A Quantitative Approach to Functional Debugging

Darko Kirovski and Miodrag Potkonjak
Computer Science Department, University of California, Los Angeles, CA 90095-1596

Abstract

We introduce a novel cut-based debugging paradigm. It coordinates design emulation and simulation and enables fast transition from one to another. Emulation or functional implementation is used for fast application execution; simulation provides complete design observability and controllability. The implementation of the new debugging approach poses several CAD tasks. We formulate the optimization tasks and develop constraint-based heuristics to solve them. Effectiveness of the approach is demonstrated on a set of designs.

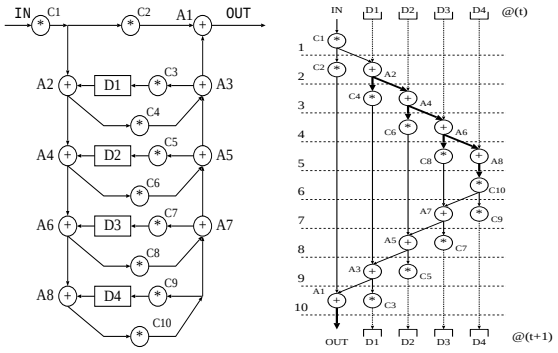
1 Introduction

Functional debugging dominates both development time and cost of modern design process [12]. The existing, widely employed, approaches for functional debugging are design simulation and chip emulation or fabrication. Design simulation results in complete observability and controllability of the simulated architecture. Unfortunately, simulation is extremely computation intensive process. Emulated or fabricated designs on the other hand provide full execution speed, but limited observability and controllability of the design internal structure.

We propose a new paradigm for debugging of integrated circuits. The paradigm integrates design emulation and simulation in a way that the advantages of the two are combined, while the disadvantages are eliminated. The debugging method is fully transparent; it can be used in conjunction with any existing or future synthesis system or manual design approach as a postprocessing step. To develop and explain the proposed debugging process we introduce the notion of a *complete cut* of a computation. A *complete cut* is a set of variables generated within one computation iteration which bisects all possible paths in the computation. Such set of variables fully determines the state of the machine.

The functional debugging process, includes four steps: test input generation, error detection, error diagnosis, and error correction. The test input vectors generated in the first phase are used to drive the design emulation. While primary inputs, cuts and outputs are stored in the permanent storage of the supervising workstation, error detection techniques are deployed to signal to the workstation whether the emulation should be terminated. Upon error detection, the error has to be localized and characterized. This is done by running the design simulation on a workstation starting from a designer selected breakpoint. The computation is continued from the selected breakpoint due to the recorded cuts and their primary inputs and outputs. Error is diagnosed using a symbiosis of the design simulation and emulation. Design simulation is used for accurate system function test, while the emulation is used for fast execution of the test input vectors. The exchange of information between the simulation and emulation domain is accomplished by observing breakpoints' complete cuts of both test domains. Upon error diagnosis the emulator is updated or built-in fault tolerance mechanisms are activated. The ability to read/write breakpoint cuts from/to the emulated design is enabled by the design for debugging postprocessing step. It inserts hardware resources into the initial design specification.

The diagnosis approach and accompanying optimization issues are illustrated using 4th order continued fraction infinite impulse response (CF IIR) filter [3]. Figure 1 shows the computation before and after scheduling.



(a) Computation CDFG. (b) Scheduled CDFG.

Figure 1: Optimal cut example for 4th order CF IIR filter.

The design for debugging aims for allocating minimal hardware resources to enable design observability and controllability. One possible solution is to select variables $D1$, $D2$, $D3$, and $D4$ as a complete cut. Since these variables are concurrently alive, they have to be stored in four different registers. In order to provide design controllability and observability, the designer has to have the ability to read/write into those registers from the designated I/O pins. Since four registers are in the cut, four register-to-I/O connections have to be allocated. If the cut is defined among the output variables of adders $A2$, $A4$, $A6$, and $A8$ (bold lines), only one register is required to hold the values of the cut variables, since they are not alive simultaneously during the computation. Now, only one instance of the selection hardware is dedicated to the register which holds the cut. Cut dispensing is performed in four consecutive control steps (cycles 2, 3, 4, and 5). Note that in both cases the variable included in the cut but not mentioned is the actual output of the chip.

2 Related Work

The most comprehensive treatment of debugging has been conducted in the software compiler domain. Several debuggers, such as GDB [10] and Purify [1], have been widely used. The majority of efforts related to debugging in the compiler domain have been dedicated to symbolic level debugging on uniprocessors.

Numerous processors are supported by in-circuit emulators which enable efficient debugging [7], [11]. An in-

0-89791-993-9/97 $10.00

circuit emulator of a processor is a system that can imitate the behavior of the microprocessor. The major drawback of in-circuit emulation is high cost. An FPGA-based emulation is sometimes used during development of new microprocessors (e.g. Quickturn, MentorGraphics). Potkonjak et al. [8] propose a design for debugging technique. Their technique focuses only on the error diagnosis phase. It assumes that the designer specifies the debugging variables at design time and only provides controllability and observability of the specified variables.

3 Preliminaries

We outline all relevant debugging and design for debugging background material. For the sake of conceptual simplicity all our examples follow a synchronous computation data flow model [6]. Such restriction on the computation model imposed by the proposed debugging technique is very mild. We assume deterministic hardware behavior and continuous semi-infinite operation mode (not necessarily periodic). We do not impose any register-transfer level restrictions on the interconnection scheme of the hardware model. The four key debugging assumptions are:

- The design is fully specified and its functionality and realization is not disturbed by the debugging process with the exception of enabling the user to write into specific controllable variables.
- In order to support debugging, we allocate additional debugging hardware to satisfy all debugging requirements. The goal is to add as little as possible hardware and in particular, I/O pins.
- For proper debugging support, all variables in the optimal cut should be controllable and observable in the same iteration of the computation.
- Required controllable and observable variables are not known at compile or synthesis time. The goal is to enable simultaneous complete controllability and observability of all user defined variables.

4 The New Debugging Approach

The essence of the new approach is the establishment of the concept of a complete cut. It provides information exchange between the two execution domains with small hardware overhead. *A complete cut is subset of variables which contains sufficient information required to correctly continue the computation.* A complete cut corresponds to a subset of variables in the computation which bisects all paths between states that delimit successive computation iterations. If one has complete controllability and observability over the values for all variables in the complete cut for a specific breakpoint, the computation can be continued functionwise correctly from that breakpoint. The cut-based debugging approach is conducted using the following two phases of the design and debugging process:

- Design postprocessing - We define the cut from the initial design specification at the functional level and augment the design specification with cut statements.
- Error diagnosis - Simultaneous and coordinated execution of the emulator and appropriate simulator.

In the first phase, the computation iteration at the behavioral level of specification is logically partitioned into two or more components such that the cut between the partitions is complete. The synthesis support for exchange of information between simulation and emulation has the following three degrees of design freedom:

- The determination of variables which form the cut;
- The determination of the exact control step when a particular variable is read from its register or replaced by a user specified value.
- The assignment of specific sets of I/O pins used to transfer variables to or from the chip.

The optimization procedure has a unique goal: to add minimum hardware resources into the initial design, while obtaining its full controllability and observability.

The debugging environment consists of a workstation and interfaced fabricated integrated circuit or emulator. The designer starts the debugging process by preparing a set of test inputs intuitively or using a software tool which searches the test input generation database. Once the test vectors are determined, functional testing starts at the emulation testbed. While test inputs are forwarded to the chip, the workstation stores the debugging information in permanent storage. For each iteration of the computation, its primary inputs, complete cut and resulting outputs are stored into a workstation supervised cut-database.

If error is detected, the next step in the debugging flow is to provide the designer with a set of tools to diagnose, i.e. locate and characterize the error. At this point the cut-database is searched for cuts which might point and clear out the error. The designer, guided by a debugging guidance tool, selects the most suspicious cut and runs a cycle-accurate chip simulation on the workstation. If the simulator provides sufficient information, the error can be discovered at that point. Otherwise, the designer continues the investigation by intuitively modifying internal registers likely to discover the error. The modification is followed by the simulation of the next iteration's cut. Once the cut is retrieved from the cycle-accurate simulator, all variables in the cut-set are injected into the design emulator while tempering the appropriate input vector. The generated outputs in the next few iterations are likely to uncover the error.

5 Synthesis for Debugging

The determination and integration of the inserted debugging resources is performed by a number of fully modular tasks. The goal is to select a set of variables which represent a control data flow graph (CDFG) cut such that all variables in the set are stored in minimal number of registers. The computation graph and timing bounds have to allow all variables in the cut-set to be output from the chip through a designated set of I/O pads within a single computation iteration. The synthesis flow resumes searching for an optimal output scheduling of the cut-set variables. The schedule has to be determined so that the number of connections between the output pins and registers containing the cut-set variables is minimal. After cut-set variables are assigned and scheduled, the initial specification is updated with the set of resources which enable cut output.

The design flow points out to three fundamental problems which enable optimal integration of debugging resources with the initial design specification. Each of these problems is proven to be NP-complete in [5]. In the first phase of the design for debugging flow we consider two fundamental problems: cut selection and checking whether the cut-set can be scheduled out from the chip. In the second phase, when the optimal cut-set is defined, the problem that arises is how to schedule the cut-set variables to specific I/O pins in such a way that minimal number of register-to-port connections are inserted into the initial design specification.

Cut Selection. The goal is to find a cut-set of variables S which partitions the control data flow graph into two subgraphs such that there exists no path in the initial CDFG which connects the two partitions and does not require computation of at least one variable in the set. In

addition to that, the variable set has to be of minimal cardinality of registers that store the cut variables. In order to explain this problem and describe the developed heuristics, we introduce a set of definitions.

A scheduled CDFG is given for each variable V_i: the control step C_i^{start} when it is created and steps C_i^{first}, C_i^{last} when V_i is used for the first and last time. An assigned CDFG has all its variables assigned to distinct registers. The read life-time of variable V stored in register R begins at the control step when variable V is computed until the control step when variable V is overwritten by another variable W or it is recomputed. In the latter case the read life-time of a variable spans across the iteration. *Write life-time of variable V stored in register R starts at the control step when variable V is computed and ends at the step when V is used for the first time. An input sensitive graph - ISG - is a directed graph with multiple input nodes and single output node. Arbitrary number of edges is connected to a single input or single output. At most one edge connects an output to an input. A port is a set of K I/O pins. When variable V is assigned to port P, then V is output or input in its entirety through port P in one control step.*

Algorithms that support controllability are identical with the exception that write life-times are used instead of read life-times. The problem formulation is stated using the standard Garey-Johnson format [4].

PROBLEM: Optimal Cut-Set for Debugging.

INSTANCE: *Control data flow graph G with read life-times of its variables, variable-to-register assignments, P ports, and a set S of control steps when each port is busy.*

QUESTION: *Is there a subset of variables V such that each path in the control data flow graph G contains at least one variable $V_i \in V$, the cardinality of the set of registers that contains each variable $V_j \in V$ equals K, and there exists such schedule that each variable $V_j \in V$ can be output through P ports at control steps not included in S?*

ISG = $Construct\_ISG(CDFG)$.
ISG = $Input\_Sens\_Trans\_Closure(ISG, CDFG)$.
Search in the solution space for the optimal input-sensitive dominating set D in the ISG. Optimality of the set is described as minimal number of registers storing all variables in D. There has to exist a schedule such that all variables in D can be output through a given set of ports.

Figure 2: Pseudo-code for the optimal cut search.

For each pair of edges $E_{M_a^o,M_{b,m}}, E_{M_b^o,M_{c,m}} \in ISG$ such that $E_{M_a^o,M_{b,m}}$ connects M_a^o to M_b and $E_{M_b^o,M_{c,m}}$ connects M_b^o to M_c
If the difference in control steps between the starts of read life-times of nodes (variables) A and C is less or equal than the total number of control steps in one iteration
Insert edge $E_{M_a^o,M_{c,m}}$ connecting M_a^o and $M_{c,m}$.

Figure 3: $Input\_Sens\_Trans\_Closure(ISG, CDFG)$.

We developed a new least-constraining most-constrained global heuristic which constructs its solution based on the analysis of the ISG presentation of the initial CDFG. The pseudo-code of the heuristic is given in Figure 2. The ISG is built from the CDFG according to the pseudo-code in Figure 4. An important step in the algorithm is the input-sensitive transitive closure operation performed on the ISG. This operation is described using the pseudo-code in Figure 3. Using the definition of scheduled and assigned input sensitive graph, the initial problem can be reformulated as:

P: Input Dominating Set of an ISG.

I: *Input sensitive graph G with read life-times of its variables, variable-to-register assignments. P ports and a set S of control steps when each port is busy.*

Q: *Is there an input dominating set of variables V such that each input is covered with at least one variable in V, the cardinality of the set of registers that contains all variables from V equals K, and there exists a schedule such that all variables in V can be output through P ports at control steps not included in S?*

For each node $N_i \in CDFG$
Create a node $M_i \in ISG$.
For each edge E_{N_i,N_j} directed from N_i to N_j
Create an input port $M_{i,m}$ for M_i,
where m is the index of the input port.
For each edge E_{N_i,N_j} directed from N_i to N_j
Create an edge $E_{M_i^o,M_{j,m}}$ which connects M_i^o and $M_{j,m}$,
where M_i^o is the output port of M_i,
and $M_{j,m}$ is the m^{th} input of M_j.

Figure 4: $Construct\_ISG(CDFG)$.

The pseudo-code of the proposed heuristic is shown in Figure 5. The objective function that evaluates the proposed cut-sets is run-time dependent. Initially, the algorithm favors solutions with as few as possible storing registers. As the search progresses, the cost's dominating factor becomes the equilibrium of read life-time periods of the selected variables.

Preprocessing: Set of primary output variables $V_i^{output} \in ISG$ is static part of the cut. Variables read-alive during the iteration are static part of a cut.
1. Select a random subset of nodes $CS \in ISG$, such that CS covers all node inputs in ISG. Best solution $CS* = CS$.
If there does not exist a schedule such that CS can be output through P ports at steps not included in S go to 1.
Repeat $GLOBAL$ times
Update random subset of nodes $\in CS$ such that at least one node input is uncovered. Randomly select a subset of nodes $subCS$ from (ISG - CS) that covers the uncovered set of inputs. Merge $subCS$ and CS.
If Cost(CS) < Cost($CS*$ and there exists schedule such that CS can be output through P ports: $CS* = CS$.
Repeat $LOCAL$ times
$CS+ = CS*$. Update random subset of nodes $\in CS+$ such that at least one node input is uncovered. Randomly select a subset of nodes $subCS$ from (ISG - $CS+$) which covers the uncovered set of inputs. Merge $subCS$ and $CS+$.
If Cost($CS+$) < Cost($CS*$) and there exists schedule such that CS can be output through P ports: $CS* = CS+$.
Return: $CS*$ as the optimal input dominating set.

Figure 5: Search for the optimal input dominating set.

Cut-Set Output Scheduling. The design has to be able to output all variables in the cut-set through a limited number of I/O ports within a single iteration. We propose a most-constrained least-constraining heuristic technique to solve this problem.

P: Scheduling of a Set of Variables in a CDFG.

I: *Set of variables V, each with its read life-time. P ports and a set S of steps when each port is busy.*

Q: *Is there a schedule such that all variables in V can be output through P ports at control steps not in S?*

The heuristic developed for this problem assigns variables to ports in a greedy fashion (pseudo-code shown in Figure 6). First, the constraint of each control step C_i in the scheduled and assigned CDFG is calculated as the number C_i^{var} of variables being read-alive during that step. For each variable V_i, its constraint is computed as a sum of C_j^{var}, where j is in the set of control steps when V_i is read-alive. Consequently, the most constrained variable is assigned to the least constraining step. Constraints are

computed and variables are scheduled to control steps until all variables in the cut-set are not output scheduled.

Variable-to-Port Scheduling. The second phase of the synthesis for debugging process has as an input the selected cut-set from the first phase and the information about the read life-times of each variable as well as its storing register. The number of output ports is also available. The key design question is to assign each cut-set variable to a specific output port in a way that the number of connections between registers and I/O ports is minimal.

Repeat until all variables are not scheduled
For each variable V_i
If V_i can be scheduled only in one control step C_j, schedule V_i to C_j and any port P_x available at C_j.
For each control step C_i
Set C_i^{var} as number of all variables read-alive at C_i.
Schedule variable V_i with $max(Cost(V_i)) = max(\frac{\sum_{j=AllControlSteps} (C_j^{var} \cdot ReadAlive(V_i, C_j)}{ReadLifeTime(V_i)-1})$
to control step C_k which has $min(C_k^{var})$
and any port P_x still available at C_k.

$ReadAlive(V_i, C_j)$ returns 1 if V_i is read-alive at C_j. Otherwise, it returns zero. $ReadLifeTime(V_i)$ returns the number of control steps for which V_i is read-alive.

Figure 6: Pseudo-code for the cut-set output scheduling.

P: Optimal Output Scheduling of a Set of Variables in a CDFG for Debugging.

I: *Set of variables V, each with its read life-time and designated register. P ports and associated set S of control steps when each port is busy.*

Q: *Is there a schedule such that all variables can be output through P ports at control steps not included in S, and the cardinality of register-port connections is K?*

We first partition the problem into two fully modular optimization subproblems. Initially, register to port assignment is performed such that optimization requirements are met and variables of the proposed cut-set can be output through all designated ports. In the second phase, for each port, all variables that are assigned to it are scheduled for output. For the assignment problem we develop a global most-constrained least-constraining heuristic described using the pseudo-code in Figure 7. The heuristic iteratively assigns most-constrained variables (registers) to least-constraining ports. Objective function that quantifies the constraint of variable V_i is:

$$Cost(V_i) = \sum_{j=AllControlSteps} \frac{C_j^{var}}{ReadLifeTime(V_i)-1}$$

where $ReadLifeTime(V_i)$ returns the number of control steps for which V_i is read-alive. Similarly, the objective function that quantifies the constraint of port P_x is:

$$Cost(P_x) = \sum_{j=AllVariables} \frac{ReadLifeTime(V_j) \cdot Assigned(V_j, P_x)}{1+NumberOfRegistersAlreadyAssignedToP_x}$$

where $Assigned(V_j, P_x)$ returns 1 if V_j is already assigned to P_x. Otherwise, it returns 0.

For each control step C_i set C_i^{var} as number of variables read-alive during C_i.
Repeat until all variables are assigned to ports
Assign variable V_i with $max(Cost(V_i)) = max(\sum_{j=AllCtrlSteps} \frac{C_j^{var}}{ReadLifeTime(V_i)-1})$
to port P_x which has $min(Cost(P_x)) = min(\sum_{j=AllVars} \frac{ReadLifeTime(V_j) \cdot Assigned(V_j, P_x)}{1+Numberofregsalreadyassignedto P_x})$.

Figure 7: Pseudo-code for variable-to-port matching.

In our experimentations we used a probabilistic version of the heuristic which iteratively generated randomized solutions. The problem of assigning a set of variables with their read life-times to a single port is equivalent to the problem of **maximum bipartite matching** (pp. 601, [2]) which can be efficiently solved in polynomial time.

6 Experimental Results

We applied the new design for debugging approach on several real-life designs [11]. Table 1 shows the experimental results. The first column indicates the name of the evaluated design. Next five columns describe the behavioral structure of the design: the number of available steps, the length of the critical path, the number of computation variables, the number of used registers and the number of states. The performed design for debugging analysis generated complete cuts displayed in column seven. First, the number of variables included in the cut is presented and then the number of register-to-port connections required to schedule the injection and dispensing of the selected cut. In all cases only one I/O port was sufficient to support full observability and controllability.

Design	Structure					Cut
Descr.	Ctrl steps	Crtcl path	Variables	Registers	States	Vars/ Regs
Linear GE	12	12	48	19	5	8/3
controller1	24	12	48	23	5	13/1
Wavelet	16	16	31	20	15	15/1
filter	32	16	31	20	15	15/1
D/C	132	132	354	167	74	76/3
conv.	132	264	354	171	74	77/2
8th order	18	18	35	19	8	8/1
CF IIR	36	18	35	19	8	8/1
Long echo	2566	2566	1082	1056	1024	1027/5
canceller	5132	2566	1082	1061	1024	1027/4

Table 1: Experimental results: low hardware overhead.

7 Conclusion

We introduced a cut-based debugging paradigm which integrates design emulation and simulation in a way that advantages of both domains are fully utilized. We identified the associated optimization synthesis tasks and developed global heuristics to solve them. The experimental results clearly indicate the power of the new approach.

References

[1] G. Brooks, et al. "A new approach to debugging optimized code", SIGPLAN Notices, Vol.27, No.7, pp.1-11, 1992.

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

[3] R.E. Crochiere, A.V. Oppenheim, "Analysis of Linear Networks", Proc. of IEEE, Vol.63, No.4, pp. 581-595, 1975.

[4] M.R. Garey, D.S. Johnson, "Computers and intractability: a guide to the theory of NP-completeness", W. H. Freeman, San Francisco, CA, 1979.

[5] D. Kirovski, M. Potkonjak, "A Quantitaive Approach to Functional Debugging", Technical Report, CSD, UCLA, 1997.

[6] E.A. Lee and D.G. Messerschmitt, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing", IEEE Trans. on Computers, Vol. 36, No. 1, pp. 24-35, 1987.

[7] D. Phillips, "Advanced in-circuit emulation design", Mini/Micro Southwest, pp. 8/2/1-4, 1984.

[8] M. Potkonjak, S. Dey, K. Wakabayashi, "Design for Debugging of Application Specific Designs", ICCAD, pp. 295-301, 1995.

[9] J. Rabaey, et al., "Fast Prototyping of Datapath-Intensive Architectures", Design and Test of Computers, Vol.8, No.2, pp. 40-51, 1991.

[10] R.M. Stallman, et al. "A guide to the GNU source-level debugger", Free Software Foundation, Cambridge, MA, 1991.

[11] P. Wynn, "In-circuit emulation for ASIC-based designs", VLSI Systems Design, Vol.7, No.10, pp. 38-45, 1986.

[12] L. Yang, et al. "System design methodology of UltraSPARC-I", Design Automation Conference, pp. 7-12, 1995.