High Level Synthesis of Degradable ASICs Using Virtual Binding

N. Honarmand[1], A. Shahabi[1], H. Sohofi[1], M. Abbaspur[2] and Z. Navabi[1]
[1] CAD Laboratory, School of Electrical and Computer Engineering, University of Tehran, Tehran, IRAN
[2] School of Electrical and Computer Engineering, Shahid Beheshti University, Tehran, IRAN

{nima, shahabi, h_sohofi}@cad.ece.ut.ac.ir, maghsoud@ipm.ir, navabi@ece.neu.edu

***Abstract*—As the complexity of the integrated circuits increases, they become more susceptible to manufacturing faults, decreasing the total process yield. Thus, it would be desirable to develop techniques for reusing faulty dies, even with a degraded performance. In this paper, a new method for high level synthesis of degradable ASICs is presented. Our technique introduces the concept of Virtual Binding. In this approach, the operations are bound to virtual components that are linked with actual non-faulty components using a set of configuration multiplexers and flip-flops embedded in the data-path. Using virtual components simplifies the synthesis algorithm and decreases the size of generated control unit. Virtual-to-physical mapping of the components will be established by programming the configuration flip-flops after diagnosing the faulty components. The experimental results show that the area and delay overhead of the resulting circuits have acceptable values compared to the original, non-degradable circuits.**

Index Terms— Degradability, High Level Synthesis, Virtual Binding.

I. Introduction

While advances in silicon integration technology has made it possible to put more and more transistors on a single die, it has also caused the dies to be more susceptible to manufacturing faults and decreased the process yield. Manufacturing faults tend to be local and affect only a limited area of an IC. Hence, it is possible (and very desirable) to find methods to make such faulty ICs reusable. Such faults could occur in different parts of a circuit like control unit, steering logic, functional units, etc. This work focuses on faults that affect functional units.

One possible method to work around this problem is to use spare modules and to replace the faulty ones with modules of the same [1] or a different [2] type when necessary. This method will result in less-than-full hardware utilization and large hardware overhead, especially if multiple faulty modules should be considered.

Reconfiguring the faulty IC to avoid using the faulty modules and get the work done using the remaining non-faulty ones is another approach, usually referred to as *Degradability*. Traditionally, reconfiguration based techniques have been used for highly regular circuits such as memory chips [3] and processor arrays [4], but in the realm of general ASIC designs, effective methods should still be sought for.

When a functional unit, in an already built ASIC, becomes faulty, the operation(s) originally bound to that unit should be performed by other available units. This means that the circuit should use a different binding and, much likely, a different scheduling than those used in the non-faulty circuit. The need for such reconfigurations imply great challenges for designing and manufacturing efficient control units and data paths, and thus, for operation scheduling and hardware binding algorithms. Hence, it seems natural for the proposed methods to target the High Level Synthesis (HLS) [5] techniques.

Several methods, targeting the problem of degradability in ASIC domain, have been proposed in the literature. Buonanno et al [6] have considered an environment in which several processes should be implemented as separate dedicated hardware modules. They propose a high-level synthesis methodology in which each of the generated modules is both capable of the nominal execution of the related process itself, in a fault-free environment, and simultaneous execution of a reconfigured pair of processes in a fault-affected environment. In [7], authors describe how to add extra interconnect to render the resulting micro architecture reconfigurable in the presence of any single functional unit failure. L/U reconfiguration in [8] and band reconfiguration in [9] are about modifying the scheduling and binding of the original non-faulty circuit to make it suitable for the faulty one, supporting at most one fault in each hardware resource class. In [10], authors address the similar problem for circuits implemented using FPGAs. They propose a solution in which originally unused blocks and routing resources replace faulty ones.

Generally, degradability-based methods imply a design and manufacturing flow including the following steps: First, at the design stage, special HLS algorithms should be used which result in reconfigurable data paths and control units. Then,

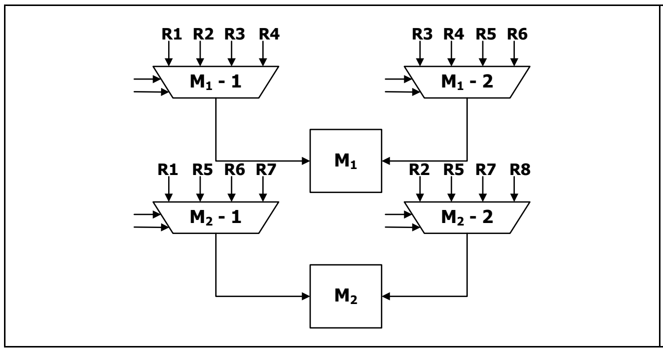

Fig. 1. A sample, non-degradable cicuit with two multipliers

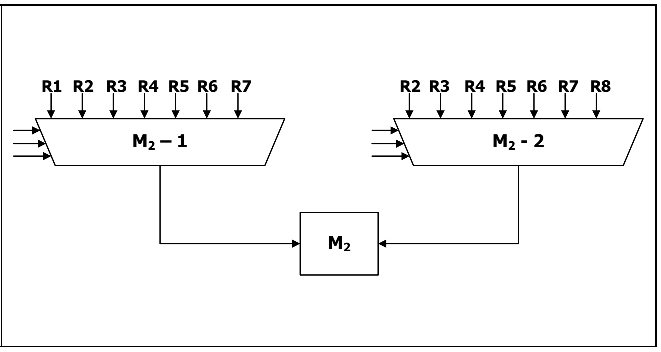

Fig. 2. Impact, on the input multiplexers of M_2, of applying Static Binding method to the circuit of Fig. 1

after the IC has been built, diagnosis techniques should be used to detect the faulty components of the circuit. Then, based on the obtained fault pattern, a proper configuration, which bypasses the faulty modules, should be chosen and programmed into the circuit. This implies that the circuit should unavoidably have some programmable elements in it and post-manufacturing reconfiguration should be possible. But, this is by no means a great overhead because widely-used built-in provisions such as JTAG or scan chains can be used for this purpose.

In this paper we propose an HLS technique that targets the functional unit faults and makes it possible to reconfigure every faulty circuit with at least one non-faulty resource of each hardware resource class. The technique does not pose any constraints on the particular scheduling and binding algorithms that can be used for synthesis. Instead, it provides a micro-architectural style that can be exploited to reduce the run-time of the synthesis process as well as the complexity of the synthesized control unit. The technique achieves this goal by reducing the number of different fault patterns that should be considered while synthesizing the degradable circuit.

We start, in Section II, by presenting a straightforward technique, called the Static Binding, that will be used for comparison. Then we present the new method, called Virtual Binding, and the related micro-architectural style. Section III presents the experimental results and Section IV gives the summary and conclusions.

II. Synthesis Techniques

A typical high-level synthesis (HLS) process includes three different steps: Control/data flow extraction, operation scheduling and resource allocation, and resource binding, from which the last two are main concerns of this work. The results of applying these three steps to a high level description are RT level models of the control unit and data path of the target hardware. If the resulting hardware is to be usable in the presence of some permanent faults, there should be some built-in provisions to reconfigure the circuit and get the work done using the remaining FUs. The problem becomes more complex when there are multiple faulty units. In this section we investigate some techniques that could be used to achieve this goal.

For the rest of paper, suppose that there are K different classes of FUs (*e.g.* ALUs and multipliers) in the circuit, and for each i-type class there are N_i instances of that type. We will denote the *j*-th instance of the *i*-type FU class by $FU_{i,j}$. The total number of FUs in the circuit will be

$$T = \sum_{i=1}^{K} N_i \qquad \text{-1-}$$

Throughout this paper, we represent the number of available FUs in a circuit *C* using a vector C_{fs}, called a FU Count Vector (FCV):

$$C_{fs}(C) = \langle n_1, \ldots, n_K \rangle \text{ in which, } 0 \le n_i \le N_i \qquad \text{-2-}$$

Each FU class has a corresponding entry in the vector, the value of which indicates the number of available non-faulty FUs of that class. Also, we represent the state of the functional units in a circuit *C*, using a vector V_{fs} with 0/1 elements, called a Fault Pattern Vector (FPV):

$$V_{fs}(C) = \langle u_1, \ldots, u_T \rangle$$
$$u_i = \begin{cases} 1, & \text{if } FU_i \text{ is functinal} \\ 0, & \text{otherwise} \end{cases} \qquad \text{-3-}$$

Each FU has a corresponding entry in the vector. The value of 0 in an entry indicates that the corresponding FU is faulty and 1 indicates that it is functional. We say that a particular FPV, V_{fs}, is compatible with some FCV, C_{fs}, if the number of *i*-type non-faulty FUs in V_{fs} is equal to n_i in C_{fs}.

We assume that for a circuit to be repairable there should be at least one non-faulty FU of each hardware class. We use the terms Repairable Fault Pattern Vector (RFPV) and Repairable FU Count Vector (RFCV) to refer to the FPV and FCV of such circuits, respectively. The total number of possible RFPVs is

$$N_{RFPV} = \prod_{i=1}^{K} (2^{N_i} - 1) \qquad \text{-4-}$$

and the total number of possible RFCVs is

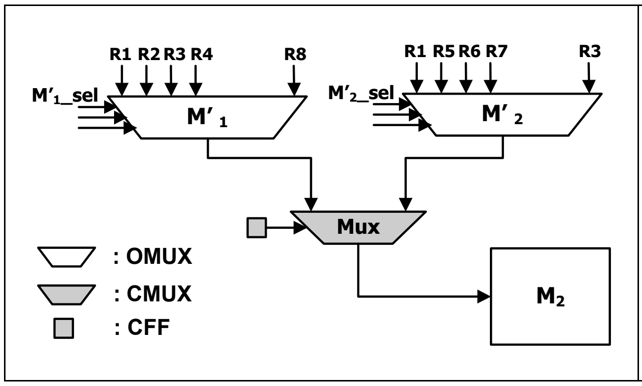

Fig. 3. Selecting the left input of multiplier M_2 based on assigned role in Virtual Binding method

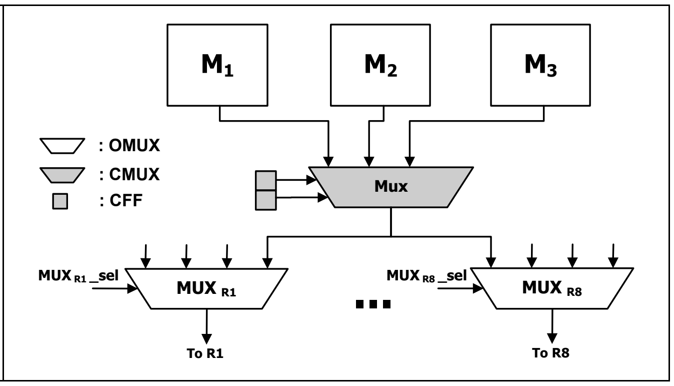

Fig. 4. Selecting the output of virtual multiplier M'1 based on assigned role in Virtual Binding method

$$N_{RFCV} = \prod_{i=1}^{K} N_i \qquad \text{-5-}$$

A. Static Binding Method

One possible technique for creating degradable circuits is to consider all the possible RFPVs of interest (*i.e.,* Those RFPVs for which we are willing to be able to reconfigure the circuit) and perform the scheduling and binding for all of them and combine all the resulting circuits into one big circuit. We call this technique Static Binding. When faults occur, we can diagnose the faulty FUs, detect the particular active RFPV and reconfigure the control unit to switch to the provisioned scheduling and binding for that RFPV. To demonstrate the technique, Fig. 1 shows the data path of a circuit with two multipliers. In this figure, M_1 and M_2 are the two multipliers and Ri, $1 \leq i \leq 8$, represent registers used to save the intermediate results of the computations.

For this circuit, there are three RFPVs in which we might be interested: When both multipliers are functional, when M_1 becomes faulty and when M_2 becomes faulty. When M_1 becomes faulty, M_2 should be able to perform those operations originally assigned to M_1. This means that this multiplier should be able to receive inputs from those registers originally connected to M_1 and provide output to those registers having M_1 as one of their possible sources. Fig. 2 illustrates the necessary changes in the input multiplexers of M_2 to cope with this situation. The case of M_2 going faulty implies similar changes in the input multiplexers of M_1.

In each of the three RFPVs above, the control unit should be able to issue proper control signals to select the inputs of the multipliers and registers. One possible solution, to make this possible, is to have one flip-flop associated with each FU in the control unit, whose state indicates the faultiness of the associated FU (*e.g.*, '0' for faulty and '1' for functional). Based on the values in these flip-flops, the control unit can detect the RFPV at hand and issue the control signals accordingly. These flip-flops could be configured either online or off line. Thus, in this technique the current RFVP of the circuit determines the functionality (or mode) of the control unit.

Figure 5 shows the Static Binding algorithm. The inputs of the algorithm are Data Flow Graph (DFG) of the design and the maximum allowable number of FUs of different classes, denoted by $<N_1,...,N_K>$. The algorithm, one by one, considers all possible RFCVs and for each RFCV schedules the given DFG. Because the scheduling algorithms only need to know the number of available FUs, determined by the RFCV, and not the exact fault pattern, the scheduling needs to be done once for each RFCV and not once for each RFPV. Any particular scheduling algorithm such as list scheduling [5] or force-directed scheduling [11] may be used here. Then, for each compatible RFVP of this RFCV, the algorithm will bind the scheduled DFG to the non-faulty FUs. Here, again, any resource sharing and binding algorithm, like those presented in [5], can be employed. The result of binding depends on the exact fault pattern and thus we should do the binding once for each RFVP. Then, the control unit will be augmented to use resulted binding when the related RFVP is active. Also, the required data transfer paths, *i.e.*, paths from registers to FUs and vice versa which are implied by the bound DFG, will be added to the data path.

While this method may seem attractive for being able to repair the circuit under many RFPVs (including multiple faulty FUs), the associated cost, in terms of area and delay overhead of the hardware and complexity of the synthesis algorithm, tends to be rather high. The BindDfg function should be invoked N_{RFPV} times (see Equation -4-), which is an exponential function of the maximum number of FUs and, this increases the synthesis time greatly. Also considering N_{RFPV} different bindings would result both in excessive increase in the size and delay of the control unit and also in introducing very large multiplexers at the inputs of functional units and registers of data path, thereby increasing the data path area and delay. These drawbacks make Static Binding an impractical method for circuits with large number of FUs and control intensive natures.

B. Virtual Binding Method

To work around the drawbacks of the Static Binding method, we propose a new technique, called Virtual Binding. In this method, the number of the different bindings to be considered in the control unit becomes a polynomial function

```
StaticBinding(dfg, <N_1...,N_K>)
begin
  for each RFCV, $C_{fs}$ = <n_1, ... n_k>, do
    scheduled_dfg = ScheduleDfg(dfg, $C_{fs}$);
    for each RFPV, $V_{fs}$, which is compatible with $C_{fs}$ do
      bound_dfg = BindDfg(scheduled_dfg, $V_{fs}$);
      Add < $V_{fs}$, bound_dfg> to the control unit;
      Add data transfer paths implied by bound_dfg to the data path;
    end for;
  end for;
end
```

Fig. 5. Static Binding Algorithm

```
VirtualBinding(dfg, <N_1...,N_K>)
begin
  for each RFCV, $C_{fs}$ = <n_1, ... n_k>, do
    scheduled_dfg = ScheduleDfg(dfg, $C_{fs}$);
    $V'_{fs} = <\underbrace{1,...,1}_{n_1}, \underbrace{0,...,0}_{N_1-n_1}, ..., \underbrace{1,...,1}_{n_K}, \underbrace{0,...,0}_{N_K-n_K}>$
    bound_dfg = BindDfg(scheduled_dfg, $V'_{fs}$);
    Add < $C_{fs}$, bound_dfg> to the control unit;
    Add data transfer paths implied by bound_dfg to the data path;
  end for;
  Add required CMUXs and CFFs to the data path;
end
```

Fig. 6. Virtual Binding Algorithm

of the number of FUs. For a circuit with 3 multipliers, for example, we need only to consider 3 different situations: 3 non-faulty multipliers, 2 non-faulty multipliers and 1 non-faulty multiplier (in contrast with 7 situations required in the Static Binding method). In general, the maximum number of bindings to be considered is equal to N_{RFCV} (See Equation -5-).

In this technique, the main idea is to bind the operations to **virtual FUs** (or **roles**) and let the virtual-to-physical mapping of the resources take place while the circuit is working, using the Configuration Multiplexers (CMUXs) and Configuration Flip-Flops (CFFs) embedded in the data path. In what follows, we refer to the virtual resources using names with prime (') sign, like M'_1 and M'_2, while names without the prime sign, like M_1, refer to the physical resources.

In Virtual Binding, the number of virtual FUs equals the number of physical FUs, and each physical FU should be able to play the role of those virtual FUs whose numbers are less than or equal to its own number, *i.e.*, FU_1 is either faulty or playing the role of FU'_1, FU_2 is either faulty or playing the role of FU'_1 (when FU_1 is faulty) or playing the role of FU'_2 (when FU_1 is non-faulty), and so on. There is a single multiplexer for each input of each virtual FU (white multiplexers in Fig. 3). Also, the input of each register will be selected using a single OMUX (white multiplexers in Fig. 4).We call these Ordinary Multiplexers (OMUXs), to be distinguishable from CMUXs. Control signals for these multiplexers stem from the control unit. OMUXs at the inputs of virtual FUs provide the inputs of the CMUXs at the inputs of physical FUs (shaded multiplexers in Fig. 3). These CMUXs choose the proper input of the physical FUs based on the role assigned to them. The control signal for this multiplexer stems from the CFFs embedded in the data path (shaded rectangle in Fig. 3). The contents of CFFs should be programmed based on the active RFPV. Similar modifications are introduced at the outputs of the FUs. For each virtual FU, there is an output CMUX (shaded multiplexers in Fig. 4) whose inputs are connected to the outputs of associated physical FUs. These CMUXs will choose the proper output for the virtual FU based on the role assignments The OMUXs at the inputs of the registers will then select among the outputs of these CMUXs.

We demonstrate the concept using a simplified example of a circuit with 3 multipliers. The control unit has to consider 3 cases: (1) 3 non-faulty multipliers, (2) 2 non-faulty multipliers and (3) 1 non-faulty multiplier. In each of the three situations, the control unit has to generate control signals for the following multipliers:

1. M'_1, M'_2 and M'_3 for case (1)
2. M'_1 and M'_2 for case (2)
3. M'_1 for case (3)

In each case, the control unit issues the control signals for those virtual multipliers in use. For example, in case 2, the control unit may decide that in the 3rd clock cycle, multiplier M'_2 should receive its left and right operands from registers R_2 and R_3 and its result should be stored in register R_8. The rest of the work is the responsibility of the CMUXs and CFFS in the data path. They should get the outputs of registers R_2 and R_3 to the inputs of the physical multiplier that is playing the role of M'_2 and get the output of that multiplier to the input of register R_8.

Based on the active RFPV, each non-faulty physical multiplier should play the role assigned to one virtual multiplier. For example, physical multiplier M_2, should be able to play the following roles:

1. In situation (1), it should be able to play M'_2.
2. In situation (2), either it is faulty or it should play the role of M'_1 (if M_1 is faulty) or the role of M'_2 (if M_3 is faulty).
3. In situation (3), either it is faulty or it should play M'_1.

Thus, this multiplier should be able to play one of the two roles $\{M'_1, M'_2\}$. Selecting proper inputs for each role could be done using one CMUX with 2 inputs, and 1 select line (shaded multiplexer in Fig. 3). In Fig. 3, the multiplexers in the upper row are OMUXs and the CMUX in the lower row selects the proper input based on the configured role of the physical multiplier M_2. On the other hand, since the role of virtual multiplier M'_1 might be assigned to any of the physical multipliers M_1, M_2 or M_3, we need a CMUX with 3 inputs and 2 control lines to select the output of M'_1. This is the responsibility of the shaded multiplexer in Fig. 4.

Fig. 6 shows the Virtual Binding algorithm. The DFG of the design and the maximum allowable number of FUs are inputs of the algorithm. The algorithm, one by one, considers all possible RFCVs and generates an scheduling for each RFCV. Here again, any particular scheduling algorithm could be used. Because the algorithm binds the DFG to virtual FUs, for each RFCV, there is one virtual RFPV, denoted by V'_{fs}. In this virtual RFPV, the first non-faulty *i*-type virtual FU will always be $FU'_{i,1}$, the next one will be $FU'_{i,2}$ and the last will be FU'_{i,n_i}, where n_i comes from the C_{fs}. The scheduled DFG will be bound to virtual FUs in the V'_{fs}. Next, the control unit will be augmented to use the resulted binding when the related RFCV is active (Note that, here, the result of the binding depends only on the RFCV and not a particular RFPV. Thus, in Virtual Binding, it is the active RFCV that determines the mode (or behavior) of the control unit, in contrast with the case of Static Binding method where the active RFPV determines the mode). In the next step, the required data transfer paths will be added to the data path. After, considering all the possible RFCVs, CMUXs and CFFs should be added to the data path. For each physical FU, the input CMUXs will be connected to the OMUXs of the associated virtual FUs. Also, for each virtual FU, the output CMUXs will be connected to the outputs of associated physical FU. For each added CMUX, CFFs should be inserted to hold the control values of the CMUXs.

To configure the circuits synthesized with Virtual Binding method, one should first diagnose the faulty units. After detecting the current RFPV (and thus the current RFCV), the control unit should be programmed according to the current RFCV and the CFFs in the data path should be programmed according to current RFPV. It is interesting to note that the existence of the CFFs and CMUXs in the data path greatly simplifies the diagnosis process of such circuits. For test and diagnosis purposes, one can efficiently use these elements by putting the CFFs in the scan chain. More details, however, are out of the scope of this paper and will not be discussed.

This technique greatly reduces the number of different situations to be considered in the control unit, and hence, reduces the area and delay of the control unit in comparison to the Static Binding method. Also, according to the experimental results, this technique tends to result in a smaller data path than that of the Static Binding. This can be explained as follows: In the Static Binding method, the multiplier M_2 of the previous example should be able to play the operations originally bound to M_1 and M_3 because those multipliers might go faulty. This means that M_2 should receive all the original inputs of M_1 and M_3 as its inputs, enlarging the input multiplexers of M_2. But in Virtual Binding method, each virtual multiplier has its own input multiplexers and the physical multipliers only use CMUXs to choose among the possible roles that they would play. This will result in much smaller multiplexers in the resulting data path, hence decreasing its size.

III. Experimental Results

We have applied the proposed techniques to nine standard benchmark circuits, as described in Table I. For each circuit, the maximum number of FUs of each class is given. We have also reported the number of clock cycles in the schedule of the original, non-degradable circuit. This number provides a simple intuition into the complexity of the associated control unit. Also, the number of possible RFPVs is given. This number has been calculated using Equation -4-.

TABLE I. Benchmark circuits

Circuit	# of FUs			# of Scheduling Steps	# of Possible RFPVs
	ALU	Mult	Div		
DCT	4	4	0	5	225
CASCADE	1	3	0	4	7
DIFF_EQ	1	4	0	7	15
OVEN_CONT	3	1	1	6	7
PAOULIN	1	4	0	5	15
POLY_EVAL	2	3	0	5	21
REAL	2	2	0	5	9
TSENG	3	1	0	5	7
PAR_IIR_4	2	4	0	7	45

The synthesis results of the degradable circuits are reported in TABLE II and TABLE III. TABLE II shows the normalized area of the degradable counterparts of the benchmark circuits. The values were normalized to the area of the original, non-degradable circuit. TABLE III presents a similar data regarding the delay of the degradable circuits. The area and delay values have been separately reported for control unit and data path and steering logic (*i.e.*, multiplexers and configuration flip flops) for the two methods. Also the last column (titled 'VB/SB %') shows the relation between the results of the two methods. Smaller numbers indicate better performance of the Virtual Binding method as compared to the Static Binding. As shown, the Virtual Binding method slightly reduces the data path area, greatly shrinks the control unit (especially for complex circuits like DCT) and increases the data path delay with regard to the Static Binding method. However, according to data in TABLE III, the data path delay overhead is less than 5% compared to Static Binding method and less than 15% compared to the original, non-degradable circuit while the area reduction in control unit, compared to Static Binding method, could be as high as 76% (in case of DCT). Also, as shown in TABLE II, Virtual Binding does not increase the data path area of the circuit very much compared to the original circuit, and reasonably increases the control unit area regarding the large number of RFPVs that are handled.

IV. Conclusions

In this paper, we proposed a new technique for high level synthesis of degradable ASICs. This technique is capable of repairing ASICs when there are multiple faulty units. In fact, it is capable of repairing ASICs even when there is only one non-faulty instance of each hardware resource class, compared to the previous techniques that can handle at most one faulty functional unit in each class. Because the technique does not use spare modules, reconfigured circuits will have lower performance due to the decreased number of available functional units. This is a common phenomenon in all the techniques not using spare modules to replace the faulty ones.

We proposed the Virtual Binding method that allows for smaller control units, and slightly smaller data paths, than those obtained through the more straight forward Static Binding method. For circuits with a relatively high number of functional units, and thus many possible RFPVs like DCT and 4th order parallel IIR filter, Virtual Binding results in a much smaller control unit.

The efficiency of the proposed technique has been examined using a number of standard benchmark circuits. The experimental results show that area and delay overhead of the circuits obtained through Virtual Binding method are in acceptable ranges compared to original, non-degradable circuit. For benchmark circuits, delay and area overhead of the degradable circuits are no more than 15% and 36%, respectively.

REFERENCES

[1] L.M. Guerra, M.M. Potkonjak, and J.M. Rabaey, "High-Level Synthesis Techniques For Efficient Built-In-Self-Repair", The IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems, 1993, pp. 41-48.

[2] L.M. Guerra, M. Potkonjak, and J.M. Rabaey, "Behavioral-Level Synthesis of Heterogeneous BISR Reconfigurable ASIC's," IEEE Transactions on VLSI Systems, vol. 6, no. 1, pp. 158-167, Mar. 1998.

[3] R. Naidu, and S. Mahapatra, "Fault Tolerance in N-MOS Random Access Memories with Dynamic Redundancy Methods", Microelectronics and Reliability, vol. 28, no. 2, pp. 193-200,1988.

[4] R. Negrini, M.G. Sami, and R. Stefanelli, "Fault Tolerance Through Reconfiguration in VLSI and WSI Arrays", MIT Press, 1989.

[5] G. De Micheli, "Synthesis and Optimization of Digital Circuits", McGraw-Hill, Hightstown, NJ, 1994.

[6] G. Buonanno, M. Pugassi, and M.G. Sami, "A high-level synthesis approach to design of fault-tolerant systems" in Proc. VTS'97, pp. 356-361, 1997.

[7] B. Iyer, R. Karri, and I. Koren. "Phantom Redundancy: A High-Level Synthesis Approach for Manufacturability", In Proc. ICCAD, pp. s-661, Nov. 1995.

[8] Wah Chan, and A. Orailoglu, "High-level synthesis of gracefully degradable ASICs," in Proc. ED&TC, pp. 50-54, 1996.

[9] A. Orailoglu, "Microarchitectural Synthesis of Gracefully Degradable, Dynamically Reconfigurable ASICs," in Proc. of ICCD'96, pp.112-117, Oct. 1996.

[10] S. Mitra, W.-J Huang, N.R. Saxena, S.-Y. Yu, and E.J. McCluskey, "Reconfigurable Architecture For Autonomous Self-Repair", IEEE Design & Test of Computers, vol. 21, no. 3, pp. 228- 240, May-Jun. 2004.

[11] P. Paulin and J. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASICs", IEEE Transactions on CAD/ICAS, Vol. CAD-8, No. 6, pp. 661-679, July 1989.

TABLE II. NORMALIZED AREA OF DEGRADABLE CIRCUITS

Circuit	Static Binding			Virtual Binding			VB/SB%		
	Control	DPath	Steering	Control	DPath	Steering	Control	DPath	Steering
DCT	10.85	1.40	7.44	2.58	1.36	6.73	23.81	97.06	90.35
CASCADE	1.61	1.14	3.90	1.31	1.10	3.20	81.28	97.02	82.06
DIFF_EQ	2.30	1.23	4.81	1.42	1.15	3.52	61.64	93.62	73.20
OVEN_CONT	1.42	1.22	4.99	1.17	1.15	3.61	82.38	94.13	72.36
PAOULIN	2.10	1.21	6.97	1.30	1.14	3.55	61.82	94.38	50.92
POLY_EVAL	1.69	1.20	5.54	1.19	1.12	3.88	70.27	94.10	69.92
REAL	2.05	1.26	4.50	1.45	1.18	3.55	70.57	93.52	78.92
TSENG	1.59	1.26	3.20	1.19	1.18	2.48	74.88	93.12	77.48
PAR_IIR_4	5.40	1.36	4.59	2.38	1.29	3.89	44.02	94.90	84.72

TABLE III. NORMALIZED DELAY OF DEGRADABLE CIRCUITS

Circuit	Static Binding			Virtual Binding			VB/SB %		
	Control	DPath	Steering	Control	DPath	Steering	Control	DPath	Steering
DCT	1.20	1.10	2.03	1.04	1.15	2.74	87.07	105.00	134.89
CASCADE	1.04	1.08	5.27	1.08	1.14	6.84	104.35	105.38	129.87
DIFF_EQ	1.00	1.11	1.33	1.00	1.15	1.61	100.23	104.15	121.58
OVEN_CONT	1.02	1.04	1.30	1.02	1.03	1.66	100.00	101.5	127.89
PAOULIN	1.07	1.05	2.36	1.03	1.09	2.95	96.10	104.27	125.22
POLY_EVAL	1.05	1.06	5.72	1.00	1.08	6.47	95.73	101.98	113.05
REAL	1.04	1.07	1.55	1.02	1.08	1.90	98.30	101.35	122.53
TSENG	1.04	1.01	1.28	1.06	1.00	1.57	101.72	101.5	123.00
PAR_IIR_4	0.94	1.10	1.38	0.98	1.12	1.76	103.59	101.78	127.46