Techniques for Functional Test Pattern Execution

Inki Hong and Miodrag Potkonjak
UCLA Computer Science Department
Los Angeles, CA 90095-1596 USA

Abstract— Functional debugging of application specific integrated circuits (ASICs) has been recognized as a very labor-intensive and expensive process. We propose a new approach based on the divide and conquer optimization paradigm for the functional test pattern execution. The goal is to maximize the simultaneous controllability of an arbitrary set of the user selected variables in the design at the debugging time for facilitating the functional test pattern execution while minimizing the hardware overhead. The approach imposes minimal restriction on register sharing so that the synthesized designs will have the desired characteristic while minimizing the additional hardware overhead and minimizing the disruption of the optimization potential when scheduling, allocation and binding tasks in high-level synthesis are performed. The effectiveness of the proposed approach is demonstrated on a number of designs.

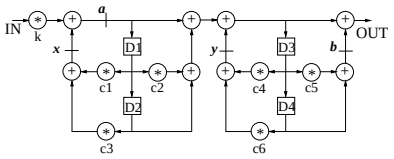
I. Introduction

Functional debugging of ASIC designs is a difficult activity mainly due to the limited controllability and observability of the storage elements or variables in designs. This situation is likely to become more serious in the future since the key technological trends indicate that the percentage of controllable and observable variables in designs will steadily decrease. There is, however, a key technological factor, the clock rate, likely to help remedy the situation. The clock rate has been doubling every three years. Thus, higher levels of resource sharing become feasible and economically desirable with each new generation of technology, while at the same time resource sharing is becoming increasingly important for satisfying applications requirements.

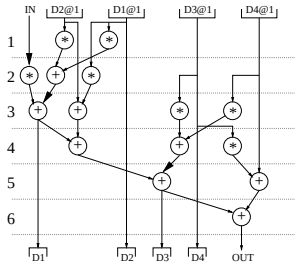
In this paper, we address the test pattern execution (FTPE) phase of the functional debugging of ASIC designs in high level synthesis and develop techniques for design-for-debugging which facilitates the test pattern execution phase while minimizing its overhead.

To illustrate the key ideas behind the new approach, consider the design shown in Figure 2(a). The design is a cascade form Avenhaus filter composed of direct-form II sections. For the simplicity of illustration, we assume that all operations take one control step. The critical path is six control steps. It is simple to verify that the minimum hardware requirement is two adders and two multipliers.

Fig. 1. Motivational example: cascade-form Avenhaus filter



(a) CDFG



(b) Scheduled CDFG

To debug the functionality of the design, in the first step, functional test patterns, which are most likely to make functional errors visible, are generated. These patterns are given as the specific values of a set of variables in the design that need to be set in the same iteration. The functional test patterns should be executed by providing the proper values to the primary inputs. It is hard to execute functional test patterns in the original design due to several reasons. First, there is only one primary input while there are four functional delays and many intermediate variables, which results in low direct controllability of the design. Secondly, there are four loops in the design due to its sequentiality. It is necessary that at least one of the variables in a loop is controllable in order to control variables in the loop.

To overcome the difficulties, the key design element to be explored is resource sharing, especially register sharing between primary inputs and intermediate variables. Our approach is to impose minimal restriction on register sharing so that the synthesized design will have the desired characteristic while minimizing the additional hardware overhead and minimizing the disruption of the optimiza-

tion potential when performing scheduling, allocation and binding tasks in high-level synthesis. In order to explain the proposed approach, we need to introduce the notion of a complete cut of a computation.

Definition 1 (Complete Cut) *A complete cut is a subset of variables which includes at least one variable from every loop in control data flow graph (CDFG). Any incomplete cut is called partial cut.*

A complete cut bisects all loops in the CDFG. For example, all functional delays or state variables form a complete cut. A complete cut plays an important role to overcome the difficulties mentioned above. Providing controllability to the variables in a complete cut allows us to isolate the consecutive iterations. By sharing registers between the primary inputs and the variables in the complete cut such that every cut variable shares a register with one of the primary input variables, direct controllability to the cut variables is provided. Based on this idea, our approach selects a complete cut and imposes a minimal restriction on scheduling, allocation and binding phases such that every cut variable should share a register with one of the primary input variables.

Consider a complete cut composed by all four state variables. Since all variables in the cut and the primary input **IN** are concurrently alive in control step 1 regardless of scheduling algorithm to be used, they have to be stored in five different registers. Therefore, the direct of the cut variables is not provided by the primary input **IN**. Allocation of extra hardware, four sets of register-I/O connections with write ability to the registers for the state variables can provide direct controllability. Without the extra hardware, it is impossible or very hard at best to set arbitrary requested values in the functional delays. Providing the ability of resetting the functional delays to a fixed value, e.g., 0, may reduce the complexity, but the resetting has a serious drawback of interrupting functionality of the computation.

Consider a different complete cut consisting of the variables x and y (the bold lines in Figure 2(b)). In this case, all the variables in the cut and the primary input **IN** can share a register because they can be scheduled such that they are not alive simultaneously. The two phase clocking scheme in which read operations are performed in the first phase and write operations are performed in the second phase is assumed. One such schedule is shown in Figure 2(b). The schedule requires only two adders and two multipliers which are exactly the lower bound.

Now the complete cut can be used to control other variables in the design. Any other variable can be described by a linear equation of the variables in the complete cut and the primary inputs. Thus, by solving a system of linear equations for the requested set of variables to control, the feasibility of the request can be determined.

We stress here that our technique is not limited to linear designs. For nonlinear designs, a technique, which isolates all output variables of nonlinear operations through register sharing with the primary input variables, is employed.

To the best of our knowledge this is the first attempt to study the functional test pattern execution problem in high level synthesis and in general CAD techniques. This is also the first approach which addresses the functional test pattern execution problem for the custom ASIC designs in an optimization-intensive way.

The rest of the paper is organized in the following way. In the next section we introduce several definitions. Section III outlines the previous work. The proposed approach is described in Section IV. In Section V, the optimization problem for minimizing the hardware overhead in achieving test pattern execution requirement is defined, its computational complexity is established, and an efficient algorithm is proposed for the problem. In Section VI, we present experimental results. Finally, we draw conclusions in Section VII.

II. Preliminaries

In this Section, we define several terminologies used in this paper.

Definition 2 (Irredundant Complete Cut) *An irredundant complete cut is a complete cut in which removal of any variable in the cut results in an incomplete cut. Any other complete cut is called redundant complete cut.*

Definition 3 (Partially Redundant Complete Cut) *A partially redundant complete cut is a complete cut in which removal of any variable in the cut, originating from a linear operation, results in an incomplete cut.*

A partially redundant complete cut is simply an irredundant complete cut for a linear design.

III. Related Work

Potkonjak, Dey, and Wakabayashi [8] proposed a behavioral synthesis technique for design-for-debugging on the error detection phase of debugging. While the behavioral synthesis for functional debugging has not received much attention, behavioral synthesis for manufacturing testing has been extensively discussed [6, 4, 5].

IV. Design for Functional Test Pattern Execution

A. Global Flow of the Approach

The goal is to maximize the simultaneous controllability of an arbitrary set of the user selected variables in the design at the debugging time for facilitating the functional test pattern execution while minimizing the hardware overhead. There are several problems that make it hard to achieve high controllability. First, there are usually only a few primary inputs in designs. Primary inputs are directly controllable and are instrumental in providing controllability of other variables. Secondly, there are loops in designs. To control variables in a loop, we must be able to control one of the variables in the loop. Thirdly, there are often nonlinear operations in designs.

Fig. 2. The algorithm for computing the maximum number of iterations required for any functional test pattern execution

All strongly-connected components (SCCs) of the computation are identified;
A graph $G = (V, E)$ which describes the dependencies between SCCs is constructed;
For each vertex v in V
  $W(v)$ = the number of functional delays in v;
For each vertex v in V
  If there are no incoming edges to v
  Then $N(v) = W(v)$;
  Else $N(v) = \max_{\forall w:(w,v)\in E}(N(w) + W(v))$;
Return $\max_{\forall v \in V} N(v)$;

Nonlinear designs are particularly harder to achieve controllability than linear ones. The key design element to address the three problems is resource sharing, especially register sharing between primary inputs and variables in designs. Our approach is to impose minimal restriction on register sharing so that the synthesized designs will have the desired characteristic while minimizing the additional hardware overhead and minimizing the disruption of the optimization potential when performing scheduling, allocation and binding tasks in high-level synthesis.

The idea is to find a complete cut that satisfies the following requirements:

- All output variables of nonlinear operations are included in the cut.
- The cut is a partially redundant complete cut.
- The cut minimizes the disruption of optimization potential in scheduling, allocation and binding.
- The cut includes the minimum number of variables.

We explain the reasons behind the requirements. First, it is difficult to indirectly control the output variables of nonlinear operations from other directly controllable variables because it may involve solving a system of nonlinear equations. Thus, the output variables of nonlinear operations are directly controlled by the primary inputs through register sharing. Secondly, more variables in the cut impose more restriction on the later synthesis steps. Thus, if a variable can be removed from a complete cut without resulting in a partial cut, it should be removed. Thirdly, because we need to impose the restriction that every variable in the cut should share a register with any one of the primary input variables, the cut should be selected to impose minimal restriction. Finally, because direct controllability of a variable requires hardware overhead described in Section II, the number of variables in the cut should be minimized to reduce the debugging hardware overhead.

After an optimal complete cut is found, we perform scheduling, allocation and binding such that every variable in the cut shares a register with any one of the primary input variables. If we fail to find such a cut, allocation of extra hardware, register-I/O connections with write ability to the registers can provide direct controllability to the cut variables which are incompatible with all primary inputs. The goal is to minimize the allocation of the extra hardware.

Given functional test patterns, a system of linear equations for the requested control variables with a function of only the variables in the complete cut and the primary input variables is constructed and solved.

B. Maximum Number of Iterations for Functional Test Pattern Execution

It is interesting to know the number of iterations required for any functional test pattern execution. A simple maximum bound is the number of functional delays plus 1. The exact number, however, is smaller or equal to the number depending on the dependencies between functional delays. The maximum number of iterations required for any functional test pattern execution can be computed using the algorithm described in Figure 2.

V. Selection of Optimal Partially Redundant Complete Cut

We use *distribution graphs* (DG's) similar to the ones used in force-directed scheduling by Paulin and Knight [7]. The possible control step at which an operation is scheduled spans between its *as soon as possible* (ASAP) control step and *as late as possible* (ALAP) control step. By noting that an operation can be assigned to any control step between its ASAP and ALAP control steps, the lifetime of a variable before scheduling spans between the second phase of the ASAP control step of the source operation and the first phase of the maximum of ALAP control steps among all destination operations. The lifetime of a primary input variable includes only the control steps that the variable is used and spans the first phase of the minimum of ASAP control steps and the first phase of the maximum of ALAP control steps among all destination operations. We assume uniform probability of assigning an operation to any feasible control step. The height of a rectangle in a distribution graph for a variable means the probability that the corresponding control steps of the rectangle belong to the lifetime of the variable after scheduling is performed. Lemmas 1 and 2 show that any variables can share a register if the distribution graphs for the variables do not have such intersecting rectangles that the probabilities of the rectangles for all variables are 1.

Lemma 1 *If the distribution graphs for variables x and y do not have such intersecting rectangles that the probabilities of the rectangles for the two variables are 1, then the two variables x and y can share a register by scheduling, allocation and binding.*

The Lemma 1 can be easily extended for n variables.

Lemma 2 *If the distribution graphs for variables $x_1, x_2, \cdots, x_n$ do not have such intersecting rectangles that*

Fig. 3. CDFG for a low-pass lattice filter designed for communications application

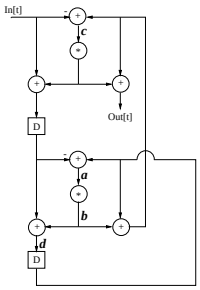
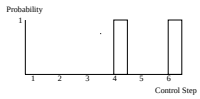
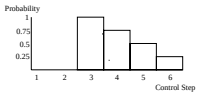


Fig. 4. Some distribution graphs for the CDFG in Figure 3

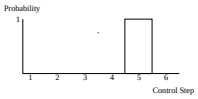


(a) DG for a variable IN

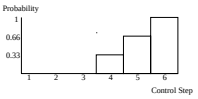
(b) DG for a variable a



(c) DG for a variable b



(d) DG for a variable c



(e) DG for a variable d

Fig. 5. The pseudo code of the heuristic for the Optimal Complete Cut for Functional Test Pattern Execution problem

```
Cuts C_n and C_l are initially set to empty;
Include all output variables of nonlinear operations in the
cut C_n;
If the cut C_n is a complete cut
Then return C_n;
Else {
  Include all functional delay variables in the cut C_l;
  C_t = C_l;
  For each variable v in C_t
    Compute COST(v);
  COST_old = COST(C_t);
  COST_best = COST_old;
  Identify all strongly-connected components;
  Repeat {
    Randomly select a variable x from C_t with a probability
    proportional to its COST;
    Replace x with the output variables of all destination
    operations in strongly-connected components;
    Remove all input variables of the destination operations of
    x from C_t;
    For each newly added variable v in C_t
      Compute COST(v);
    COST_new = COST(C_t);
    If (COST_new < COST_best) {
      COST_best = COST_new;
      C_l = C_t;
    }
  } Until (no improvement in k consecutive moves);
  Remove redundant variables from C_l;
  Return C_n ∪ C_l;
}
```

the probabilities of the rectangles for all the variables are 1, then the n variables $x_1, x_2, \cdots, x_n$ can share a register by scheduling, allocation and binding.

We define the cost for a variable x, $COST(x)$, and the cost for a cut C, $COST(C)$ and use the cost measures to compare different variables and cuts, respectively, where $COST(x)$ = intersecting areas between x and primary inputs if x is compatible with some primary input and $COST(x) = \infty$, otherwise. For a cut C, $COST(C) = \sum_{\forall x \in C} COST(x)$.

To illustrate the creation of distribution graphs for variables, consider the design shown in Figure 3. The design is a sub-part of a low-pass lattice filter of degree 3, which is designed to be used in a 12-channel transmultiplexer [2]. The critical path of the design has six control steps. The available time is also six control steps. The distribution graphs for the variables **IN**, a, b, c and d in Figure 3 are shown in Figure 4. Note that $\{a\}$, $\{b\}$ and $\{c, d\}$ are irredundant complete cuts.

It is simple to see that the variables a and **IN** can share a register without disrupting the optimization potential because there are no intersecting rectangles. The variables b and **IN** can also share a register, but should disrupt the optimization potential because the destination multiplication operation of the variable b must be scheduled at control step 3 even though it can be scheduled in any control step between 3 and 6. The variable c can share a register with **IN**, but the variable d can not because d and **IN** are both alive at control step 6. Therefore the complete cut $\{a\}$ is clearly the best choice for our objective. Our cost measure reaches the same conclusion because $COST(\{a\}) = 0$, $COST(\{b\}) = 0.5$ and $COST(\{c, d\}) = \infty$.

To facilitate the construction of the complete cut, we use strongly-connected components. Every operation in non-trivial strongly-connected components (those with more than one operation) is part of a loop. [1] By Lemma 3, a complete cut for a computation can be obtained by the separate complete cuts for all strongly-connected components of the computation.

Lemma 3 *The complete cuts for all non-trivial strongly-connected components form a complete cut for the entire computation.*

This lemma can be easily extended for partially redundant complete cuts.

[1] There may be more than one loop in a strongly-connected component. See an example in Figure 3. The computation is a strongly-connected component which has two loops.

The problem to find an optimal complete cut for functional test pattern execution is NP-complete since the FEEDBACK ARC SET problem [3] can be reduced to our problem by only considering that the number of variables in the complete cut is minimized.

Due to the computational complexity, we have to resort to a heuristic method to find a good solution for the problem. The pseudo code of the heuristic for the Optimal Complete Cut for Functional Test Pattern Execution problem is provided in Figure 5. Checking if a cut is complete can be efficiently done by removing all the corresponding edges of the cut from the CDFG and running any directed graph cycle detection algorithm on the graph [1]. Replacing a variable with the output variables of all destination operations in strongly-connected components maintains the completeness of the cut. When performing the replacement, we can also remove all input variables of the operation from the complete cut because all the loops containing the input variables also contain the output variables. As a postprocessing step, we remove all redundant variables to make the complete cut to be partially redundant complete cut. The redundancy of a variable v can be checked by removing all the corresponding edges of the complete cut except v from the CDFG and running any directed graph cycle detection algorithm on the graph. If there is no cycle, v is redundant.

VI. Experimental Results

We applied our approach to design for functional test pattern execution on a set of 10 industrial examples. Table I provides the characteristics of the considered designs and presents the experimental results for the designs. The second order Volterra filter, third order Volterra filter and LMS audio formatter are nonlinear designs while the rest are linear designs. The sixth column corresponds to the timing requirements in terms of the critical path length. The ninth column presents the hardware overhead of the new approach for the functional test pattern execution. The tenth column provides the maximum number of iterations for any functional test pattern execution on the design.

All designs were synthesized using the Hyper high-level synthesis system from University of California, Berkeley [9]. Original synthesized designs without considering functional test pattern execution showed poor functional test pattern execution ability. In all cases, we were required to use extra I/O pins. Due to the interconnect and register file model of the Hyper system, the hardware overhead for controlling registers in debugging mode was not incurred. The other hardware overhead in I/O pins and registers was minimal. In all cases, no extra I/O pins were required. In only a few cases, one extra register was required. We note that for 12th order IIR filter, the number of registers actually decreased by 1 when the available time was equal to the critical path length. Since the scheduling, allocation and binding algorithms the Hyper system uses are randomized, adding a few "good" constraints may result in better solutions. The constraints we impose on register sharing are minimal and sometimes beneficial since the selected variables to share registers with primary inputs are highly compatible with primary inputs. For all examples, no area overhead was incurred. The designs who used one more register resulted in less area for interconnect.

VII. Conclusion

We proposed a new approach for the functional test pattern execution phase of the ASIC functional debugging process in high level synthesis. The goal was to maximize the simultaneous controllability of an arbitrary set of the user selected variables in the design at the debugging time for facilitating the functional test pattern execution while minimizing the hardware overhead. The new approach, based on the divide and conquer optimization paradigm, provided the full controllability of each component by exploiting resource sharing of a set of variables in the components and primary inputs. The selection of the variables introduced an interesting combinatorial optimization problem. We established its computational complexity and proposed a non-greedy heuristic algorithm. The effectiveness of the proposed approach was demonstrated on a number of designs.

References

[1] T.H. Cormen, C.E. Leisserson, and R.L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1990.

[2] A. Fettweis. Wave digital filters: theory and practice. *Proceedings of the IEEE*, 74(2):270–327, 1986.

[3] M.R. Garey and D.S. Johnson. *Computer and Intractability: A Guide to the theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, 1979.

[4] F.F. Hsu, E.M. Rudnick, and J.H. Patel. Enhancing high-level control-flow for improved testability. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 322–328, 1996.

[5] T.-C. Lee, W.H. Wolf, and N.K. Jha. Behavioral synthesis for easy testability in data path scheduling. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 616–619, 1992.

[6] C. Papachristou and J. Carletta. Test synthesis in the behavioral domain. In *International Test Conference*, pages 693–702, 1995.

[7] P.G. Paulin and J.P. Knight. Force-directed scheduling for the behavioral synthesis of ASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(6):661–679, 1989.

[8] M. Potkonjak, S. Dey, and K. Wakabayashi. Design-for-debugging of application specific designs. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 295–301, 1995.

[9] J. Rabaey, C. Chu, P. Hoang, and M. Potkonjak. Fast prototyping of data path intensive architectures. *IEEE Design & Test of Computers*, 8(2):40–51, 1991.

TABLE I

THE CHARACTERISTICS AND EXPERIMENTAL RESULTS OF THE EXAMPLES USED TO DEMONSTRATE THE EFFECTIVENESS OF THE NEW APPROACH. CP - CRITICAL PATH LENGTH

Design	# of Primary Inputs	# of Nonlinear Operations	# of SCCs	# of Variables	Available Time	Initial /Final Area	Clock Cycles	Overhead Pins/ Registers	# of Iterations for FTPE
12th order IIR	1	0	6	56	cp	48.13	13	0/-1	7
					1.1*cp	48.03	14	0/0	7
					5*cp	10.48	65	0/0	7
Avenhaus direct form	1	0	1	40	cp	92.49	10	0/0	9
					1.1*cp	54.17	11	0/0	9
					5*cp	20.32	50	0/0	9
Avenhaus cascade form	1	0	4	34	cp	12.27	10	0/0	9
					1.1*cp	8.97	11	0/0	9
					5*cp	5.98	50	0/0	9
Avenhaus parallel form	1	0	4	39	cp	11.47	8	0/0	3
					1.1*cp	8.67	9	0/0	3
					5*cp	6.71	40	0/0	3
Avenhaus continued fraction	1	0	1	35	cp	37.80	18	0/0	9
					1.1*cp	32.67	20	0/0	9
					5*cp	14.02	90	0/0	9
Avenhaus ladder	1	0	1	50	cp	16.87	27	0/0	9
					1.1*cp	11.45	30	0/0	9
					5*cp	5.73	135	0/0	9
DAC	1	0	2	354	cp	27.06	132	0/1	3
					1.1*cp	24.61	145	0/0	3
					5*cp	16.02	660	0/0	3
2nd order Volterra	1	6	1	29	cp	10.81	12	0/0	1
					1.1*cp	10.81	13	0/0	1
					5*cp	5.40	60	0/0	1
3rd order Volterra	1	12	1	50	cp	12.50	20	0/0	1
					1.1*cp	9.98	22	0/0	1
					5*cp	2.10	100	0/0	1
LMS audio formatter	1	2	3	464	cp	73.26	202	0/1	4
					1.1*cp	56.24	222	0/0	4
					5*cp	38.83	1010	0/0	4