Design of Steering Vectors for Dynamically Reconfigurable Architectures

Nick A. Mould[1], Brian F. Veale[2], John K. Antonio[3], Monte P. Tull[1], and John R. Junger[1]

[1]*School of Electrical and Computer Engineering*
University of Oklahoma
Norman, OK 73019 USA
{nick_mould, tull, jjunger}
@ou.edu

[2] *Austin, TX 78729 USA*
veale@acm.org

[3] *School of Computer Science*
University of Oklahoma
Norman, OK 73019 USA
antonio@ou.edu

Abstract

An architectural framework is studied that can perform dynamic reconfiguration. A basic objective is to dynamically reconfigure the architecture so that its configuration is well matched with the current computational requirements. The reconfigurable resources of the architecture are partitioned into N slots. The configuration bits for each slot are provided through a connection to one of N independent busses, where each bus can select from among K configurations for each slot. Increasing the value of K can increase the number of configurations that the architecture can reach, but at the expense of more hardware complexity to construct the busses. Our study reveals that it is often possible for the architecture to closely track ideal desired configurations even when K is relatively small (e.g., two or four). The input configurations to the collection of busses are defined as steering vectors; thus, there are K steering vectors, each having N equal sized partitions of configuration bits. A combinatorial approach is introduced for designing steering vectors that enables the designer to evaluate trade-offs between performance and hardware complexity associated with the busses.

1. Introduction

A number of studies have been conducted that illustrate the potential advantages of dynamic reconfiguration [1, 2], also called runtime reconfiguration [3]. The dynamic reconfiguration approaches devised by such studies are often evaluated through simulation of the assumed underlying reconfigurable architecture. Simulation is used because commercially available reconfigurable devices generally cannot achieve reconfiguration times that are small enough to be plausible for applications and approaches that require highly dynamic reconfiguration.

In this paper, a framework for a dynamically reconfigurable architecture is described, which includes an interconnection scheme between steering vectors and the reconfigurable resources, described earlier in [3 – 5]. The framework is relatively generic and can be applied to model a number of existing approaches for dynamic reconfiguration. For example, it is applicable to instruction-level architectures in which the functional units of a superscalar processor are assumed to be able to be dynamically reconfigured [4, 5]. It is also applicable to task-level architectures in which dynamic reconfiguration is used to support higher-level computations such as signal processing [6] or data compression [7].

The next section introduces a parameterized model for the assumed architectural framework. Section 3 describes a combinatorial approach to designing the steering vectors associated with the assumed framework. The design of steering vectors is important because it impacts which configurations are reachable by the architecture. An illustrative example of applying the proposed approach is provided in Section 4, followed by some concluding remarks in Section 5.

2. A Framework for Dynamically Reconfigurable Architectures

Figure 1 illustrates a conceptual framework for an architecture that can support dynamic reconfiguration. The framework has three main components: reconfigurable resources; an interconnection network; and steering vectors.

The reconfigurable resources are partitioned into N slots, as shown in Figure 1 for $N = 5$. Configuration

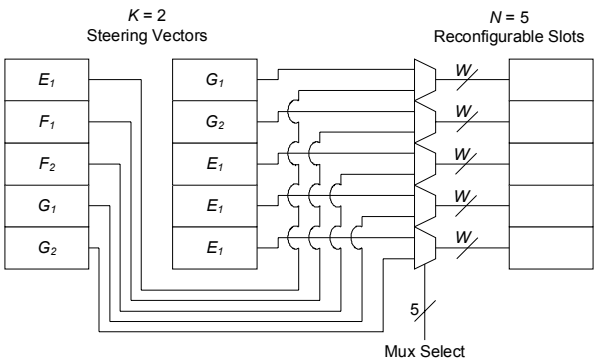

Figure 1. Conceptual framework for a dynamically reconfigurable architecture with *K* = 2 steering vectors, *N* = 5 reconfigurable slots, and busses of width *W*.

bits are used to define the configuration of each reconfigurable slot, and these bits are stored in memory that defines the steering vectors. Each slot can be reconfigured independently from the other slots. Thus, it is possible for one or more slots to be loading new configuration bits (i.e., reconfiguring) while other slots are performing computations. Furthermore, adjacent slots can be ganged together to form a functional unit that spans multiple slots.

The width of the data paths, W, as illustrated in Figure 1, defines the number of configuration bits that are loaded in parallel on each bus cycle. Thus, at one extreme, $W = 1$ represents the case where the hardware only supports configuration bits being loaded in a bit-serial fashion. At the other extreme, W could be on the order of thousands or even hundreds of thousands, which would drastically reduce the time required to load configuration bits into the reconfigurable slots. For dynamic reconfiguration to be practical and useful, the value of W must be sufficiently large so that the delay associated with loading configuration bits can be tolerated. The fundamental issue is that the time required to reconfigure must be more than compensated for by the advantage, in terms of performance, in electing to perform the reconfiguration.

The interconnection network (i.e., the MUXs in Figure 1) provides switching action from configuration bits (stored in the steering vectors) to the reconfigurable resources. For the study here, the interconnection network is assumed to be comprised of N independently controllable busses; however, in principle, more sophisticated interconnection schemes can be assumed for this component of the framework. Each bus assumed here can be independently controlled to select from among K configurations. As mentioned above, constructing wider buses has the advantage of decreasing reconfiguration time, but the disadvantage of requiring more hardware to implement. The value of K impacts the number of overall configurations that can be reached by the reconfigurable resources: larger values of K generally afford a larger number of configurations to be reached. However, larger values of K translate, again, into more hardware to implement the busses.

The input values associated with each of the N busses in Figure 1 are the configuration bits stored in memory, and are defined to be corresponding elements of K steering vectors. Each of the N elements of a steering vector stores the configuration bits associated with a functional unit, or a portion of a functional unit. In the example shown, there are configuration bits stored in the steering vectors that represent configurations for three functional units, denoted by E, F, and G. The configuration bits for unit E fit into one slot, and are denoted by E_1. Units F and G each require two slots; thus, their configuration bits require storage that spans two adjacent elements of a steering vector, denoted by elements F_1,F_2 and G_1,G_2, respectively.

Two obvious examples of configurations that can be reached by the steering vectors of Figure 1 are the two steering vectors themselves, i.e., $(E_1, F_1, F_2, G_1, G_2)^T$ and $(G_1, G_2, E_1, E_1, E_1)^T$. Furthermore, because the $N = 5$ busses are independently controlled, it is possible to reach a configuration that is a combination of the two steering vectors, such as $(G_1, G_2, E_1, G_1, G_2)^T$. Thus, the architectural framework enables the reconfigurable slots to be loaded with a mixture of elements from the steering vectors. In general, there are a total of $N \log_2 K$ select lines that are available for controlling the selection lines of the MUXs. For the case shown in Figure 1 with $N = 5$ and $K = 2$, there are five selection lines.

Based on the steering vectors defined in Figure 1, observe that it is not possible to reach a configuration in which there are two copies of unit F, and one copy of unit E loaded into the reconfigurable resources. However, if the steering vectors defined in Figure 2 were to be employed instead, then this configuration is indeed reachable. The question addressed in Section 3 is how to design the steering vectors so that desired configurations of the reconfigurable resources can be reached.

An important objective considered in this study is to utilize a value of K (i.e., number of steering vectors) that is as small as possible, because large values of K require greater hardware complexity to implement. This complexity manifests itself as logic complexity required to implement the $K \times 1$ busses. Note that the

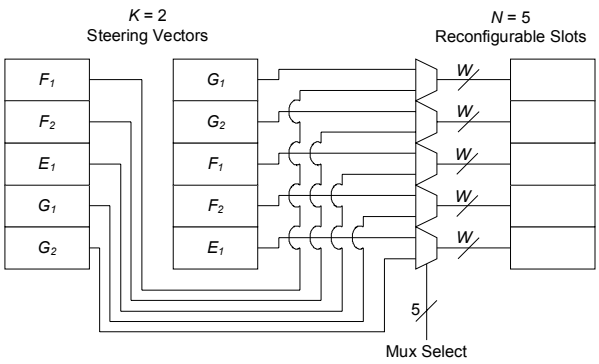

Figure 2. Same conceptual framework as Figure 1, but with modified steering vectors.

configuration bits for each unit only need to be stored once, and fanned out to the appropriate MUXs. The overarching theme, therefore, is to design K steering vectors, with K being as small as possible, to allow the system to reach those configurations that are known to be desirable. For example, it is a waste of hardware complexity to implement a system that supports four steering vectors, if two steering vectors exist that enable the architecture to reach all desirable configurations. In such a case, the complexity saved by decreasing K from four to two could be re-applied to increase the bus width, W, and thereby decrease reconfiguration time.

3. Steering Vector Design

Up until this point, only architectural examples in which the steering vectors were already defined have been considered, e.g., refer to Figures 1 and 2. This section addresses how to determine, in a systematic way, the best choices for the number and composition of steering vectors.

3.1. Mathematical Notation

To precisely formulate the steering vector design problem, mathematical notation is introduced. Denote the K steering vectors as $s_1, \ldots, s_K$, where each steering vector is of size N slots and each slot represents a distinct portion of a functional unit. The i^{th} element of a steering vector stores configuration bits that are used (when selected) to configure the i^{th} slot of the reconfigurable resources.

To model the control of selection for the busses, define K control vectors $c_1, \ldots, c_K$, where each control vector is of length N. A valid collection of K control vectors must satisfy the following two conditions:

- The elements of the control vectors can only be zero or one, i.e., $c_i \in \{0,1\}^N$, for all $i \in \{1,2, \ldots, K\}$.
- The sum of all K control vectors equals a vector having all elements equal to unity.

For a given collection of control vectors, the configuration that is loaded into the reconfigurable resources is denoted by the vector ℓ, where

$$\ell = \sum_{i=1}^{K} c_i \circ s_i \tag{1}$$

and the "$\circ$" operator denotes Hadamard (entrywise) product of two vectors. To illustrate the notation, the $K = 2$ steering vectors in Figure 2 are defined by s_1 and s_2 as:

$$s_1 = \begin{pmatrix} F_1 \\ F_2 \\ E_1 \\ G_1 \\ G_2 \end{pmatrix} \text{ and } s_2 = \begin{pmatrix} G_1 \\ G_2 \\ F_1 \\ F_2 \\ E_1 \end{pmatrix} \tag{2}$$

An example of two valid control vectors are:

$$c_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \text{ and } c_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} \tag{3}$$

Thus, the overall configuration that would be loaded into the reconfigurable resources is given by:

$$\ell = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \circ \begin{pmatrix} F_1 \\ F_2 \\ E_1 \\ G_1 \\ G_2 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} \circ \begin{pmatrix} G_1 \\ G_2 \\ F_1 \\ F_2 \\ E_1 \end{pmatrix} = \begin{pmatrix} F_1 \\ F_2 \\ F_1 \\ F_2 \\ E_1 \end{pmatrix} \tag{4}$$

Two configurations of the reconfigurable resources are defined to be equivalent if each configuration contains the same number of each type of functional unit. For example, the configuration $(E_1, E_1, G_1, G_2, E_1)^T$ is equivalent to the configuration $(G_1, G_2, E_1, E_1, E_1)^T$. Thus, if one configuration is a permutation of another, they are said to be equivalent and are members of the same equivalence class. As another example, the two steering vectors defined in Figure 2 belong to the same equivalence class.

3.2 Design Methodology

For this study, assume that the number of slots, N, and the number and size of each type of functional unit are given. In [5], a methodology was developed for enumerating all equivalence classes of configurations of reconfigurable resources (given the number of slots and the number and size of each type of functional unit).

It is important to note that the methodology of [5] does not specify the number and composition of steering vectors, rather, it defines all possible configurations based on the given number of slots and the number and sizes of the functional units. In [5], an example calculation is performed for the case of $N = 8$ and five functional units in which the first functional unit is of size one, the second is of size two, the third is of size two, the fourth is of size three, and the fifth is of size three. For that particular example, it is shown in [5] that all possible configurations are represented by only 36 equivalence classes. Note that, many of the equivalence classes contain a relatively large number of permutations.

The present approach first requires the designer to specify a collection of configurations that are deemed most important (i.e., should be reachable). For the case described in the previous paragraph, the designer specifies which of the 36 equivalence classes must be reachable. Suppose, for the sake of discussion, that the designer specifies that only twelve of the 36 possible equivalence classes need to be reachable. A secondary specification from the designer is the degree of importance of each of the configurations that need to be reachable. Given this input from the designer, the objective of our approach is to aid the designer in specifying a minimal set of steering vectors (two is better than four) that satisfy the designer's requirements.

To formalize the approach thus far; given that the number of slots N is specified and that the number and size of each functional unit is specified, the first step is to determine the associated equivalence classes for all possible configurations. Denote the number of equivalence classes by Q, and denote representatives from each of these equivalence classes with the vectors $V_1,...,V_Q$. Let $[V_i]$ represent the equivalence class associated with V_i, and $|[V_i]|$ represent the number of members (i.e., permutations) in $[V_i]$. Define $\mathcal{U}$ as the universe of all possible permutations, which is the union of all equivalence classes:

$$\mathcal{U} = \bigcup_{i \in \{1,...,Q\}} [V_i] \tag{5}$$

Consider a collection of K steering vectors chosen from the universe $\mathcal{U}$. Let L denote the number of possible ways to select K steering vectors from the universe $\mathcal{U}$, thus

$$L = \frac{|\mathcal{U}|!}{(|\mathcal{U}| - K)!K!}. \tag{6}$$

Let $\mathcal{S}_i \subset \mathcal{U}$ denote the i^{th} collection of K steering vectors selected from the universe, where $i \in \{1,2,\ldots,L\}$. Construct an $L \times Q$ matrix M, where each column in the matrix is associated with an equivalence class and each row is associated with a set of steering vectors. The value of matrix element m_{ij} denotes the number of members (i.e., permutations) of equivalence class $[V_j]$ that can be reached by employing the collection of steering vectors associated with $\mathcal{S}_i$.

The i^{th} row of the matrix M is associated with the i^{th} possible selection of K steering vectors, $\mathcal{S}_i$. The values of the elements in the i^{th} row of M correspond to how many of the members of each equivalence class can be reached by employing the steering vectors in $\mathcal{S}_i$. Thus, different choices for steering vectors can be compared across the rows of M. If a particular equivalent class represents reachable configurations that are very important to the designer, then a row in which the corresponding element is non-zero would matches this requirement, whereas a row in which the element is zero implies that the corresponding equivalence class cannot be reached. The more important a particular equivalence class is to the designer, the higher the corresponding value in the row should be. For example, choosing a row (a choice of steering vectors) in which the value associated with a particular equivalence class is higher, compared to another choice, means that there are more ways for the architecture to arrive at a configuration associated with the desired equivalence class.

Because equivalence classes are of different sizes, it could be important to normalize the elements in M by dividing the elements in each column of M by the size of the corresponding equivalence class. In so doing, each element will be normalized to between zero and one, representing the fraction of the possible members of each equivalence class that can be reached by the choice of steering vectors. Thus, an ideal choice of a row (steering vectors) would correspond to a row of ones, meaning that all possible permutations are reachable. In a constrained design, however, it is

desirable for K to be as small as possible, which inevitably translates to zero entries in the matrix M. Because it is assumed that the designer knows which configurations (i.e., equivalence classes) are important, and which ones are not, these requirements can be translated into a desired row of values. Selecting the best collection of steering vectors then reduces to the problem of finding a row in M that is equal (or similar) to a row containing the desired values. Example 1 below shows the calculations associated with the process described in this section.

Example 1. Assume three functional units of type A, B, and C, each requiring 1, 2, and 3 slots, respectively. Also, assume a configuration space size of $N = 4$ slots and that it is desired to use $K = 2$ steering vectors.

First, generate the equivalence class representatives; in this case there are $Q = 4$. These can be determined according to the method presented in [5]; they are given by:

$$\begin{aligned} V_1 &= (A_1, A_1, A_1, A_1)^T \\ V_2 &= (A_1, A_1, B_1, B_2)^T \\ V_3 &= (B_1, B_2, B_1, B_2)^T \\ V_4 &= (A_1, C_1, C_2, C_3)^T \end{aligned} \tag{7}$$

Next, generate the permutations (members) for each equivalence class to construct the universe $\mathcal{U}$ of all possible permutations:

$$[V_1] = \left\{ \begin{pmatrix} A_1 \\ A_1 \\ A_1 \\ A_1 \end{pmatrix} \right\} \quad [V_2] = \left\{ \begin{pmatrix} A_1 \\ A_1 \\ B_1 \\ B_2 \end{pmatrix}, \begin{pmatrix} A_1 \\ B_1 \\ B_2 \\ A_1 \end{pmatrix}, \begin{pmatrix} B_1 \\ B_2 \\ A_1 \\ A_1 \end{pmatrix} \right\} \tag{8}$$

$$[V_3] = \left\{ \begin{pmatrix} B_1 \\ B_2 \\ B_1 \\ B_2 \end{pmatrix} \right\} \quad [V_4] = \left\{ \begin{pmatrix} A_1 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix}, \begin{pmatrix} C_1 \\ C_2 \\ C_3 \\ A_1 \end{pmatrix} \right\} \tag{9}$$

Because there are seven total vectors in the universe $\mathcal{U} = \bigcup_{i \in \{1,\ldots,Q\}} [V_i]$, and we are assuming $K = 2$ steering vectors are to be employed, there are $L = \dfrac{7!}{(7-2)!2!} = 21$ possible sets of steering vectors that need to be considered. For the sake of space, each independent set is not shown here; instead the completed matrix is shown in Equation (10).

For example, the steering vectors associated with the last row of M can only reach the two members of $[V_4]$.

$$M = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 2 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 2 \end{bmatrix} \tag{10}$$

Observe that if a collection of two steering vectors existed that could reach at least one member of every equivalence class, there would be a row in M with all non-zero entries. Thus, because there is no such row, it is clear that all possible choices of two steering vectors are unable to reach at least one configuration in every equivalence class. As mentioned above, the final choice (last row of M) of steering vectors can reach only configurations in equivalence class $[V_4]$. The

third row of the matrix corresponds to a choice of steering vectors that can reach a maximum number of configurations; however, this choice cannot reach configurations associated with the equivalence class $[V_4]$. Observe that if K is defined to be three or four (instead of two), then the number of rows in M would increase. If K is four, there will exist a row in the matrix in which the four steering vectors are from each of the four equivalence classes; these four steering vectors would able to reach all equivalence classes. But recall that allowing K to be large increases the hardware complexity associated with constructing the busses. The aim is to keep K as small as possible, and still arrive at a choice of steering vectors that enable the architecture to reach desirable configurations. So, for the current example, if the configuration associated with $[V_4]$ is never needed, then the steering vector choices associated with the fourth or eighth rows would be good design choices.

4. Case Study

A general-purpose processor architecture with dynamically reconfigurable functional units was proposed in [3]. This basic concept was studied further and extended in [4] and [5]. For the case study presented in this section, it is assumed that the reconfigurable processor has $N = 5$ slots and that the objective is to design $K = 2$ steering vectors that are well matched to the configurations determined to be important. The specific objective is to exploit as much instruction-level parallelism as possible by being able to reach important configurations, i.e., those configurations that enable as many instructions to be executed in parallel as possible. For example, if it is the case that multiplication instructions can never (or rarely) be executed in parallel, but parallel addition instructions can often be executed in parallel, then the choice of steering vectors should comprehend this reality and enable configurations with two or more adder units to be reachable.

For the purposes of this study, the practical, yet application specific, techniques of code execution profiling and tracing are employed to identify important configurations. This approach is a practical off-line design strategy in which the implemented system has extreme performance requirements.

To identify the desired configurations for this study, a benchmark program is traced and potential instruction-level parallelism is analyzed by simulating the execution of the benchmark. In particular, the Susan benchmark from the Automotive and Industrial Control category of the MiBench set of embedded benchmarks was traced and analyzed [8].

Four functional units are considered that can be loaded in the reconfigurable resources. This collection of functional units is assumed to be capable of executing all of the instructions required by the benchmark. The functional unit descriptions and relative sizes are:

- Integer Arithmetic Logic (IAL) Unit of size 1;
- Integer Multiply Divide (IMD) Unit of size 2;
- Floating Point Arithmetic Logic (FAL) Unit of size 2; and
- Floating Point Multiply Divide (FMD) Unit of size 3.

In this simulation, instructions are placed in an instruction buffer that can hold eight instructions. On each clock cycle, the instruction buffer is analyzed; instructions that have no dependencies are removed from the buffer and assigned to a functional unit. The simulation assumes ideal availability of the functional units required to exploit all of the parallelism present in the instruction buffer during each cycle.

Ideal availability of functional units is equivalent to being able to reach any configuration necessary for exploiting the available instruction-level parallelism. Simulation of the temporal evolution of the instruction buffer is used to determine which configurations of functional units provide the greatest advantage in terms of clock cycle time. Further analysis of available instruction-level parallelism at each clock cycle provides a means of determining the global importance of each configuration.

Table 1 shows the number of clock cycles during which each configuration must be utilized in order to exploit all of the available instruction-level parallelism. Note that Table 1 does not report the total number of cycles required to execute the program using a single configuration, i.e., all of the configurations listed in Table 1 were required for exploiting all of the instruction-level parallelism. The summation of values listed in the "Utilization" column of Table 1 represents the total number of cycles required to execute the program (assuming ideal parallelism). Furthermore, the values reported do not account for clock cycles devoted to reconfiguration time.

In Table 1, the available instruction-level parallelism associated with the first seven configurations are all satisfied by the configuration containing one FAL, one IMD, and one IAL. For example, the first configuration does not make parallel use of functional units; however, the single unit requirement (one FAL unit) is indeed a subset of the seventh configuration, which has three units and utilizes all $N = 5$ slots of the reconfigurable resources. Thus, the configuration vector $V_1 = (\mathrm{FAL}_1, \mathrm{FAL}_2, \mathrm{IMD}_1, \mathrm{IMD}_2, \mathrm{IAL}_1)^T$ represents configuration number

7 in Table 1, which covers the first seven entries in the table.

It is assumed here that important configurations can be identified based on the number of clock cycles required of a specific configuration, and that each functional unit is required to appear in at least one position among the steering vectors. The combinatorial technique presented in Section 3 can be used to design a set of steering vectors to operate on the Susan benchmark.

Equation (11) is one example set of steering vectors obtained with the combinatorial technique introduced in Section 3, for $K = 2$ steering vectors and $N = 5$ reconfigurable slots.

$$s_1 = \begin{pmatrix} \text{FAL}_1 \\ \text{FAL}_2 \\ \text{IMD}_1 \\ \text{IMD}_2 \\ \text{IAL}_1 \end{pmatrix} \quad s_2 = \begin{pmatrix} \text{FMD}_1 \\ \text{FMD}_2 \\ \text{FMD}_3 \\ \text{IAL}_1 \\ \text{IAL}_1 \end{pmatrix} \tag{11}$$

Generation of these steering vectors was performed using execution times from Table 1, assuming that the goal of the design is to maximize possible instruction-level parallelism.

The combinatorial techniques presented in Section 3 provide a powerful method of generating steering vectors from specific design constraints. In addition, alternative methods of identifying the important configurations lend themselves well to this approach, as generation of the steering vector sets do not depend on the method used to assign importance to a configuration, i.e., rather than selecting configurations based solely on execution time and exploitation of parallelism, one could integrate the cost of reconfiguration in terms of power and/or time.

5. Conclusions

In this paper, we have extended the work in [3], [4] and [5] by generalizing a framework for reconfiguration that considers the challenge of designing steering vectors with respect to specific hardware constraints; namely, constraints related to the interconnection network (MUXs) between the reconfigurable resources and the steering vectors, and the size of the steering vectors. We have demonstrated a method by which a designer can determine the best possible set of steering vectors given the functional unit information, the steering vector size, and the sizes of the multiplexers used in the interconnection network, i.e., K, the number of steering vectors

Table 1. Simulation results showing cycle counts associated with configurations to exploit available parallelism. Configurations utilizing < 5 slots are noted with "-" and those requiring > 5 slots are noted with a "+".

Configuration Number	# of Units of Each Type				Utilization (Cycles)
	FMD	FAL	IMD	IAL	
1-	0	1	0	0	14,687,394
2-	0	0	1	0	8,073,949
3-	0	0	0	1	5,305,970
4-	0	1	0	1	4,831,781
5-	0	1	1	0	3,927,892
6-	0	0	1	1	2,197,350
7	0	1	1	1	1,761,679
8-	0	2	0	0	1,345,299
9-	0	1	0	2	999,982
10-	0	0	0	2	392,283
11+	0	1	1	2	317,736
12-	0	0	0	3	314,990
13-	1	0	0	0	202,321
14	0	2	0	1	88,724
15+	0	2	0	2	81,216
16+	0	2	0	3	43,320
17-	0	0	2	0	21,387
18-	0	0	0	4	16,528
19	0	0	2	1	14,273
20+	2	0	0	0	8,378
21-	1	0	0	1	7,426
22	1	1	0	0	5,219
23+	1	1	0	1	3,577
24+	0	3	0	1	2,952
25+	1	2	0	0	1,908
26+	1	1	0	2	1,890
27	1	0	0	2	1,704
28+	0	3	0	0	1,476
29	1	0	1	0	840
30+	2	0	0	1	830
31+	1	2	0	1	636
32-	0	0	1	2	635
33	0	0	1	3	395
34+	1	0	0	3	388
35	0	1	0	3	323
36+	0	0	2	2	287
37+	3	0	0	0	31
38	0	0	0	5	19
39+	0	0	0	6	15
40+	0	0	1	4	3
41+	0	1	0	4	1

that can be supported.

The approach taken here is combinatorial and provides a consistent view of the configurable space, and as a measure of that space, the number of configurations that can be reached with the selection of a given set of steering vectors. Future work includes approaching this problem with optimization techniques, which may yield results without exhaustive combinatorial techniques.

6. References

[1] Hauser, J.R. and Wawrzynek, J., "Garp: A MIPS Processor with a Reconfigurable Coprocessor," *Proceedings of the 5th Annual IEEE Symposium on Field Programmable Custom Computing Machines*, 1997, pp. 12-21.

[2] C. Iseli and E. Sanchez, "Beyond Superscalar Using FPGAs," *Proceedings of the 1993 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 1993, pp. 486-490.

[3] Niyonkuru, A. and Zeidler, H.C., "Designing a Runtime Reconfigurable Processor for General Purpose Applications," *Reconfigurable Architectures Workshop (RAW 2004), Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, pp. 143–149, Apr. 2004.

[4] Veale, B.F., Antonio, J.K., and Tull, M.P., "Configuration Steering for a Reconfigurable Superscalar Processor," 12th Reconfigurable Architectures Workshop (RAW 2005), Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS 2005), Apr. 2005.

[5] Mould, N.A., Veale, B.F., Tull, M.P., and Antonio, J.K., "Dynamic Configuration Steering for a Reconfigurable Superscalar Processor," *13th Reconfigurable Architectures Workshop (RAW 2006)*, *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, Apr. 2006.

[6] Cardoso, J.M.; Simoes, J.B.; Correia, C.M.B.A.; Combo, A.; Pereira, R.; Sousa, J.; Cruz, N.; Carvalho, P.; Varandas, C.A.F**.,** **"**A high performance reconfigurable hardware platform for digital pulse processing," *IEEE Transactions on Nuclear Science*, June 2004, pp. 921-925.

[7] Bishop, S.L.; Rai, S.; Gunturk, B.; Trahan, J.L.; Vaidyanathan, R., "Reconfigurable Implementation of Wavelet Integer Lifting Transforms for Image Compression," *IEEE International Conference on Reconfigurable Computing and FPGAs,* Sept. 2006.

[8] Guthaus, M. R., Ringenberg, D. E., Austin, T. M., et al, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," *Proceedings of the 4th Annual IEEE Workshop on Workload Characterization*, Dec. 2001, pp. 3-14.