Experimental Generation of Configurable Circuits for Rotationally Symmetric Functions

Andreas C. Doering
IBM Research – Zürich
Säumerstrasse 4
8803 Rüschlikon/Switzerland
ado@zurich.ibm.com

***Abstract*—With increasing one-time costs for the production of integrated circuits, the drive to integration of configurable circuits together with standard processor cores and interface will increase. So far, either established FPGA fabrics (e.g. Xilinx ZYNQ family) have been used or the configurable units were custom designed for a very specific function (e.g. PowerEN EFSM – TBD). It is therefore of interest to investigate the structures and algorithms for configurable circuits for a well-defined set of functions. As a first step, this paper investigates the class of functions which are invariant under cyclic shifts of their input vectors.**

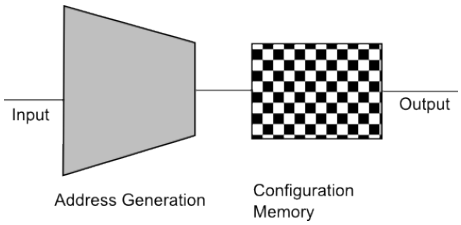

Fig. 1. Configurable circuit for target function set defined by an input property

I. Introduction

Configurable circuits, such as Field Programmable Gate Arrays (FPGAs) or Programmable Logic Devices (PLDs), have been developed with a pragmatic approach: benchmarks are used to evaluate whether a given architecture can implement the functions of the desired domain with acceptable efficiency. Some work [1] has been done to adapt circuits to a given set of target functions, but by far not systematically. With the current high integration of several processor cores and specialized processing cores, such as Digital Signal Processing, or Graphics Processing Units, configurable logic blocks on a System-On-Chip device will have very specific application field[2]. In the same way as application-specific processors have become established building blocks, application-specific configurable logic is interesting from a scientific and a commercial point of view. For non-configurable circuits, trade-off strategies between speed and cost are well understood, but the field of configurable circuits has additional parameters such as size of configuration data, speed of reconfiguration, or coverage of the target function space.

One approach to gaining a deeper understanding is the design of configurable circuits for a given application domain, e.g., cryptographic functions, even if that field can only be vaguely defined. This paper follows the opposite approach: a mathematically well understood set of functions is chosen and their configurable logic is investigated. This has the advantage that the exact size of the set of target functions is known. One of the simplest cases for a special class of functions is obtained by restricting the input: A single input set of binary vectors of length n, $\{(0,0,\ldots,0),\ldots,(1,1,\ldots,1)\}$ is partitioned into sets. Only those functions are considered that have the same input value on the partition sets. Formally, $\{f : a \sim b \Rightarrow f(a) = f(b)\}$. To obtain interesting results the equivalence relation $\sim$ is chosen such that the set of equivalence classes is considerably smaller than the original input set, but not too small. This definition of a function class could be thought of as an input property. Accordingly, the resulting circuit consists of a first stage that compresses the input space into a continuous address range and a look-up table (Figure 1). Therefore, and not surprisingly, the properties of the input set and the set of function values are independent: the compression circuit that assigns an address to each equivalence class of input values could be the same for a small or a large output set. For a small output set (e.g. one bit), there might however be a tradeoff by using a less complex compression circuit that maps some equivalent input values to different addresses. The configuration memory would be larger than the minimum. The functions could be considered filters of the input.

To investigate circuits, a cost measure is needed. To handle the complex functions used in this paper, a synthesis approach is used. The sub-circuits used to build the address generation circuit, such as pairwise-AND followed by a population count, were generated by a C program in verilog source and then synthesized. The un-routed netlist after driver strength tuning is used because placement and wiring are better done for the combined circuit. The open-source tool chain Qflow (www.opencircuitdesign.com) combines various academic tools and an open gate library (for 0.35μm). Cost is measured relative to the size of the smallest inverter. A perl script was developed that sums the sizes of all cells. In this way the methods and results of this paper can easily be reproduced in contrast to using a potentially more optimized synthesis tool and cell library.

The algorithm presented has a high degree of parallelism on several levels, including batch parallelism to run the verilog circuit generator, synthesis, and cost analysis script for a considerable number of basic rotationally symmetric functions.

The paper is organized as follows:

- Section II contains the terminology, and an introduc-

tion of rotationally symmetric functions.

- Section III describes the circuit generation algorithm.
- Section IV lists the cost of basic rotationally symmetric functions for various input sizes, one detailed study for one input size, run-time
- Section V discusses alternative approaches.
- The paper concludes with an outlook on further work.

II. Rotationally Symmetric Functions

In the following, the Boolean values true and false are interpreted as natural numbers 1 and 0. Hence, the population count function of a vector of Booleans, which gives the number of elements with value true, can be written as

$$\mathrm{pc}(x_{n-1},\ldots,x_0)=\sum_{i=0}^{n-1}x_i.$$

The length n of the input or intermediate vectors is typically omitted. Of course, cost, speed, algorithm complexity, etc. are considered in relation to n.

A fully symmetric function f is invariant under any permutation of its inputs. The population count (pc) function covers all fully symmetric functions, i.e., any given symmetric function can be represented as a concatenation of pc and a third function: $f(x)=g(\mathrm{pc}(x))$, as each input could be sorted without changing the function f.

A rotationally symmetric function is invariant only under rotations (cyclic shifts) of its input vector:

$$\forall(x_0,\ldots,x_{n-1}):f(x_0,\ldots,x_{n-1})=f(x_{n-1},x_0,\ldots,x_{n-2})$$

It is sufficient to require only shift by one position because the equivalence under wider shifts follows by several applications of the rule. There is an entire mathematical discipline that covers functions under invariants, called invariance theory. As this field is quite mature, I refer the interested reader to a text book [3].

In Table I several rotationally symmetric functions are listed as examples, but also because of their relevance later in this paper. For clarity, the following function names are used: $r_i(x_0,\ldots,x_{n-1})=(x_i,\ldots,x_{n-1},x_0,\ldots x_{i-1})$ denotes the rotation by i positions, $m(x_0,\ldots,x_{n-1})=(x_{n-1},\ldots x_0)$ is the reflection ("mirror image") of the given vector, and $z(\mathbf{x})=x_0\&x_1\&\ldots\&x_{n-1}$ is the AND-combination of the vector elements.

As can be seen all functions listed have polynomial cost. By the rotational symmetry, the equivalence classes contain at most n elements. Therefore, approximately m^{2^n}/n different functions exist (where m is the size of domain). Hence, almost all rotationally symmetric functions have exponential cost. This applies in particular for the configuration memory in Figure 1.

The equivalence problem for vectors under rotational symmetry is known in combinatorics as *0-1-colored necklace*.

The mirrored vector $m(x)$ is in some cases equivalent under rotational symmetry with x, for instance if $\mathrm{pc}(x)\in\{0,1,2,n-2,n-1,n\}$. Whether this is the case is is indicated by "Invariance under Reflection". It is interesting because it complements the counted product term functions. For those input vectors which are not invariant under reflection it is desirable to distinguish between the two classes. The function Orientation in the table does this, so at a high cost. For odd length I have found a better function.

The counted product term functions represent a large set of functions, which contains both population count and the canonical functions. For example, if the product term is $x_0\&\bar{x_1}$, the corresponding counted product terms function determines the number of sequences of ones in the input vector. Therefore, the counted product term for $\bar{x_0}\&x_1$ is the same function. The parameter $\mathbf{c}$ gives the variables in the product term, $\mathbf{b}$ marks those variables which are inverted. If all input bits are included in the product term, the canonical functions result. Again, there are redundancies, if rotating both $\mathbf{a}$ and $\mathbf{c}$ yields the same function. For each input equivalence class exists a canonical function that is one (or n) only for input values from this class. Consequently, the set of canonical functions alone is sufficient to identify any input class, and thus the set of counted product term functions even more so.

Symmetric functions can be constructed from a given arbitrary function f by creating all rotated copies $f\circ r_i$ and combining them, for instance, by using parity, AND, OR, or any other (fully) symmetric function. The counted product terms function in Table I belong to this class. This method can be used to create a large number of rotationally symmetric functions with low (linear or polynomial) cost.

A function partitions its input set, where each partition belongs to those input values that yield the same function result. The same applies to a set of functions: $P(\{f_i\},(X))=\{\{x:\forall i f_i(x)=y_i\},\forall i:\exists x\in\mathbf{X},y_i=f_i(x)\}$. On each partition, all functions are constant. An address generation function results in a partition that corresponds to the given equivalence relation $\sim$; in the case considered here equivalence is rotational symmetry.

III. Circuit-Generation Algorithm

The algorithm for generating the addressing circuit for the configurable rotationally symmetric circuit is an A*-search on a directed, levelled (and hence acyclic) graph. Each level corresponds to a basic function (for instance the product-term-based functions of Table I). At each node it is decided whether the function of this level will be included into the resulting address generation. Hence, each node corresponds to a set of functions and thus a partitioning of the input set. In the general case, in which all rotationally symmetric functions should be implemented by the circuit constructed, the root corresponds to a partition with a single set, the input set. However, the algorithm can also be applied to special cases in which not all functions are used by selecting a different partition (e.g. a partition which consists of a true subset of all equivalence classes) at the root. A leaf is reached if the partition corresponds to the rotational symmetry. In my implementation, the cost of a node is simply the sum of the costs of the functions selected. This is an overestimation because when combining several functions, part of their additional information can be redundant.

TABLE I. TABLE OF TYPICAL ROTATIONALLY FUNCTIONS

Function	Definition	Cost	Logic Depth
Population Count	$\mathrm{pc}(\mathbf{x}) = \sum x_i$	$O(n \log n)$	$\log^2 n$
Counted Product Terms	$\mathrm{pc}(s_i), s_i = z((r_i(\mathbf{x})\mathrm{XOR}\mathbf{a})\&\mathbf{c})$	$O(n \log n + n(\mathrm{pc}(\mathbf{a})))$	$(\log^2 n) + \log \mathrm{pc}(\mathbf{a})$
Normalization	$k(\mathbf{x}) = \min(\{r_i(\mathbf{x})), i \in \{0, \ldots, n-1\}\})$	$O(n^2)$	$\log^2 n$
Invariance under reflection	$k(\mathbf{x}) = k(m(\mathbf{x}))$	$O(n^2)$	$\log^2 n$
Orientation I	$k(\mathbf{x}) < k(m(\mathbf{x}))$	$O(n^2)$	$\log^2 n$
Canonical	$k(\mathbf{x}) = \mathbf{c}$	$O(n^2)$	$\log n$

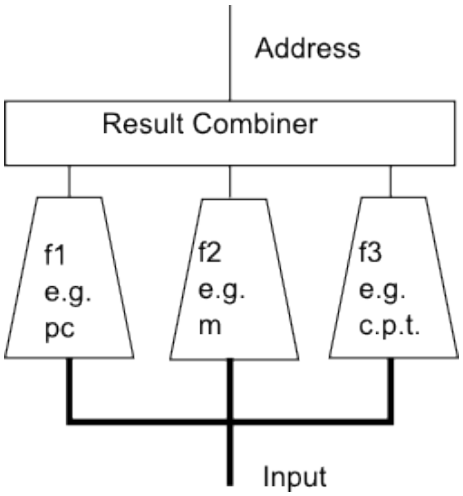

Fig. 2. Concept of address-generation circuit by combining several functions of the target set

The results of selected functions are combined in a final step, see Figure 2. This combination circuit is not discussed in this paper.

The A*-search also needs an estimation of the remaining cost. Of course, the sizes of the partition sets are one indicator how many more functions will be needed. In the ideal case, the functions at the lower levels will divide each partition set into nearly equally large sub-sets, so that the size of the largest set of the partition of the current node could be used. Its logarithm based on the result set size of the next functions is the lower bound of the number of remaining levels.

As an example, consider $n = 7$. Table II shows an example path through the search graph, adding functions with each row. Only the normalized result of each class is listed. Since 7 is prime, there are $(2^7-2)/7+2$ equivalence classes. Two classes with one element only (1111111 and 0000000), and 18 classes of 7 elements each.

Note that for larger n there are many more classes that are not invariant under reflection. These larger cases would have been impractical for a table, though. The first two functions are much stronger then.

The processing at each node requires the application of the new function to all non-singular partitions. This step has a high level of parallelism. At the high levels of the search graph, 2^n independent function applications have to be computed and the results collected in sets, e.g. using a hash table. The partitions can be stored in a distributed way as only their size is needed. The communication effort between the processors is low, as it scales with the number of function results of the new function.

TABLE III. SIZES AND RUN-TIME FOR SELECTED VALUES OF n

n	Cost	Functions used	runtime(s)	Nodes visited
6	198	4	5	5205
7	305	4	0.1	99
8	596	6	1.1	559
9	845	7	48	20000
10	1153	9	65	16645
11	2960	6	98	20000
12	3982	11	111	20000

A prerequisite for the search to converge is that the cost of filters has to grow eventually with increasing levels. Otherwise the search could follow the path that does not add new functions infinitely, in particular if the functions are worse than the heuristic estimates. Functions that are considered particularly useful can be put at high levels even if they are costly.

IV. RESULTS

The algorithm has been implemented in C++ using the Xerces-C and the Boost graph libraries. The algorithm is configured using a file in XML format. In addition to options, such as the length of the input vector and required output files, also a number of steps starting at the root can be controlled to be pre-computed before the search is started. Table III lists the runtime and the resulting size of functions for a range of input length values.

The search algorithm terminates when either no nodes for investigation are available anymore, three solutions were found or 20000 nodes visited. Of course, these quantitative limits are configurable. Once the first solution is found the algorithm discards all open nodes that have a higher cost than the previously found nodes. A further condition would be desirable, yet is not yet implemented: the condition that the search should also stop at levels below which only functions appear that are more expensive than the already found solution. As mentioned the cost has to increase eventually to enforce convergence. This is difficult to implement when the assignment of functions to tree levels is configurable. This sequence was for the results in Table III: Reflect, Orientation, Population Count, AND-2 (counted product terms with two positive literals), all other product terms. None of the solutions used the expensive Orientation function, most used population count.

The exceptional low runtime for the case $n = 7$ is due to the fact that the rotational symmetry has a simpler structure for prime number vector length cases. Why $n = 11$ was not simpler, I can't say. Maybe a different selection or sequence

TABLE II. REFINED PARTITIONING FOR $n = 7$, OR. STANDS FOR ORIENTATION I, IUR IS INVARIANCE UNDER REFLECTION, AND C.P.T. IS COUNTED PRODUCT TERM

Function	Partition
Start	$\{0, 1, 3, 5, 7, 9, 11, 13, 15, 19, 21, 23, 27, 29, 31, 43, 47, 55, 63, 127\}$
IuR	$\{0, 1, 3, 5, 7, 9, 15, 19, 21, 27, 31, 43, 47, 55, 63, 127\}, \{11, 13, 23, 29\}$
Or.	$\{0, 1, 3, 5, 7, 9, 15, 19, 21, 27, 31, 43, 47, 55, 63, 127\}, \{11, 23\}\{13, 29\}$
pc	$\{\{1\}, \{3, 5, 9\}, \{7, 19, 21\}, \{15, 27, 43\}, \{31, 47, 55\}, \{63\}, \{127\}, \{11\}, \{23\}, \{13\}, \{29\}\}$
C.P.T., $\mathbf{a} = 0000011$	$\{\{1\}, \{3\}, \{5, 9\}, \{7\}, \{19\}, \{21\}, \{15\}, \{27\}\{43\}, \{31\}, \{47, 55\}, \{63\}\{127\}\{11\}\{23\}\{13\}\{29\}\}$
C.P.T., $\mathbf{a} = 0000101$	$\{\{1\}, \{3\}, \{5\}, \{9\}, \{7\}, \{19\}, \{21\}, \{15\}, \{27\}, \{43\}, \{31\}, \{47\}, \{55\}, \{63\}, \{127\}, \{11\}, \{23\}, \{13\}, \{29\}\}$

of the functions would have revealed a better solution. The reason for the role of prime numbers is as follows: if the vector length n has a divider $1 < d < n$ then there are input pattern that already repeat after a rotation of d digits. Hence, the equivalence class for this pattern has only $n/$ elements, and therefore in total there are more equivalence classes than for comparable prime number length cases.

Figure 3 shows the search tree for $n = 6$. The best solution (with cost 357) is the one at the bottom, at level 13, which is found last.

For reference, Figure 4 shows the cost after synthesis of various functions.

V. ALTERNATIVE APPROACHES

It is not evident that the proposed method will find solutions that are at least nearly optimal. Therefore, it is desirable to investigate alternative approaches. Search methods such as genetic algorithms could be used to search directly for an address-generation function, or the address generation function is constructed bit-wise.

Another method is to apply functional decomposition methods to the problem. To do this the entire configurable circuit is given as one complex function; the configuration pins are also treated as inputs. This function is then analyzed by decomposition methods such as the one described in [4]. This has the advantage that memory addressing and access multiplexing are not enforced upfront, as in the approach presented here. The address to the configuration memory in Figure 1 is assumed as a binary address in the algorithm of Section III.

Finally, one could first establish an upper bound for the circuit size and then represent all possible circuits a closed form, so that either Binary Decision Diagrams or a SAT solver can be used to find optimal solutions.

VI. OUTLOOK

Although cost of the address generation function for the implementation of configurable rotationally symmetric functions can be determined with the algorithm presented, a comparison with existing configurable architectures is not yet possible, because this requires the determination of the minimum number of resources (look-up tables for FPGAs or product terms for PLDs) for the set of target functions. A first indication could be obtained by applying the method in [5] to randomly selected functions. As many functions will have a high complexity, a randomly selected sample should come close to the most

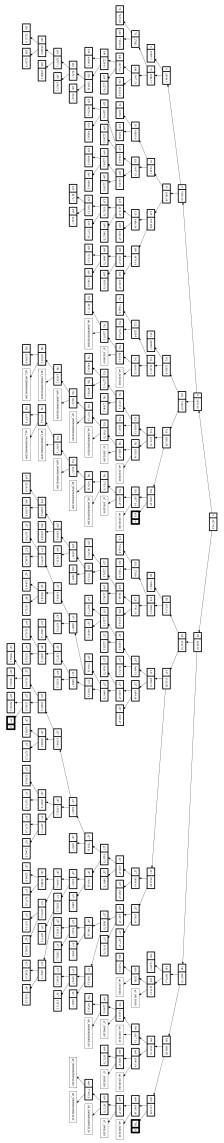

Fig. 3. Search tree for $n = 6$

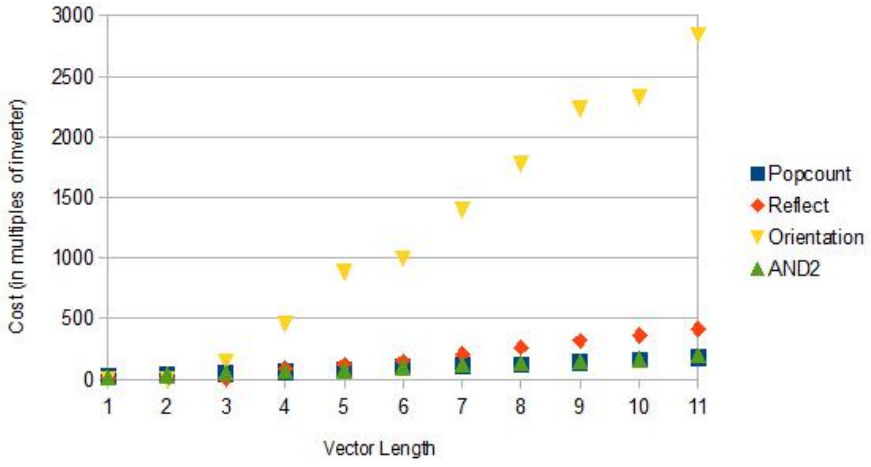

Fig. 4. Cost of various rotationally symmetric functions after synthesis as used as input to the search algorithm

expensive case. This approach could entail of course a high computational effort.

In addition to the creation of a configurable circuit, another important aspect is a mapping algorithm that translates a given function into a configuration of the circuit. Given the output-table-based architecture, this task is trivial for the architecture presented if the symmetry cycle is known. In the definition of rotational symmetry, the cycle was pre-defined as $(0, 1, \ldots, n-1)$. If the circuit presented is integrated in a system that allows arbitrary routing, such as an the routing structure in an FPGA, also functions that have a different symmetry cycle could be implemented in the circuit presented. I have developed an algorithm that can determine whether an arbitrary function has a symmetry, including symmetries consisting of several cycles or cycles covering only a subset of the inputs. This algorithm has been published in [6], to prevent a patent on the algorithm by others, but without any details or explanations. Another algorithm for the same problem is presented in [7], but it entails however a very high computation effort.

When a configurable circuit for a set of functions with a common input property is to be created, the primary need is a method to generate a large number of functions from the target set with low cost.

REFERENCES

[1] M. Holland and S. Hauck, "Automatic creation of domain-specific reconfigurable cplds for soc," in *Proc. FPL*, 2005, pp. 95–100.

[2] M. B. Taylor, "A landscape of the new dark silicon design regime," *IEEE Micro*, vol. 33, no. 5, pp. 8–19, 2013.

[3] B. Sturmfels, *Algorithms in invariant theory.*, ser. Texts and monographs in symbolic computation. Springer, 1993.

[4] C. Scholl, *Functional Decomposition with Application to FPGA Synthesis*. Kluwer, 2001.

[5] K. Atasu, T. Todman, O. Mencer, and W. Luk, "Optimal Implementation of Combinational Logic on Lookup Tables," in *The Fourth Conference on Ph.D. Research in Microelectronics and Electronics (PRIME'08), Istanbul, Turkey*, May 2008.

[6] A. Doering, "Determination of a generating cycle for a rotationally symmetric function," Ip.com, April 2009, iP.com Disclosure Number: IPCOM000182091D.

[7] P. Jasionowski, "Matrix characterization of symmetry groups of boolean functions," *Journal Algebra and Discrete Mathematics*, vol. 14, no. 1, pp. 71–83, 2012. [Online]. Available: http://adm.lnpu.edu.ua/downloads/issues/2012/N3/adm-n3(2012)-6.pdf