Flexibility Measurement of Domain-Specific Reconfigurable Hardware

Katherine Compton
Department of ECE
University of Wisconsin-Madison
Madison, WI 53706 USA
kati@engr.wisc.edu

Scott Hauck
Department of EE
University of Washington
Seattle, WA 98195 USA
hauck@ee.washington.edu

ABSTRACT
Traditional metrics used to compare hardware designs include area, performance, and power. However, these metrics do not form a complete evaluation of reconfigurable hardware. For these designs, flexibility is also a key issue, since it is the flexibility of reconfigurable hardware that allows it to implement a variety of circuits. Despite its importance, there is not yet an established method to measure flexibility. This paper explores the flexibility testing issue for domain-specific reconfigurable architectures. We discuss the concept of flexibility as it pertains to domain-specific architectures, and propose a flexibility testing technique involving synthetic circuit generation. This technique is then used to compare three different domain-specific architecture generation algorithms, demonstrating that the testing can in fact differentiate between architectures of differing levels of flexibility.

Categories and Subject Descriptors
B.6 [**Logic Design**]; B.6.3 [**Design Aids**]: Automatic synthesis; B.8.2 [**Performance Analysis and Design Aids**]

General Terms
Algorithms, Measurement, Design, Experimentation.

Keywords
Flexibility, Programmable Hardware, Reconfigurable Hardware

1. INTRODUCTION
The design of a reconfigurable architecture differs from the design of a conventional ASIC in a key aspect: flexibility. The quality of an ASIC is generally measured in terms of power, performance, and area. However, with reconfigurable hardware, flexibility is equally important, as it allows reconfigurable architectures to implement various circuits using a single set of hardware resources. In particular, if a reconfigurable subsystem is needed in a system-on-a-chip (SoC), the designer must choose the best design for the purpose at hand based on all comparison metrics, including flexibility.

FPGA'04, February 22–24, 2004, Monterey, California, USA

Unfortunately, thus far the flexibility metric has been largely ignored by the FPGA and reconfigurable computing community. FPGA devices are frequently measured by "gate count", which many feel is a nebulous assessment of the abilities of the device. The "gate count" metric is even less appropriate for domain-specific reconfigurable architectures. For these designs, flexibility is not simply a sum of the number of configuration points or generic logic structures, but instead the ability of a design to implement a particular type of circuit. Not all resources are equally useful. If a large number of resources are added which will not be used by any circuit of the domain, the flexibility of the architecture has not actually been increased. The useless resources only contribute to overhead.

For this reason, flexibility testing for domain-specific architectures should be based on the ability to implement circuits from the given domain. If all circuits from the domain are available in advance, a reconfigurable architecture can be created specifically to implement those circuits, and be completely flexible within the domain [1]. However, in many cases, the reconfigurable architecture will be created based on a few representative circuits, but be expected to implement additional circuits as new applications are found for the hardware, or changes to the original applications are made. The challenge is to measure the flexibility of the architectures when the complete domain is not yet known.

The process of creating an architecture based on one circuit set, and testing flexibility with another circuit set from the same domain, can be emulated through the use of a synthetic circuit generator. The known circuits of the domain can be profiled to determine a range of characteristics for the domain. Then, synthetic circuits can be created with characteristics selected randomly from within the range. The next section discusses issues involved with the use of synthetic circuits for flexibility testing, and proposes a flexibility testing technique. This is followed by an example of how the proposed flexibility testing technique can be used to test the flexibility of three different automatic reconfigurable architecture generation algorithms.

2. TESTING FLEXIBILITY
The synthetic circuits are created based on the real circuits of the application domain. The application circuits are profiled in order to measure key defining characteristics such as number of logic nodes, number of circuit I/Os, and complexity of the interconnections. The characteristics are then used to generate new circuits with structures similar to the original "parent" circuit or circuits. Using a set of circuit profiles, a description of an application domain can be created. The minimum, maximum,

mean, and standard deviation across the set of profiles are computed for each of the circuit characteristics. This provides a range of potential values for each characteristic. New profile information is generated by randomly choosing values within the given range according to a Gaussian distribution for each characteristic (using the mean and standard deviation to describe the function).

Randomness is frequently used during the circuit generation process to allow the creation of a wide variety of similar, but not identical, circuits. For the case where only a single circuit is used as the "parent" of one or more synthetic circuits, these synthetic circuits could be considered "clones" of the parent, as a generator generally attempts to achieve nearly the same characteristics. However, when a circuit is generated from a domain profile, its characteristics may not exactly match those of any one of the parents. In this case, the circuit is more of a "child" to the parent circuits. The domain provides a range of possibilities for each characteristic, obtained through the profiling of the parent circuits. The characteristics for the child are chosen randomly from that range, and the process is repeated for each characteristic used in circuit generation.

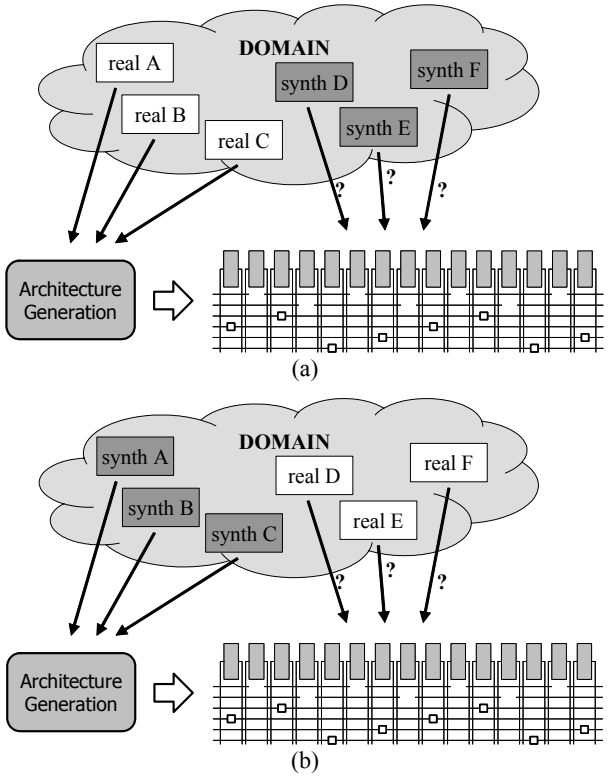

Figure 1: Two possibilities to test flexibility of an architecture generator using synthetic circuits. In each case, the architecture is created based on one set of circuits, and tested using a different set of circuits from the same domain. (a) Use the synthetic circuits for the final testing. (b) Use the real circuits for the final testing.

The flexibility of a reconfigurable architecture could be tested by determining the percentage of these synthetic circuits that can be successfully placed and routed onto the architecture, as per Figure 1a. However, this particular technique depends on the validity of synthetic circuits for the final test. While synthetic circuits might have similar characteristics to real circuits, they are not guaranteed to have "realistic" operation, and subtleties of the circuit structure not captured by the circuit profiling may be lost. Therefore, it is not guaranteed that an architecture that can implement synthetic circuits will be able to implement real circuits with similar characteristics.

However, when testing automatically generated architectures, the roles of the real and synthetic circuits can be reversed, as shown in Figure 1b. Architectures can be generated from the synthetic circuits, and tested using the real circuits. In this case, the synthetic circuits represent the known domain circuits, while the real circuits represent the unknown circuits of the domain that are similar, but not identical, to the known circuits. Using this order allows one to determine if an architecture created for one set of (synthetic) circuits is able to implement other real circuits from the domain.

3. EXAMPLE FLEXIBILITY TEST

The flexibility testing method proposed above was used to examine the relative flexibility of three different architecture generation algorithms [2]. These algorithms create customized reconfigurable architectures based on the needs of an input set of circuit netlists. The netlists were originally created for the RaPiD architecture [3], shown in Figure 2. For this testing, a circuit generator was developed to create the synthetic circuits used to generate the test architectures. The synthetic circuit generator will first be described, then the architecture generation methods used in the example flexibility testing will be summarized, and finally, the results of the testing will be given.

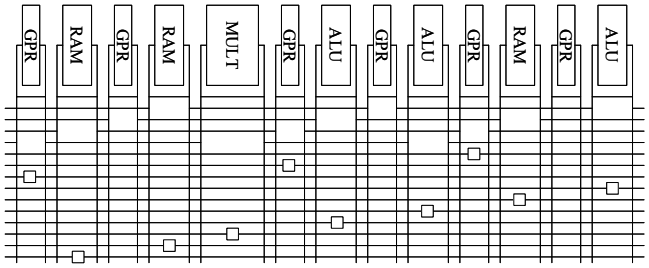

Figure 2: A single cell from the RaPiD architecture [3, 4].

3.1 Circuit Generator

Previous work in synthetic circuit generation for reconfigurable architectures generally focuses on the use of LUTs [5, 6, 7, 8]. However, the architectures generation algorithms that will be tested here are much more coarse-grained in nature, using word-width ALUs, multipliers, and other large logical units. Therefore, one of the existing circuit generation techniques [6] has been significantly modified here to compensate for heterogeneity, coarse granularity, and RaPiD circuit structure. The operation of the circuit generator involves first measuring a few key characteristics of a "parent" netlist or netlists, then generating different circuits (through the use of some degree of randomness) from these guidelines. The next few sections describe the process of profiling netlists, followed by the techniques used to generate the synthetic circuits.

3.1.1 Profiling RaPiD Netlists

Circuits are profiled in order to measure key defining characteristics. These characteristics are then used to generate new circuits with structures similar to the original parent circuit. RaPiD netlists currently form the source circuit material, and these netlists have a distinct high-level structure. RaPiD netlists are split into stages, where each stage operates in parallel, and communicates only with adjacent stages. Inter-stage communication is limited, as is the number of primary inputs and outputs. For the synthetic circuits to mimic RaPiD netlists, we must account for the stage and communication structures. Each netlist is converted to a directed graph, and measured for several characteristics, including:

- I/O requirements of the netlist
- Number of netlist stages
- Minimum/maximum connections between stages
- Minimum/maximum number of delay levels (logic levels) within the stages
- Number of logic nodes
- % of nodes of each type (ALU, multiplier, etc)
- Proportion of signals to nodes
- Proportion of signals that are back edges (source at higher delay level than sink)

Figure 3 illustrates a sample RaPiD netlist with 28 logic nodes, represented as a directed graph. Note that all multi-terminal signals are split into a set of 2-terminal signals for this profiling.

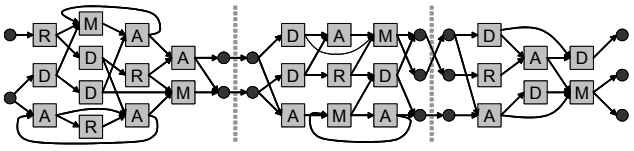

Figure 3: A directed graph of a sample RaPiD netlist of three stages to be profiled. The left stage has four delay levels, while the others each have three. Logic is represented as squares labeled by type (*R*AM, *A*LU, *M*ultiplier, *D*ata register). Signals are represented as arrows. I/O of the stages and overall circuit is represented by dark circles.

A new synthetic circuit profile can be created from a set of RaPiD netlist profiles representing an application domain. Each characteristic in the domain is represented by a Gaussian function controlled by the minimum, maximum, mean, and standard deviation of that characteristic across the different member circuits. The synthetic profile is created by choosing a random point on the Gaussian function for each circuit characteristic. This profile can then be used in the circuit creation step below.

3.1.2 Circuit Creation

The characteristics used to specify a synthetic circuit can be obtained through manual input, from the profile of a single netlist, or from a synthetic profile created from a domain. A graph of the synthetic circuit is created based on the desired characteristics of the profile. The goal of this generator is to create a circuit that, when it is profiled, will have characteristics that approximate the circuit characteristics of the specification. First the general structure, or "skeleton", of the circuit graph is created. Next the logic nodes are created, and then the edges are created to connect the logic nodes together to form a directed graph. Figure 4 demonstrates these steps for an example synthetic circuit created from the profile of Figure 3. Finally, the completed graph describing the circuit is converted into an actual netlist by assigning the signals on each node to actual ports of the logic unit the node represents, and the inter-stage I/O connections are propagated to connect units directly. Full details of this algorithm can be found elsewhere [9].

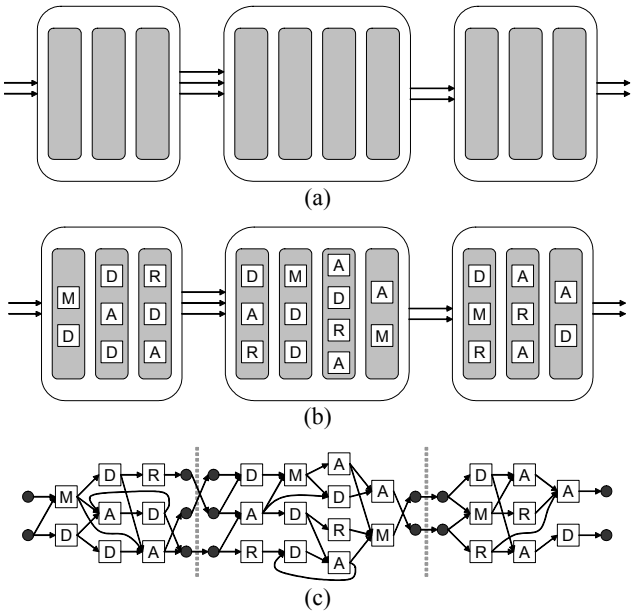

Figure 4: Steps in the creation of an example synthetic circuit graph. (a) The skeleton, including the number of stages (white bubbles) and delay levels within the stages (shaded bubbles). The connectivity between stages is also indicated by the arrows. (b) Logic nodes are then added to the skeleton. (c) Finally, the edges are added to connect the logic nodes.

3.1.3 Validation

The synthetic circuits are intended to mimic the structure of actual RaPiD netlists, without duplicating those circuits exactly. To test that this is the case, ten synthetic circuits were created for each of 26 actual RaPiD netlists. The synthetic circuits were then profiled with the same techniques used to profile the RaPiD netlists. The profiled values were normalized to the characteristics of the original real circuits, and averaged across the ten synthetic circuits. A comparison of the normalized characteristics of the generated circuits is given in Table 1. Here, cross-section refers to the signal cross-section when the circuit is placed along a 1D axis (the architecture generation algorithms, as discussed later, create 1D architectures).

Some characteristics, such as number of inputs and the number of stages, are identical to the parent circuits. The generated circuits are able to be constructed to match these characteristics exactly. On the other hand, characteristics such as percent back edges and stage I/O, are more difficult to control in the current method of circuit generation where the results depend significantly on random number generation. Future versions of the synthetic circuit generator could attempt to bring these characteristic values closer to those of the parent circuit. However, the data in this chart indicates that in general, the synthetic circuits have very similar characteristics to the original circuits. Additionally, most

standard deviations listed are above zero, which indicates that there is some variety in the characteristic values, as desired.

Table 1: A comparison of the characteristics of the generated synthetic circuits to those of the original circuits. All characteristics were normalized to the original netlists. The average values and standard deviations across ten synthetic circuits were computed for each parent circuit, and the average values across all 26 parent netlists are given.

Characteristics	Avg	Dev
# Inputs	1.00	0.00
# Outputs	1.00	0.00
# Instances	1.04	0.02
# Signals	1.06	0.03
Sig:Inst Ratio	1.02	0.03
# Stages	1.00	0.00
% Backedges	0.57	0.33
Stage I/O	0.72	0.06
# Delay Levels	1.05	0.05
Inst / Stage	1.04	0.02
Inst / Delay Level	0.92	0.06
Avg Node Fanin	0.84	0.02
Avg Node Fanout	0.92	0.02
Cross-section	1.00	0.14
% registers	1.05	0.03
% multipliers	0.98	0.01
% RAMs	0.99	0.01
% ALUs	0.99	0.01

3.2 Architecture Generation Algorithms

As stated previously, three different automatic reconfigurable architecture generation algorithms [2] from the Totem Project are used here in an example flexibility test. The algorithms all create architectures in the style of the RaPiD architecture [3], shown in Figure 2. RaPiD uses a series of coarse-grained logic units, such as 16-bit ALUs and 16x16 multipliers, aligned along a 1D axis. A 1D configurable routing structure allows the units to be connected into different circuits. The vertical lines of Figure 2 represent multiplexers and demultiplexers on every unit which allow it to connect to any of the routing tracks. The topmost routing tracks shown in the RaPiD cell diagram are local routing tracks, containing short wires. The bottom set of routing tracks from the figure are distance routing tracks, and the small boxes represent segmentation points, where two wires can be optionally connected together to form a much longer wire.

Like RaPiD, the Totem architectures use coarse-grained units along a 1D axis. Again, multiplexers and demultiplexers are used to connect the units to every track in the routing architecture. Totem architectures can vary in logic composition, as the resources are chosen based on the actual needs of an input circuit set. Likewise, the routing structure, in terms of track quantity and track types, is also dependent on the needs of the specification.

The first algorithm, Greedy Histogram (GH), focuses on customization and area savings over flexibility. Logic units are placed and wire lengths are chosen according to the needs of the input circuit set. However, due to this customization, these architectures may not be able to implement circuits differing significantly from the specification.

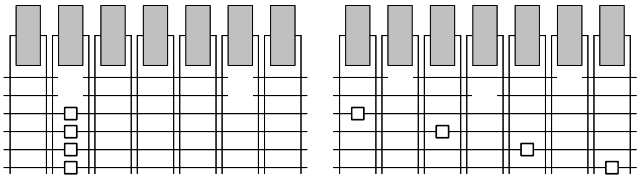

Figure 5: (a) An extreme example of a non-distributed routing architecture, and (b) a distributed routing architecture.

The second algorithm, Add Max Once (AMO), creates more generic structures able to implement a wider variety of circuits. Logic units are evenly spread throughout the array. Whereas GH may cluster all multipliers at the left edge of an architecture, AMO will spread them evenly. Routing resources are also distributed, as demonstrated by Figure 5b. The wire lengths (in terms of number of logic units spanned) are powers-of-2, such as length-4 wires, length-8 wires, etc. Also, segmented routing is used extensively, which can provide many options for a place and route tool implementing a new circuit. These architectures are, however, up to 12% larger than those created by GH for the real circuits from the domains examined in section 3.3.2.

The Add Min Loop (AML) attempts to use a more sophisticated algorithm than AMO to provide comparable flexibility within a slightly smaller area. This algorithm also uses the more generic style of AMO. However, AML chooses non-segmented local tracks whenever possible, as the segmentation points contribute to the total area of the architecture. The AML architectures are up to 8% larger than those created by GH for the real circuits in the domains of section 3.3.2. Each of these algorithms creates architectures at different points of the solution space. The relative goals of the three generation algorithms are depicted in Figure 6.

Figure 6: The relative goals, in terms of area and flexibility, of the three different architecture generation algorithms tested.

3.3 Results

The flexibility testing techniques described earlier were used here to test the flexibility of the three different routing architecture generation algorithms. First, flexibility measurement using circuits generated from single profiles will be discussed, and the generation algorithms will be compared using architectures generated from these circuits. Next, the results of domain flexibility testing, using circuits generated from a domain profile, are given.

3.3.1 Single Circuit Flexibility

First the flexibility of the different architecture generation methods was tested for single circuit generation. Ten circuits were created for each RaPiD netlist, and architectures were created using the three different architecture generation algorithms [2] given a single generated circuit as input. The generated circuits were forced to have the exact same logic resources as the parent circuit so that only the flexibility of the routing structures is tested. An attempt was then made to place and route the parent circuit onto each of the corresponding architectures generated from the synthetic circuits. This is

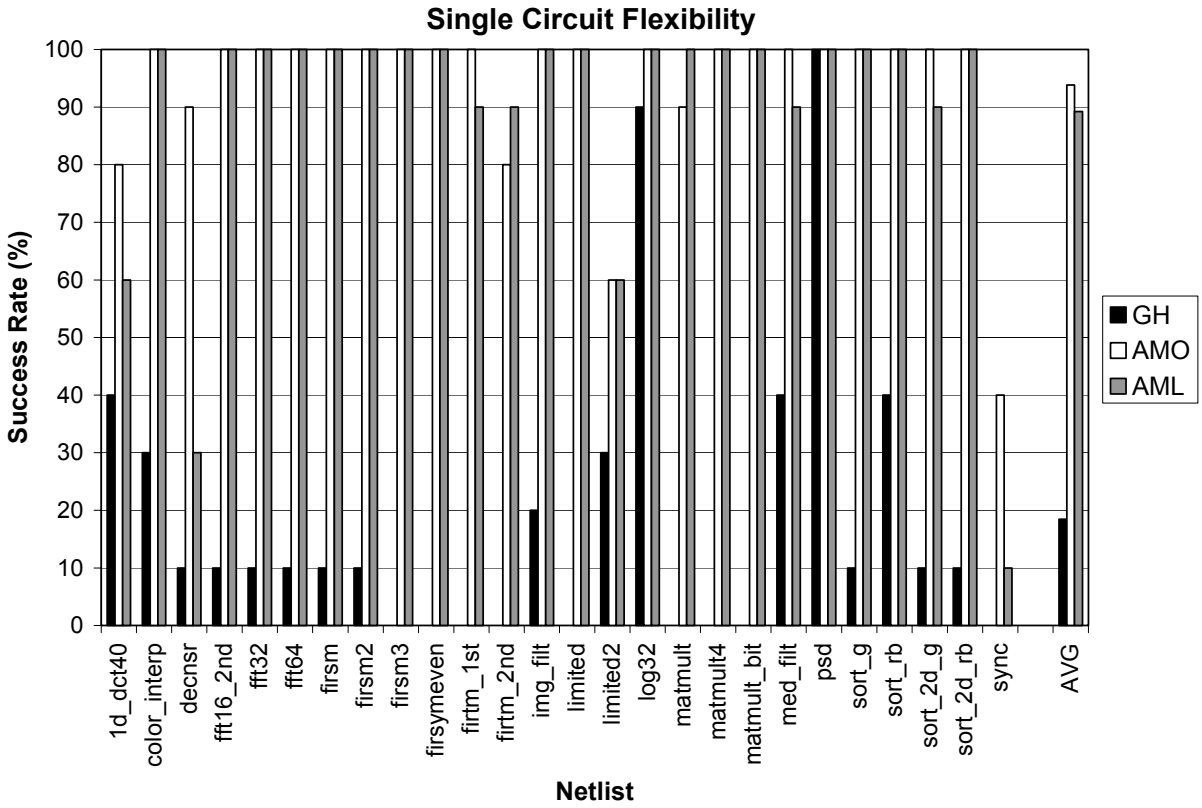

Figure 7: The success rate of implementing the original real parent circuit onto architectures created from a clone synthetic circuit. Results are given for each architecture generation method.

repeated for each parent circuit, for a total of 780 test cases—three methods used for each of ten circuits generated for each of twenty-six parent netlists.

The results, presented in Figure 7, highlight the flexibility differences of the architecture generation methods: Greedy Histogram (GH), Add Max Once (AMO), and Add Min Loop (AML). GH with synthetic circuits generally does not result in an architecture that is sufficiently flexible to implement the original circuit. This algorithm has only an 18% success rate. On the other hand, both the AMO and AML generation algorithms allow a significant percentage of the original circuits to be implemented. AMO has a 94% success rate, while AML results in an 89% success rate. These results reflect the intended flexibilities of the three algorithms as described earlier.

3.3.2 Domain Flexibility

In order to test the flexibility of the different routing generation algorithms for a given domain, first that domain must be profiled, as described earlier. The different original netlists whose profiles define the domain may vary considerably in their structure or resource requirements, and the synthetic circuits are generated randomly (with a Gaussian distribution) from the ranges of characteristic values. Generating an architecture from a range of profiles creates architectures less like any particular original netlist than if the architecture was created from a single profile, in an attempt to model possible unknown circuits of the domain.

However, it is then difficult to guarantee that the exact logic mix required by the original netlists will be represented by a set of synthetic netlists using domain circuit generation. For example, a domain may be specified by two netlists, netlist A and netlist B. Netlist A has 100 units, where 90% are ALUs and 10% are multipliers. Netlist B has 100 units, where 90% are multipliers and 10% are ALUs. A circuit is generated from the range specified by the profiles of the two netlists. This circuit contains 100 units, 50% of which are ALUs and 50% of which are multipliers. If a domain architecture is created from this one netlist alone, neither of the two original netlists could be implemented due to logic constraints using this particular method.

Instead, the domain tests allow for the specification of the minimum logic requirements of the real parent circuits. In this flow, the domains are profiled as before, but this time the minimum logic requirements are also profiled. Five synthetic circuits are generated for each domain. Architectures are created using the synthetic circuits for each of the domains. If the synthetic circuits do not provide sufficient logic resources of the necessary types (based on the domain minimums), the needed logic units are added to the architecture. The architectures are therefore guaranteed to have sufficient logic for the original netlists of their domain. The original netlists of the domain are then placed and routed onto the architectures. This process was repeated ten times for each domain, and the results are given in Figure 8.

The domain tests, like the single circuit tests, differentiate between the flexibility of the three different architecture generation methods. GH has the lowest flexibility, successfully routing 91.6% of netlists onto the architecture created for their

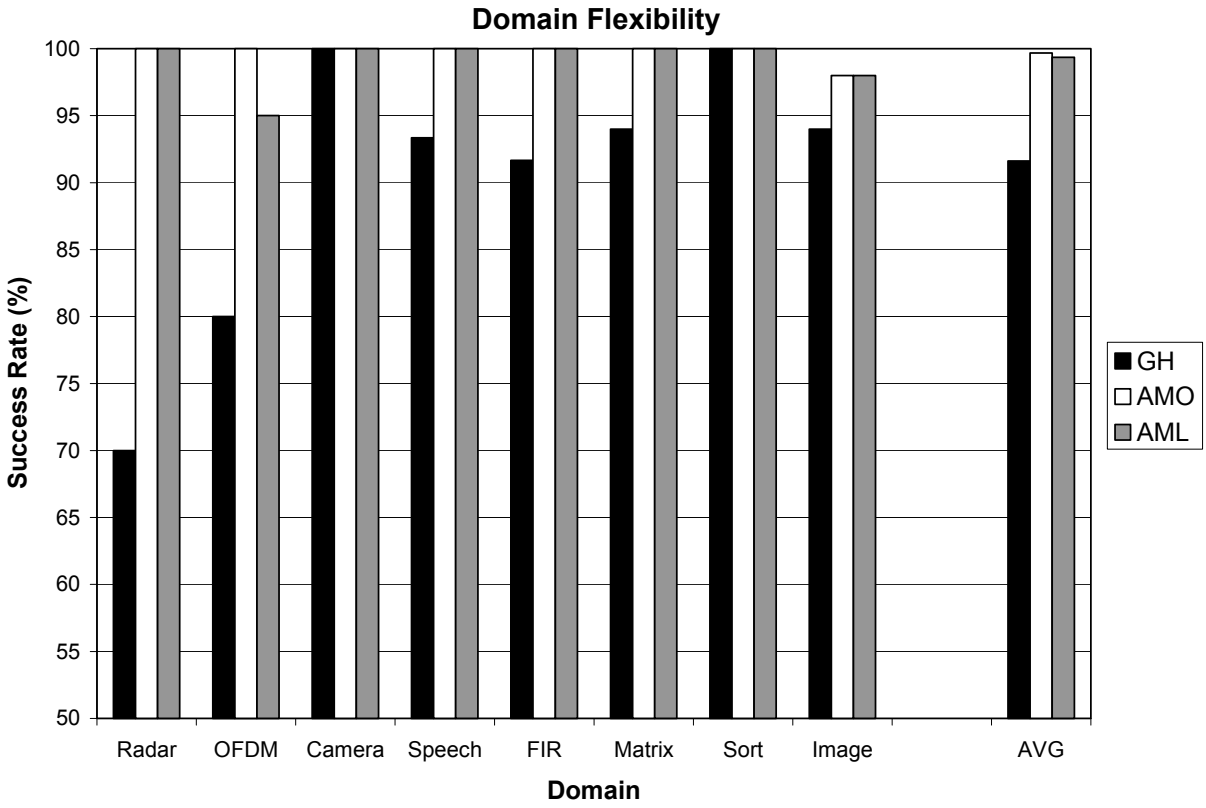

Figure 8: The success rate of implementing the real circuits of a domain onto architectures created from child synthetic domain circuits. Each domain contains between two and six real circuits. Results are given for each architecture generation method.

domain. AML had a mid-level flexibility, with a success rate of 99.4%. AMO has the highest flexibility, with 99.7% of the netlists successfully routed. These results again mirror the predicted flexibilities of the three algorithms.

4. Expanding Flexibility Work

The flexibility testing techniques presented here are able to test the relative flexibilities of architectures created using automatic architecture generators. However, flexibility testing is also important for non-generated (predefined) reconfigurable architectures. Here a designer would generate a large number of synthetic circuits for the targeted domain. The relative flexibility of different architectures within that domain would be determined by the percentage of synthetic circuits that can be successfully placed and routed onto each architecture. The architecture with the higher percentage would be considered more flexible within that domain. However, further effort should be made to verify that the synthetic circuit generator creates realistic circuits if this testing technique is to be credible.

The techniques presented here for flexibility measurement can also aid significantly in automatic reconfigurable architecture generation. In many cases, the reconfigurable logic will be generated before the target netlists have been finalized. Synthetic circuit generation can be used to fill this gap. As shown above, there are relatively few parameters that must be determined in order to specify a domain. These parameters can be estimated by the SoC designer, typically conservatively, to define the domain of applications to be supported. Then, synthetic circuits can be generated from these parameters, and fed to the architecture generator. The flexibility testing presented here indicates that architecture generation techniques such as AMO and AML have a very high probability of supporting new circuit designs, assuming the parameters can be accurately estimated. In future work, we will determine how accurately these parameters must be estimated, or conversely how conservative the estimates must be, to achieve a high likelihood of mapping success.

5. Summary

While area, performance, and power comparisons may be sufficient to compare traditional ASIC designs, these metrics fail to capture the complete merits of reconfigurable architectures. These architectures can also be compared on the basis of flexibility. However, no procedure to perform flexibility comparisons for reconfigurable hardware has yet been widely accepted. This paper proposed a technique to measure the relative flexibilities of domain-specific reconfigurable architecture generation algorithms. An example flexibility comparison was performed on three different architecture generation algorithms. The results reflected the intuitive flexibility expectations based on the different goals of the algorithms, demonstrating that the flexibility testing technique is able to determine relative flexibilities of generated architectures.

6. ACKNOWLEDGMENTS

This research was made possible by grants from NSF, NASA, and Motorola. Scott Hauck was also supported by a Sloan Research Fellowship.

7. REFERENCES

[1] K. Compton, S. Hauck, “Totem: Custom Reconfigurable Array Generation”, *IEEE Symposium on FPGAs for Custom Computing Machines*, 2001.

[2] K. Compton, A. Sharma, S. Phillips, S. Hauck, “Flexible Routing Architecture Generation for Domain-Specific Reconfigurable Subsystems”, *International Conference on Field-Programmable Logic and Applications*, pp. 59-68, 2002.

[3] D. C. Cronquist, P. Franklin, C. Fisher, M. Figueroa, C. Ebeling, “Architecture Design of Reconfigurable Pipelined Datapaths”, *Twentieth Anniversary Conference on Advanced Re-search in VLSI*, 1999.

[4] M. Scott, "The RaPiD Cell Structure", *Personal Communications*, 2001.

[5] J. Darnauer, W.W.-M. Dai, “A Method for Generating Random Circuits and its Application to Routability Measurement”, *ACM Symposium on Field Programmable Gate Arrays*, 1996.

[6] M. Hutton, J. Rose, J. Grossman, and D. Corneil, “Characterization and Parameterized Generation of Synthetic Combinational Benchmark Circuits”, *IEEE Transactions on CAD*, Vol. 17, No. 10, pp. 985-996, October 1998.

[7] S. Wilton, J. Rose, Z. Vranesic, “Structural Analysis and Generation of Synthetic Digital Circuits with Memory”, *IEEE Transactions on VLSI*, Vol. 9, No. 1, pp. 223-226, February 2001.

[8] M. Hutton, J. Rose and D. Corneil, “Automatic Generation of Synthetic Sequential Benchmark Circuits”, *IEEE Transactions on CAD*, Vol. 21, No. 8, pp. 928-940, August 2002.

[9] K. Compton, *Architecture Generation of Customized Reconfigurable Hardware*. PhD Dissertation, Department of ECE, Northwestern University, 2003.