points is found. This process is done by ordering the critical gates by the approximation function with the best similarity that fits the required cycle time.

This initial selection often produces a large set of points with an irregular distribution across the netlist. The refinement step (Lines 2-7) tries to optimize the global equivalence probability by searching for other cuts within transitive fanin and fanout of the previously selected approximation

The refinement starts by selecting the point with small similarity (Line 3). Two types of traversal are used: forward (Line 4) from the gate towards the primary outputs, and backward (Line 5) from the gate towards the primary inputs. Then, the algorithm checks the quality of the new cuts comparing their respective global equivalence probability (Line 6). The cut with the maximal global equivalence probability is selected as the new cut. The algorithm iterates with the next worst speculation point. The algorithm stops when reaches a fixed point: all the speculation points have been processed and there are no more changes in the cut.

Let us consider in more details the forward exploration. The backward one is similar. The exploration starts with the immediate fanout nodes of the previously selected speculation point. Note that only the critical gates (gates with negative slack) need to be traversed until a cut with a higher global equivalence probability is found. This exploration produces a better distribution of the speculation points since the new selected gates may dominate other points in the cut. This condition allows to remove the dominated ones because they are no longer needed to meet the cycle time. A cut with a smaller number of points tends to have a higher global equivalence probability.

Figure 5 shows an example of a forward exploration. Assume that the graph shown is the critical region of a netlist. The initial cut is $\{k,l,m\}$ and the worst speculation point is gate k. The exploration starts with its immediate fanout h. The second cut is $\{h,l,m\}$. The next cuts are $\{d,e,m\}$ and $\{a,b,c\}$. The algorithm will select the cut with maximal global equivalence probability.

For the telescopic units approach the complexity depends on the size of the error detection function and the Long/Short Path Activation Functions [4]. In contrast, complexity of our algorithm depends on two different factors: (1) the size of the critical region of the circuit, and (2) the selected precision of the simulation. The computation of any equivalence probability directly depends on the number of selected input patterns.

C. Error-aware selection of the approximation for a speculation point

This section defines how the best approximation for a speculation point is selected from the set of potential candidates. Instead of maximizing the individual probability at each speculation point, our selection of the best potential candidate during the refinement step is based on maximizing the global equivalence probability of the cut.

The reason of this selection is because the gates of the cut are not independent. They belong to the same netlist and there are "unknown" correlations among them. Therefore, we minimize the global error and not the local error produced at each point. This selection contributes to increment the sharing of errors among the speculation points.

Algorithm 2 Optimize efficient cycle time

Input: The netlist N, the cycle time *Cycle_min*, the cycle time *Cycle_max*
Output: Cut of speculation points *Best_Cut*
1: Required cycle time: $Cycle\_obj \Leftarrow (Cycle\_min + Cycle\_max)/2$
2: **if** $Cycle\_obj$ not yet explored **then**
3: Cut of spec. points: $Cut \Leftarrow$ **call** Speculation point exploration(N,$Cycle\_obj$)
4: Performance gain: $Cycle\_eff \Leftarrow$ Efficient delay of Cut
5: **if** Efficient delay improved **then**
6: $Best\_Cut \Leftarrow Cut$
7: **call** Optimize efficient cycle time(N, $Cycle\_min$, $Cycle\_obj$, $Best\_Cut$)
8: **else**
9: **call** Optimize efficient cycle time(N, $Cycle\_obj$, $Cycle\_max$, $Best\_Cut$)
10: **end if**
11: **end if**

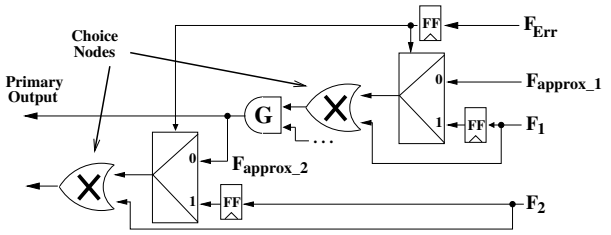

Fig. 6. Example of choice nodes.

VI. Cycle time optimization

The construction of a variable-latency design depends on the objective cycle time. However, the required cycle time may not be specified. Algorithm 2 shows the optimization procedure.

The algorithm explores several cycle times and selects the one with a cut that produces the best effective delay. A binary search is used to explore the minimum number of cycle times. This type of exploration can be performed, because, we observed there is a trade-off between the reduction of the cycle time and the increment of the probability P_{err} of the error detection function. Therefore, there is a cycle time where there is no more performance gain because the increment of P_{err} does not compensate the decrement in the cycle time.

Let C_f be the cycle time of the original fix latency circuit. The algorithm starts assigning *cycle_max* and *cycle_min* to the cycle time C_f and $C_f/2$, respectively. Only the interval $[C_f/2, C_f]$ is explored. The original netlist is transformed to a sequential circuit with two cycles. Therefore, the maximum improvement would be $C_f/2$. On each step of the binary search, the required cycle time $Cycle\_obj$ is initially computed (Line 1). The *speculation point exploration* procedure, defined on Algorithm 1, is applied (Line 3) to obtain the cut of speculation points for $Cycle\_obj$. The next step of the binary search depends whether there is a performance gain with regard the previous cut. If there is improvement the interval $[Cycle\_min, Cycle\_obj]$ is explored (Line 7), otherwise $[Cycle\_obj, Cycle\_max]$ is processed (Line 9).

VII. Construction of the variable-latency netlist

One of the contributions of the proposed approach is the negligible overhead in area. This is achieved by using the gates of the netlist as approximation functions and reducing the number of duplicated gates. This section describes how the construction technique detects the minimal number of gates that must be duplicated to built a functional variable-latency netlist.

The problem has been modeled with choices nodes similar to the technology mapping approach proposed in [13]. A choice node is an element which collects different options that can be selected on a wire. Figure 6 shows an example. Assume a variable-latency design with two speculation points. Notice that the error detection controller has not been depicted. The choice nodes represented with a cross are inserted on each speculation point. One option is the logic needed for the variable-latency netlist described in Section IV. The second one is the case when the exact function is needed. The figure shows there is a primary output that needs a speculation point to reach the cycle time. There is also another critical path in the netlist that needs the exact gate F_1 on its approximation function. Here, the primary output and the other critical path need the first and the second option of the choice node respectively. In this case, our approach only duplicates the gate G and connects the option selected in the choice node to the proper duplicated gate.

The choices nodes are explored in two steps. The objective is to identify the gates that need duplication. A gate is duplicated if there is a gate in the transitive fanout that requires the exact output of the function of the gate and there is a speculation point in the transitive fanin. First, the netlist is traversed backward from primary outputs to primary inputs in DFS order to mark the gates which their exact function is needed. Then, the netlist is processed in reverse order to mark the gates that have speculation points in the transitive fanin. The variable-latency netlist is built based on the decisions performed on each choice node and whether the condition of duplication is fulfilled. The fanins of each gate in the netlist are connected to the proper duplicated gate based on these decisions.

VIII. Experimental results

Our method of synthesis of variable-latency units has been implemented as an extension to SIS [14].

In this section, we describe the experimental results obtained by our tool on the largest examples from the ISCAS99 and MCNC89 benchmarks. The results are summarized in Table I: the left part of the table presents results after the technology-independent synthesis, while the right side - results after the technology mapping. Columns labelled with `Fix` and `Var` report synthesis results for the original fixed latency examples and for the variable-latency netlists obtained by our tool, correspondingly.

The technology-independent synthesis for the original netlists was done by applying the *algebraic* script followed up by the *speed_up* [15] script for delay optimization.

The synthesis for the variable-latency proceeded as follows. As a preprocessing step, the *algebraic* script has been applied to the original netlist. Then the netlist has been decomposed into 2-input gates to enable measuring the delay under the *unit delay model*. Then our method of variable-latency synthesis was applied, followed by the *speed_up*.

As no realistic testbenches are available for these examples our algorithm computed variable-latency units under assumption of random testbenches with equally distributed inputs.

For technology mapping we used the tree-mapping algorithm *map -AFG* with the gate library *lib2.genlib*.

The delay optimization approach has been applied to 30 largest examples from ISCAS and MCNC. Our algorithm has found variable-latency designs that improve the delay cost function for 13 netlists. For other examples no improvement has been found. Our current algorithm computes approximation functions using only existing internal gates of the netlist. We suspect that removing this limitation will allow to find variable-latency designs for other examples as well (planned as future work). The examples with no improvement are not shown in the result table.

As Table I shows, the variable-latency units obtained by our approach improves the technology-independent delay measured by the number of two input gate levels by 24% at the cost of 3% in the number of nodes and a few extra flip-flops (`FF`) required for cutting

Netlist	Technology Independent					Technology Dependent						CPU (sec)		
	Nodes		Levels		FF	Area		Delay			P_{err}	Sim +	Cuts	Total
	Fix	Var	Fix	Var		Fix	Var	Fix	Var	D_{Eff}	(%)	Cand		
apex5	719	741	24	15	5	907	905	19.59	16.18	17.12	5.79	1	2	3
table3	795	966	47	31	25	1160	1445	41.31	29.59	33.27	12.43	5	12	17
dalu	1092	1098	41	27	2	1409	1491	32.81	24.04	24.31	1.14	3	3	6
C7552	1889	1901	37	28	2	2454	2463	30.53	25.90	26.20	1.14	10	15	25
s13207	2305	2362	28	21	9	2890	2964	25.82	22.56	22.61	0.21	11	23	34
s15850	3292	3470	37	26	11	3942	4286	37.29	28.45	29.08	2.21	14	21	35
b14_1	4820	5010	67	59	4	6109	6444	53.71	46.39	48.91	5.43	32	54	86
b14	5682	5722	77	59	3	6932	7054	72.60	54.68	61.67	12.79	45	103	148
b15_1	8776	8847	63	42	10	10334	10536	61.60	42.14	42.71	1.36	73	183	256
b15	9132	9193	84	57	7	10896	11167	73.39	54.81	61.90	12.93	109	377	486
b20_1	10066	10593	92	80	5	12404	12982	83.06	73.07	77.04	5.43	129	454	583
b21_1	10078	10716	86	80	6	12540	13275	82.34	73.54	77.53	5.43	123	398	521
s38584	10576	10644	27	21	10	12690	12551	24.89	20.11	20.12	0.07	90	130	220
Norm.	1.000	1.029	1.000	0.769		1.000	1.034	1.000	0.800	0.849	5.10			

TABLE I
EXPERIMENTAL RESULTS ON ISCAS AND MCNC BENCHMARKS.

the speculation points and for the error detection controller. After technology mapping the circuit delay is improved by 20% at the cost of 3.4% area increase. The effective delay (D_{eff}) is improved by 16%. It takes into account that in cases of misspeculation the circuit needs to spend an extra cycle to compute the result.

P_{err} reports the error detection function probability (i.e. the probability of misspeculation). The optimization algorithm is able to find designs with a low probability of errors. However, the probability varies from one example to the other. E.g. for `s38584` the probability of the error is 0.07%, while for `b15` it is almost 13%. In both cases our method can find the circuits with a significant improvement of the average delay (see D_{eff}). Moreover, for some large netlists, e.g. `b14`, `b14_1`, the algorithm finds the variable-latency design with just a few speculation points. Also, the area can slightly decrease for some netlists (`apex5`, `s38584`). The runtime of the tool shows that our technique can manage 10K gates netlists within a few minutes. The runtime is also broken down into the time spent in simulation and the selection of candidates, and the time to search the best cut of speculation points.

Our method is targeting large netlists with reasonably long critical paths which can be cut by inserting the speculation points. When applied to small circuits of the MCNC benchmarks that have been used for the experiments with the telescopic units in [3], [4], we did not observe a lot of improvement as compared to the original small netlists. These circuits, such as `cc`, have 5-8 levels of logic. The algorithm did not find good approximation of the cut point nodes since there are too few candidates to choose from and even if there are good candidates the benefit of approximation is not significant since the delay at cut point is already close to minimal. On the other hand, the telescopic units methods of [3], [4] cannot handle large circuits, since the approach explodes due to the large number of inputs patterns that should be considered for the error detection function.

Method from [4] reports that the telescopic units for small circuits obtains a performance gain of 25% at the cost of large area overhead ($\sim$ 20%). In contrast, we obtained 16% delay improvement at the cost of 3.4% area increase for large circuits (after technology mapping).

IX. CONCLUSIONS AND FUTURE WORK

In this paper, a new technique to construct a variable-latency design has been proposed. We use a generalization of the techniques proposed in [10], [11] where approximation functions are used to emulates the behavior of the circuit.

As future work, we will study the exploration of new function as approximation function not included in the original netlist. New functions will increase the ratio of success to create a variable-latency design. However, the area will be negatively affected. Moreover, we will also analyze the impact on the peformance of this optimization on an entire system and the utilization of variable-latency design on asynchronous circuits.

REFERENCES

[1] S. F. Oberman and M. J. Flynn, "Design issues in division and other floating-point operations," *IEEE Transactions on Computers*, vol. 46, no. 2, pp. 154–161, 1997.

[2] L. Benini, E. Macii, M. Poncino, and G. De Micheli, "Telescopic units: a new paradigm for performance optimization of vlsi designs," *IEEE Transactions on Computer-Aided Design*, vol. 17, no. 3, pp. 220–232, Mar 1998.

[3] L. Benini, G. D. Micheli, A. Lioy, E. Macii, G. Odasso, and M. Poncino, "Automatic synthesis of large telescopic units based on near-minimum timed supersetting," *IEEE Transactions on Computers*, vol. 48, no. 8, pp. 769–779, 1999.

[4] Y.-S. Su, D.-C. Wang, S.-C. Chang, and M. Marek-Sadowska, "An efficient mechanism for performance optimization of variable-latency designs," *Proc. ACM/IEEE Design Automation Conference*, pp. 976–981, 2007.

[5] Y. Chen, H. Li, J. Li, and C.-K. Koh, "Variable-latency adder (vl-adder): new arithmetic circuit design practice to overcome NBTI," in *International Symposium on Low Power Electronics and Design*, 2007, pp. 195–200.

[6] A. Uht, "Uniprocessor performance enhancement through adaptive clock frequency control," *IEEE Transactions on Computers*, vol. 54, no. 2, pp. 132–140, 2005.

[7] J. Cortadella, M. Kishinevsky, and B. Grundmann, "Synthesis of synchronous elastic architectures," in *Proc. ACM/IEEE Design Automation Conference*, Jul. 2006, pp. 657–662.

[8] P. C. McGeer and R. K. Brayton, *Integration Functional and Temporal Domains in Logic Synthesis.* Kluwer Academic Publishers, 1991.

[9] S. M. Nowick, K. Y. Yun, A. E. Dooply, and P. A. Beerel, "Speculative completion for the design of high-performance asynchronous dynamic adders," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 1997, p. 210.

[10] S.-L. Lu, "Speeding up processing with approximation circuits," *Computer*, vol. 37, no. 3, pp. 67–73, Mar 2004.

[11] A. K. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: A new paradigm for arithmetic circuit design," *Proc. Design, Automation and Test in Europe (DATE)*, pp. 1250–1255, March 2008.

[12] K. Ravi, K. McMillan, T. Shiple, and F. Somenzi, "Approximation and decomposition of binary decision diagrams," in *Design Automation Conference*, 1998, pp. 445–450.

[13] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, "Logic decomposition during technology mapping," *IEEE Transactions on Computer-Aided Design*, vol. 16, no. 8, pp. 813–834, Aug. 1997.

[14] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," U.C. Berkeley, Tech. Rep., May 1992.

[15] K. Singh, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli, "Timing optimization of combinational logic," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 1988, pp. 282–285.