EQUIVALENCE CLASSES OF CLONE CIRCUITS FOR PHYSICAL-DESIGN BENCHMARKING

Michael D. Hutton
Altera Corporation
101 Innovation Drive.
San Jose CA 95134
mhutton@altera.com

Jonathan Rose
University of Toronto
Department of Electrical and Computer Engineering
10 King's College Rd.
Toronto, Ontario Canada M5S 3G4
jayar@eecg.toronto.edu

Abstract

To provide a better understanding of physical design algorithms and the underlying circuit architecture they are targeting, we need to exercise the algorithms and architectures with many benchmark circuits. The lack of existing benchmarks makes us consider the automatic generation of netlists.

In this paper, we formally define the equivalence class of circuit "clones" of a given seed circuits, based upon physical properties of the seed circuit's netlist graph. Using these equivalence classes of circuits, a given seed circuit can be used to generate many similar circuits. A more finely grained statistical analysis of algorithm behaviour can then be obtained from using the multiple benchmarks than would be available from using the seed benchmark alone.

1. INTRODUCTION

There are two primary motivations for the automatic generation of benchmark netlists.

The first motivation is that there are simply not enough benchmark circuits available, or not enough at a given density range. With the exception of the recent efforts such as the ISPD'98 partitioning benchmarks from Charles Alpert at IBM [1], the major drawback of public-domain benchmarks is that they lag industry by multiple orders of magnitude in numbers of gates. ASIC designs can use over 5M gates, next-generation programmable logic designs up to 1M gates. Most public domain benchmarks for partitioning, placement, routing and programmable-logic architecture design (e.g. [2]) range from 100 to 2000 4-input LUTs, or less than 10K gates. Neither logic nor algorithms scale well enough for experimentation at small density to be directly applicable to larger density problems. Even within industry, there is a fundamental problem in that new algorithms and architectures must be defined and evaluated a full design-cycle before there are any true benchmarks at the next-generation density range.

The second motivation for automatic generation of benchmarks is to improve the statistical validity of tests that we can perform on a limited number of circuits. Since algorithms for NP-hard problems are always heuristic, they can exhibit varied performance under only slightly different circumstances. Thus, it is advantageous for us to have multiple circuits which are similar enough to group in the same class, yet different enough to isolate the experiment from minor variations due to the particular heuristic bias of a given algorithm. This latter issue is the primary focus of the current ISCAS special session [3].

There have been several previous attempts at generating automatic benchmark circuits. Hutton *et. al.* [4,5,6,7] have described methods and a tool for generating circuits subject to a number of statistical properties about the physical nature of the circuit. This can be done either by extracting the parameterization from a seed circuit, or by random selection of the properties from defaults extrapolated from a collection of benchmarks. Darnauer and Dai [8] generated circuits by recursive decomposition, using Rent's Rule [9,10] as a basic paradigm. In both these approaches, the user is able to generate circuits of virtually any size, without an existing seed circuit, but the correspondence of the circuits to reality diverges as the size grows. Iwama and Hino [11], Kapur *et. al.* [12], Ghosh *et. al.* [13,14] and Harlow and Brglez [15] have addressed the "mutation" of initial circuits using various transformation rules in order to effect a circuit with similar overall structure but differing local connectivity.

One of the reasons that heuristic algorithms work well in practice is that "real netlist graphs" exhibit fundamental restrictions on simple graph-theoretic properties such as fanout, distance on paths and the location of sequential elements (flip-flops) in the netlist. In order to do valid experimentation it is thus important to capture these properties in the circuits we use for benchmarks.

In this paper, we formally define equivalence classes of circuits based upon purely physical netlist characteristics: fanout of nodes, path-length, the relative location of nodes on paths through the design, and the length of edges induced by these locations. When an equivalence class is based on the extracted characteristics of an existing seed benchmark circuit, we call the artificially generated members of the same class "clone-circuits" of the original. The work reported on in this paper formalizes and builds upon previous work [4,5,6,7] in which we created two tools: CIRC, a tool to analyze netlists and extract the properties of interest, and GEN, a tool to generate a new circuit given a set of properties. In the previous work we used pairwise comparison of clone circuits to validate the generation tool, but did not make formal definitions of clones or analyze the behaviour of equivalence classes induced by clones.

In a companion paper [16], we will outline a methodology for the evaluation of competing heuristic algorithms for physical design using clone circuits. We will apply the methodology to several different problems and comment on practical experience doing so for these and other issues requiring benchmarks, such as the design of programmable logic architectures. In doing so we will make conclusions as to where clone circuits can and cannot be used effectively.

2. EQUIVALENCE CLASSES

The purpose of the benchmarks we are generating is to analyze the performance of physical design algorithms and the interconnection networks of programmable-logic architectures. Thus, our circuit parameterization is based on purely physical graph-theoretic properties of the netlist: node types, connectivity and path-lengths, as opposed to functional properties such as "this is a multiplier."

We will assume that all netlists under discussion are synchronous designs with no bidirectional pins. Nodes in the netlist N consist of primary inputs (PI), primary outputs (PO), 4-input lookup tables (LUT) and D-type flip-flops (DFF) gated by a single global clock called 'clock' and no other secondary signals. The fanin and fanout of a node have their standard meaning, and max_fanout(N) is the maximum fanout value for any node x in N. There are no combinational cycles allowed.

To describe a netlist, N, we need to formalize a number of terms.

Definition 2.1: Define the unit *delay* of a node x: If x is a PI or DFF, then delay(x) is 0. Otherwise delay(x) is 1 + the maximum unit delay of any fanin of x. Define delay(N) as the maximum, over all nodes x in N, of delay(x).

Definition 2.2: Define the sequential *level* of a node x. If x is a PI, then level(x) is 0. If x is a DFF, then level(x) is 1+ the level of the D-input of x. Otherwise level(x) is the minimum level over all fanins of x. Define levels(N) as 1 + the maximum, over all nodes x in N, of level(x). Define *level i* of N as the subgraph N_i induced by the set of nodes in N which are at level i.

Definition 2.3: A netlist N is *combinational* if it contains no DFF nodes, and *sequential* otherwise. If N is combinational it must have exactly 1 level, and all nodes x satisfy level(x) = 0. Otherwise N has at least 2 levels, and at least one node at each level.

Under the restrictions mentioned previously (no combinational cycles or bidirectional I/Os and a single global clock), both level(x) and delay(x) are well-defined.

Definition 2.4: The *shape* function of a combinational netlist N is defined as an integer vector shape[d], d = 0..delay(N), where shape[d] is the number of nodes in N which have delay d.

Definition 2.5: Given a directed edge e=(x,y) in a netlist N, define define *length*(e) = delay(y) – delay(x). If level(y) < level(x) then e is a *back-edge*. If level(y) = level(x) then delay(y) > delay(x) and e is a *forward-edge*. Otherwise e is a *FF-edge*, and we must have delay(y) = 0, level(y) = level(x) + 1, and x is a DFF node. There are no other cases possible under the definitions of delay and level.

Definition 2.6: The *edges* function of a netlist N is defined as an integer vector edges[d], d = 0..delay(N), where edges[d] is the number of edges in N of length d.

Definition 2.7: The fanout function of a netlist N is defined as an integer vector fanout[f], f = 0..max_fanout(N), where fanout[f] is the number of nodes x of N with fanout f.

We can now outline a mechanism to decompose or partition a netlist into two or more parts. Given N and a bipartition X and Y of the nodes of N, create two graphs N_x and N_y induced by the partition. For every edge e=(x,y) where x is in N_x and y is in N_y, create a new primary input x′ in Ny for x, and a new primary output y′ in N_x for y (and similarly for edges from Y to X). The netlist graphs X and Y are now disjoint, yet by identifying or gluing the appropriate nodes {x,x′} and {y,y′} together we can re-create N.

Definition 2.8: Under the process just described for an edge e=(x,y), define additional nodes x′ to be a *ghost input* (GI) in N_y and y′ to be a *ghost output* (GO) in N_x. Define delay(x′) to be that of delay(x) and delay(y′) to be that of delay(y), supplementing the previous definitions with that of the new node-types GI and GO. Along with primary output nodes (PO), we can infer new shape functions POshape[d], GOshape[d] and GIishape[d] as we did for the delay-based shape function on the appropriate subset of nodes.

One obvious decomposition of a graph is into levels, whereby each flip-flop or back-edge induces a ghost input in one level and a ghost output in another. It is this usage that we will use to define the complete signature of a sequential netlist. We will call this process the *sequential decomposition* of a netlist N into its level-netlists N_0, N_1, etc.

For a netlist, we use the scalar parameters nPI, nPO, etc. to denote the number of nodes of the corresponding node-type.

Definition 2.9: The *signature* of a level-netlist N_i is composed of {i, n, nPI, nPO, nLOG, nDFF, nPO, nGI, nGO, delay(N_i), max_fanout(N_i), shape[], edges[], fanouts[], POshape[] GOshape[], and GIshape[] }. The signature of a sequential netlist N is defined by the collective signatures of its sequential decomposition. For an exact specification the scalar parameters are redundant given the vector parameters in the signature but are part of the signature for clarity.

Given the concept of a signature of a netlist, we can now formally define equivalence classes of netlists.

Definition 2.10. Two netlists are *equivalent* if they have the same signature. Given the set of all netlists of any size, we can then induce a mathematical equivalence class to properly partition all netlists into equivalence classes under signatures.

Definition 2.11. Given a set of circuits generated to have the same signature as a given input circuit, we refer to the original circuit as the *seed* circuit, and the other members of the equivalence class as *clone* circuits.

3. GENERATING CLONE CIRCUITS

In previous work we introduced an algorithm for generating a circuit from a set of restrictions, and implemented this algorithm in a tool called GEN. A companion tool, CIRC, can be used to extract the signature from an input netlist. An example of a circuit signature, produced by CIRC, is shown in Figure 1. The two sub-circuits L_0 and L_1 represent the sequential decomposition of the input circuit.

```
/* CIRC 3.2, compiled Jan 6,1999. */
X = { name="example"; nGI=0; nGO=0;
  L1 = (@.comb_circ) { exact=1;
    name="L1"; n=177; kin=4;
    nPI=177; nLatch=0; level=1; delay=0;
    nBot=177; shape=(177,0); nGI=0;
    GIshape=(0,0); nGO=212; GOshape=(212,0);
    nPO=0l POshape=(0,0); nEdges=0; edges=(0,0);
    outs=(177,0); max_out=0; nZeros=177;
  };
  L0 = (@.comb_circ) { exact=1;
    name="L0"; n=998; kin=4; nPI=123;
    nLatch=177; level=0; delay=92; nBot=1;
    shape=(123,36,18,10,5,2,3,2,2,3,9,7,
      5,4,4,6,8,4,5,4,7,6,6,6,7,8,9,10,6,8,
      6,10,10,8,9,9,15,19,21,24,17,15,13,8,
      11,29,41,34,30,29,30,35,34,30,22,14,13,
      9,10,9,5,4,4,3,2,2,1,1,1,2,1,1,1,2,3,4,
      3,2,6,7,5,9,8,5,4,3,3,2,2,2,1,1,1);
    nGI=212; GIshape=(3,4,2,4,4,0,2,2,1,2,3,2,
      2,1,1,2,1,3,1,0,6,11,6,7,9,9,6,8,8,6,7,8,
      5,10,2,5,5,2,2,4,4,7,6,0,2,2,3,1,1,0,0,0,
      0,0,1,0,1,0,0,2,2,2,1,1,2,1,2,0,1,0,1,0,0,
      0,2,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
    nGO=177; GOshape=(0,0,2,3,0,0,1,0,0,1,4,0,
      0,0,0,0,0,0,1,0,0,0,1,0,0,3,1,2,2,1,0,
      3,1,1,3,3,10,10,7,4,1,0,3,4,10,13,5,8,5,
      7,16,11,5,3,2,3,0,1,1,0,1,0,0,0,1,0,0,0,
      1,1,0,0,0,0,1,1,2,3,2,2,0,0,0,0,0,0,0,0,
      0,0,0,0);
    nPO=58; POshape=(0,2,8,0,0,0,0,0,0,0,2,2,2,
      0,1,1,3,0,2,1,0,0,0,0,2,0,2,1,1,0,0,1,1,0,
      0,0,0,0,1,0,0,0,0,0,2,1,0,0,0,0,0,1,2,1,0,0,
      0,1,0,1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,2,0,
      0,0,0,0,0,1,0,7,0,0,0,1,0,0,1,0,0,1);
    nEdges=2380; edges=(0,1013,220,138,96,74,70,
      69,43,33,36,32,46,16,20,10,24,22,18,14,8,
      8,14,7,7,9,11,11,7,7,17,15,13,3,11,4,9,
      14,11,4,7,8,9,10,7,4,3,5,5,7,9,12,2,2,2,
      4,3,14,11,5,5,4,1,1,0,0,0,0,1,0,0,1,1,4,2,
      4,2,1,2,3,2,14,14,2,0,1,0,0,1,0,1,0,0);
    outs=(87,505,145,64,56,32,32,25,10,16,4,4,4,
      4,1,2,1,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1);
    max_out=69; nZeros=87;
  };
  glue=(L0, L1);
};
output(circuit(X));
```

Figure 1. Example GEN clone signature extracted from a 2-level circuit from Altera Corp.

The syntax of the specification language is more complicated than might be expected because the tool is able to deal with much more general specifications in the case where the signature is produced manually. For example, the shape function can be specified as (10,100,10) even though the delay is 25; the tool will heuristically expand the shape to its full length. Also, the tool has default distribution functions for the different parameters in the case where the parameterization is under-specified.

GEN signatures do not have to come with an exact specification. The only required parameter is the number of nodes in the netlist. However, a minimal realistic specification, such as shown in Figure 2, would usually provide more. Here we request a 2000 node sequential circuit with 128 inputs and 85 outputs. The resulting circuit is a 2-level sequential circuit represented by a netlist graph on 1000 nodes (128 PI and 1872 LUT), with 85 PO and 402 DFF (not counted in "n"). We note that one could then extract a signature from the circuit X and generate a set of equivalent circuits to X without ever having an actual seed circuit.

```
X = (@.fsm_circ) {
   name="x"; nPI=128; nPO=85;
   n=1000; nLatch=300; delay=25;
};
output(circuit(X));
```

Figure 2. Example of a manually specified GEN signature.

Because we want to generate multiple circuits from a given specification the parts of the algorithm which choose the actual connectivity and which fill-in missing parameters are randomized. GEN can be parameterized with a command-line option for the random seed, which is then associated with that circuit. If no seed is specified, the random-number generator will use the clock and be different on each invocation.

2.1 Variation from the input specification.

The extraction of a signature from a netlist is deterministic. However, the algorithm for generating a new circuit from a given signature is heuristic and requires randomization in order to generate a different equivalent circuit on each invocation. As a result, GEN is allowed to output circuits which are "close" to the input signature in order to determine a feasible solution.

"Close" means that GEN is permitted to introduce small variations in the signature as part of its heuristics. If tolerances in the software are exceeded GEN will fail to produce a circuit rather than producing a circuit which fails to meet the signature specified. In the 800 clones generated for the experiment, this occurred 8 times. The number of edges was modified in almost every circuit, but on average by less than 1%. The most significant changes are in moving nodes between adjacent levels or promoting a fanout-5 node to a fanout-6 node correspondingly reducing a different fanout-4 node to a fanout-3 node to conserve the overall number of edges and allow connectivity to work properly. This, again, occurs in almost every circuit, but analysis of the log-files produced by GEN show that fewer than 10% of nodes are ever effected which is reasonable.

In the case of user-specified circuits, it is more difficult for the tool to effectively come up with a valid circuit. This is not a

problem for simple specifications such as the one in Figure 2, but as the specification is further and further constrained there is more onus on the user to choose parameters which are compatible. It is possible to generate glued circuits with multiple different components manually, but the user has to match the GI and GO distributions carefully to ensure that the circuit is "glueable". In future versions of GEN we hope to have a better solution to the issue of conflicting parameters so that we can do multi-level and hierarchical gluing without help from the user.

4. CONCLUSIONS

In this paper we have formally defined equivalence classes of clone circuits, and shown how the tools CIRC and GEN can be used to generate circuits which are equivalent to a given seed circuit.

Using the cloning mechanism we can generate a large number of benchmark circuits from a given seed circuit and effect better and more statistically significant tests than would be possible with just one benchmark. Because GEN can also create a circuit from scratch, this mechanism is not limited to the density range of current benchmark circuits. However, we point out that the validity of circuits diminishes as the number of nodes increases – we will address this further in the companion paper when we discuss methodology and applications.

CIRC and GEN are available under public-domain license via the website http://www.eecg.toronto.edu/~jayar.

References

[1] C. Alpert, " The ISPD circuit benchmark suite," in *Proc. Intl. Symposium on Physical Design*, 1998.

[2] S. Yang, "Logic synthesis and optimization benchmarks," V3.0, Tech. Report, Microelectronics Center of North Carolina, Research Triangle Park, NC. See also NCSU CAD Benchmarking Laboratory at http://www.cbl.ncsu.edu.

[3] F. Brglez and R. Drechsler, "NP-hard problems, design of experiments, and the web: Context for ISCAS'99," in *Proc. 1999 Int. Symp. Circuits and Systems (ISCAS'99)*, Orlando, Florida, 1999.

[4] M. D. Hutton, 'Characterization and automatic generation of benchmark circuits," Ph.D. dissertation, University of Toronto, Canada, 1997.

[5] M. D. Hutton, J. P. Grossman, J. Rose, and D. G. Corneil, "Characterization and parameterized random generation of digital circuits," in *Proc. 33rd ACM/SIGDA Design Automation Conf. (DAC)*, June 1996, pp. 94-99.

[6] M. D. Hutton, J. P. Grossman, J. Rose, and D. G. Corneil, "Characterization and parameterized generation of synthetic combinational benchmark circuits," *IEEE Trans. CAD of Integrated Circuits and Systems*, vol. 17, no. 10, Oct, 1998, pp. 985-996.

[7] M. D. Hutton, J. Rose, and D. G. Corneil, "Generation of synthetic sequential benchmark circuits." *in Proc. 5th ACM/SIGDA Int. Symp. FPGAs, (FPGA'97)*, Feb. 1997, pp 149-155.

[8] J. Darnauer and W. Dai, "A method for generating random circuits and its application to routability measurement," in *Proc. 4th ACM/SIGDA Int. Symp. FPGAs (FPGA'96)*, Feb 1996, pp 66-72.

[9] W. E. Donath, "Placement and average interconnection lengths of computer logic," *IEEE Trans. Comput.*, vol. C-26, no. 4, pp 272-277, 1979.

[10] B. S. Landman and R. L. Russo, "On a pin versus block relationship for partitions of logic graphs," *IEEE Trans. Comput.*, vol C-20, no. 12, pp. 1469-1479, 1971.

[11] K. Iwama and K. Hino, "Random generation of test instances for logic optimizers," *in Proc. 31st Design Automation Conf. (DAC)*, 1994, pp. 430-434.

[12] N. Kapur, D. Ghosh and F. Brglez, "Towards a new benchmarking paradigm in EDA: analysis of equivalence class mutant circuit distributions," in *Proc. ACM International Symposium on Physical Design*, April, 1997.

[13] D. Ghosh, "Synthesis of equivalence class mutants and applications to benchmarking," Tech. Report available at http://www.cbl.ncsu.edu/publications.

[14] D. Ghosh, N. Kapur, J. Harlow and F. Brglez, "Synthesis of wiring-signature-invariant equivalence class circuit mutants and appications to benchmarking," in *Proc. Design Automation and Test in Europe (DATE)*, Feb. 1998, pp. 656-663.

[15] J. Harlow and F. Brglez, "Design of experiments for evaluation of BDD packages using controlled circuit mutations," in *Proc. IEEE/ACM International Workshop on Logic Synthesis (IWLS'98)*, June, 1998.

[16] M. D. Hutton and J. Rose, "Applications of clone circuits to issues in physical design", in *Proc. 1999 Int. Symp. Circuits and Systems (ISCAS'99)*, Orlando, Florida, 1999.