Instance-specific versus Parameter-specific Circuit Generation

Jacqueline E. Rice & Troy Ronda
University of Lethbridge
Dept. of Math & Computer Science
Lethbridge, Alberta, Canada
{j.rice, troy.ronda}@uleth.ca

Kenneth B. Kent & Zhao Yong
University of New Brunswick
Faculty of Computer Science
Fredericton, New Brunswick, Canada
{ken, b15v3}@unb.ca

Abstract – *There exist many computationally intensive problems for which the use of configurable hardware can provide a satisfactory solution. This paper examines two approaches to the design of configurable solutions: an instance-specific and a parameter-specific approach. We investigate both of these approaches as applied to the computation of the autocorrelation coefficients for a Boolean function.*

I. Introduction

One technique for accelerating computation is to introduce configurable hardware solutions. This has been done for problems such as string matching [4] and to solve the Hamiltonian cycle problem [8]. In this work we use configurable hardware in the computation of a mathematical transform known as the *autocorrelation (ac) transform*. By making use of hardware to perform all or part of the algorithm it is possible to not only speed up the computations, but also to do some of them in parallel. In this paper we compare two approaches to the problem. The first approach, based on the initial work in [5], uses an instance-specific approach. Additional work is introduced that improves resource usage. The second approach is a parameter-specific approach, also attempting to maximise resource usage.

II. Background

A. The Autocorrelation Transform

The ac function is defined as $B(u) = \sum_{v=0}^{2^n-1} f(v) \cdot f(v \oplus u)$ [3]. Values for u range from 0 to $2^n - 1$ where n is the number of inputs to the Boolean function $f(X)$. This results in 2^n coefficients $B(u)$. There are various techniques that may be employed that reduce the computation requirements from the exponential run-time that a naive implementation would require. Coefficients resulting from the application of the ac transform have been used in a number of areas including variable ordering for ROBDDs [7] and testing [1].

B. Instance-specific Approach

In the first approach a Binary Decision Diagram (BDD) [2] is used to represent the function $f(X)$ [6]. The theory behind an instance-specific approach is that for some classes of problems it may be better to utilise a configurable device to implement a solution tailored to a specific instance of the problem. For certain applications with desirable characteristics this can achieve a high performance increase [8]. One problem that may arise in the circuit generation lies in how to choose amongst varying levels of parallelism, which must be based both on the problem and on the characteristics of the FPGA. Certain amounts of parallelism will be inherent to the chosen approach; however, the tools used in the generation of the instance-specific circuit should be aware of the target environment and how the resulting circuit will be placed and routed.

C. Parameter-specific Approach

The second approach is based on a significantly different algorithm, requiring that the the input function be represented as a list of disjoint cubes for input to the FPGA [6]. This requires a small amount of preprocessing, currently done in software. The hardware is then used to implement a series of comparisons of the disjoint cubes, each of which compute a contribution to the overall coefficient values. This approach can best be described as *parameter-specific*. Rather than requiring a new circuit for every new instance of the problem, we design a circuit that may be used for many instances of the problem, as long as they fall within certain parameters.

III. Circuit Architectures

A. Instance-Specific Circuit Architecture

The hardware architecture used in approach 1 is described in detail in [5]. Briefly, the hardware architecture for this solution consists of three main components: a function component, the calculator, and the controller. This architecture is depicted in

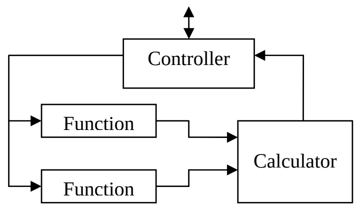

Fig. 1. Architecture of the instance-specific design used in approach 1.

Figure 1. The function component contains the logic function for which the ac coefficients are being computed. Two

instances of this component are used, one for each function calculation that is required for each summation. This component contains an embedding of the BDD representation of the logic function. The calculator is responsible for performing the exclusive-or and summation operations.

B. Parameter-Specific Circuit Architecture

The process for computing each ac coefficient from a disjoint cube-list is as follows:

- for each cube in the disjoint cube list
  - compute the exclusive-or of the cube and u;
  - search for the new cube or one containing it in the cube list
  - if either is found add 2 to the sum register as the contribution to the coefficient.

Complete details are given in [6].

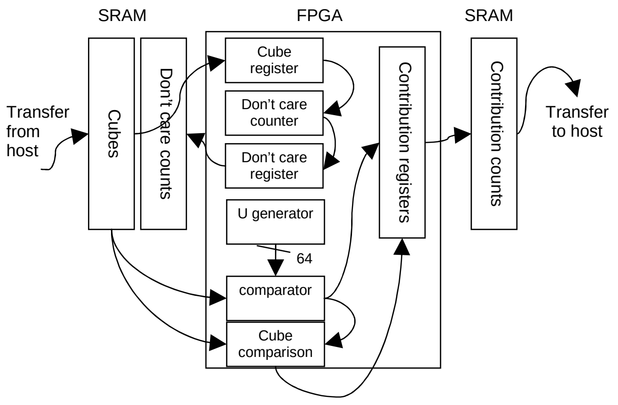

Fig. 2. Architecture of the parameter-specific design used in approach 2.

As shown in Figure 2, a daughterboard with on-board SRAM for storage of the input function was utilised. One of the keys to this algorithm is that the number of don't cares in each cube must be counted. This is done by the hardware solution, and is optimised by counting both halves of each cube word in parallel. There is a limit of one memory access to the SRAM per clock cycle; thus the design was optimised to minimise SRAM accesses and store any intermediate results on-chip.

C. Space Utilisation

As was found in both approaches, computation of the ac transform is a highly parallel problem. In each approach it was necessary to balance the addition of parallel computing components with the additional complexity and overhead such additions required.

In the instance-specific approach the architecture currently calculates one term of the summation each clock cycle. Given no restriction on design space, an obvious enhancement to the architecture would be to replicate the function components. This replication would allow for multiple terms of the summation to be computed in parallel during one clock cycle. Changes to the controller and calculator in support of these replications require minimal design space. The function component, however, is more demanding on design space. The size requirements of this component is dependent on the size of the BDD representation of the logic function, and so to create the design some estimate of the space requirements is required during circuit generation to determine the optimal amount of parallelism.

In the parameter-specific design a slightly different approach to the addition of parallelism was used. As shown in Figure 2, the u generator generates 64 values in parallel, which are then passed in one clock cycle to the comparator. The comparator is designed to have 64 comparator sub-components in order to support this. Thus for the computation of a single coefficient there is no real advantage, but for computations of more than one coefficient up to 64 can be performed in parallel.

IV. Experimental Results

The results given are for a series of single-output benchmarks from the ISCAS 89 set with a maximum of 32 inputs, as shown in Table I. This is due to the limitations of a 32-bit word size and 219 cubes inherent to the parameter-specific approach. Extensions to larger functions are discussed in Section V.

function	**inputs**	**BDD size (nodes)**	**num cubes (disjoint)**
9symml	9	25	87
cm152a	11	16	8
co14	14	27	47
ex10	5	6	16
ex20	5	11	7
ex30	5	10	4
life	9	26	512
majority	5	8	5
max46	9	75	46
mux01	21	33	36
ryy6	16	21	112
sym10	10	31	837
xor5	5	6	16

TABLE I

The functions used in these experiments and their sizes in terms of inputs, BDD nodes, and number of disjoint cubes.

A. CLB Utilisation

With the design for the instance-specific approach, it is possible to utilise up to 99.8% of the CLBs for a particular design. With the best combination of variables for the parameter-specific approach, 78% of the CLBs were utilised. Analysis of each approach is given below.

1) Instance-specific Approach: In the instance-specific approach each test utilised a varying amount of parallelism: 2 function computations (one summation term) in parallel, 4, 6, 8 and so on with the maximum amount computed in parallel being 252. For each test the percentage of CLBs utilised was measured, along with the minimum period (giving a maximum possible frequency) and the speed of computation, calculated based on the frequency. These experiments were conducted using the Xilinx ISE 6.3 tool set targeting the Xilinx Virtex-E 812E FPGA. This chip contains 18,816 4 input LUTs as CLBs (approximately 200,000 logic gates). Table II presents the complete results for one of the benchmarks, *ryy6*. The first column lists the number of parallel computations that were attempted. The results show that it is possible to maximize CLB usage through additional function components, but there is a side-effect of lowering the clock speed of the device. This drawback was not encountered in the parameter-specific approach. These results, while indicative of the general results for other benchmarks, do not necessarily hold for all benchmarks due to the nature of the instance-specific approach. Table III shows the amount of parallelism (column 2) resulting in the fastest computation for each of the benchmarks tested with the instance-specific approach. Figure 3 shows a graph relating the percentages of overall CLB usage, performance, and clock rates to the amount of parallelism introduced. This seems to indicate that the range of 158 to 252 function components in parallel will provide an increase in performance, depending on the function. This corresponds to usage of the device ranging from 91.4% to 99.8% of the available CLBs. For most functions, the highest level of performance is attained at the highest level of parallelism that the development tools could successfully place and route. The overall maximum clock rate for all of the generated circuits began to degrade rather quickly beyond the threshold of 6 function components. An important contributing factor to the performance is not just the number of CLBs used, but also the density of the circuit. The rate of increase in CLB usage in relation to the level of parallelism is approximately 1% usage per 2 functional units. At this low rate of increase the circuit does not suffer dramatically from an increase in density during the place and route process.

Parallelism	CLB Usage	Computation time (sec)	Min. period (ns)	Max. freq. (MHz)
2	2.5%	56.9	13.248	56.899
10	5.9%	13.21	15.376	13.208
20	10.2%	12.54	29.191	12.537
30	14.8%	12.002	41.198	12.002
40	19.4%	12.07	56.211	12.071
50	24%	11.636	67.726	11.636
60	28.6%	11.623	81.191	11.623
70	33.6%	11.197	91.247	11.197
80	38.1%	11.016	102.598	11.016
90	42.7%	10.637	111.447	10.637
100	46.9%	10.585	123.232	10.585
110	51.9%	10.355	132.606	10.355
120	55.8%	10.307	143.979	10.307
130	60.3%	10.2	154.358	10.200
140	65.8%	9.964	162.384	9.964
150	69.6%	9.998	174.571	9.998
160	75.2%	9.97	185.708	9.970
170	79.6%	9.867	195.291	9.867
180	84%	9.627	201.747	9.627
190	88.5%	9.542	211.047	9.542
200	93%	9.519	221.647	9.519
210	97%	9.526	232.872	9.526
214	98.9%	9.507	236.877	9.507

TABLE II

COMPLETE CLB USAGE AND TIMING RESULTS FOR THE BENCHMARK RYY6 USING THE INSTANCE-SPECIFIC APPROACH.

function	Parallelism	CLB Usage	Computation Time	Max. Freq. (MHz)
9symml	212	93.6%	0.577 ms	4.286
cm152a	226	99.8%	9.128 ms	4.066
co14	214	98.2%	593.224 ms	4.229
ex10	252	99.4%	0.00298 ms	2.726
ex20	250	99.8%	0.00298 ms	2.746
ex30	252	99.3%	0.00299 ms	2.709
life	216	97.1%	0.576 ms	4.214
majority	252	99.6%	0.00298 ms	2.726
max46	158	94.8%	0.597 ms	5.56
mux01	190	99.2%	9941 sec	4.657
ryy6	214	98.9%	9.507 sec	9.507
sym10	192	91.4%	2.340 ms	4.667
xor5	252	99.4%	0.00298 ms	2.726

TABLE III

RESULTS SHOWING THE FASTEST COMPUTATION TIMES AND CORRESPONDING AMOUNT OF PARALLELISM FOR APPROACH 1 (INSTANCE-SPECIFIC).

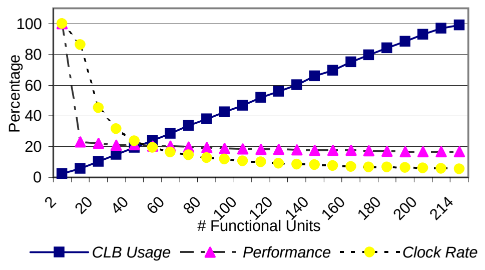

Fig. 3. Percentages of overall CLB usage, performance, and clock rates in relation to the amount of parallelism for approach 1.

2) *Parameter-specific Approach:* For the parameter-specific approach a Xilinx Virtex 812E chip was targeted with Xilinx ISE version 6.3 used for synthesis, place and route. The design was implemented using a variety of options, namely a varying number of bits for storage of each cube and a varying number of coefficients being computed in parallel. Table IV gives the resulting CLB usage, broken down by logic and routing requirements. It should be noted that for the instance-specific approach all results were obtained by simulation, while in the parameter-specific approach the results were obtained by actual execution on the targeted device.

Cube Bits	parallel coeffs	LUTs for logic	LUTs for routing	LUT Usage
32	64	13933	891	78%
26	64	12696	696	71%
21	64	11722	502	62%
15	64	10554	307	57 %
10	64	9588	176	51%
32	32	7412	477	41%
10	32	5119	114	27%
32	1	956	81	5%

TABLE IV

SPACE USAGE OF THE XILINX VIRTEX 812E CHIP FOR VARIOUS SCENARIOS OF THE SECOND (PARAMETER-SPECIFIC) APPROACH.

B. Timing Comparisons

A straight-forward comparison of timings for the various approaches is not possible. There is overhead in the various steps required for each approach. For example, for the benchmark *ryy6*, generating an instance-specific solution with no parallelism requires 110 seconds while 214 function components in parallel results in a much larger circuit that requires approximately 16 minutes to process. Table V provides a comparison of the run-times for the instance-specific approach, sequential and parallel versions of the parameter-specific approach and a software solution implemented on a Pentium 4. These timing results do not include any preprocessing or solution generation/configuration. The software solution is based on the same algorithm as the parameter-specific circuit, utilising a disjoint cube list. From the tables several interesting results are obtained. When comparing the two versions of the parameter-specific approach, with and without parallelism, the parallel version outperformed the sequential version in every benchmark except *ex30*. This is attributed to the small size of the benchmark.

	approach 1 best result (varied)	**approach 2 64 parallel (26 MHz)**	**approach 2 no parallel (26 MHz)**	**software Pentium 4 2.66 GHz**
9symml	0.000577	0.2940	1.0069	0.4160
cm152a	0.009128	0.2680	0.3181	0.0400
co14	0.593224	0.2490	0.4480	0.1050
ex10	0.00000298	0.2650	0.3024	0.0001
ex20	0.00000298	0.2680	0.2934	0.0170
ex30	0.00000299	0.3010	0.2968	0.0001
life	0.000756	0.2760	0.6843	0.1790
majority	0.00000298	0.2680	0.3102	0.0001
max46	0.000597	0.2750	0.3349	0.0430
mux01	9941	24.5830	309.2230	440.4620
ryy6	9.507	2.1330	32.8776	15.6440
sym10	0.002340	1.1740	27.9384	14.1680
xor5	0.00000298	0.2700	0.3058	0.0001

TABLE V

TIMES IN SECONDS TO COMPUTE ALL 2^n COEFFICIENTS FOR EACH OF THE VARIOUS APPROACHES.

In comparisons of the results from the various hardware and software implementations we can see that it is possible to achieve a performance gain of approximately 100 times through the use of parallelism. However, as above, given a small benchmark, the parallel parameter-specific version can provide a performance decrease. For the *ex30* benchmark software outperformed the hardware approach.

The parameter-specific version provides more "consistent" results while the instance-specific version provides a great deal of speed-up in some cases, and yet a significant performance decrease in some other cases. Most notably is the decrease in performance for the benchmark *mux01*. It is likely that the BDD representation is not an optimal choice for this particular benchmark. In all comparisons between the various approaches we must take into account that the underlying approach of either a BDD or a cube-list will perform better for some benchmarks and worse for others. In these experiments the instance-specific BDD-based approach out-performed the parameter-specific cube-list approach for 10 out of the 13 benchmarks. These correspond loosely to the smaller of the benchmarks, in terms of numbers of inputs. BDD node-count does not, however, seem to be a factor.

V. CONCLUSION & FUTURE WORK

This paper reports the results of implementing the computation of a Boolean function's ac transform. Two configurable hardware approaches are used and are compared to a software implementation.

Our results show that the use of configurable hardware can provide some speed-up in the computation of this problem. Further work is required to identify which approach, instance-specific or parameter-specific, would be most beneficial and furthermore, which underlying algorithm is best suited to compute the solution. We have also demonstrated that the addition of parallelism in each approach can lead to a speed-up of the computation, but that there is a limited amount that may be added beyond which the additional complexity of the circuit can outweigh the advantage of the added circuitry.

These tests were restricted to fairly small functions; however, the instance-specific approach is limited only by the size of the BDD for the logic function. The parameter-specific approach is currently limited by a 32 bit word size for storing cubes, but this could be modified to allow larger word sizes, and if necessary, multiple downloads to the daughterboard's SRAM from the host computer. Thus neither of these current limitations present a major drawback to the hardware techniques.

The major drawback, at least in the case of the instance-specific approach, is the time required to generate an instance circuit and then to configure the target device. This is a drawback of any instance-specific approach, and requires that the instance of the application being targeted be utilised often enough that the overhead is offset by the time saved in using a hardware approach. The parameter-specific circuit still requires this overhead, but the additional time requirements are further offset due to the fact that more instances can be solved without having to configure the device.

An interesting line of research started in this paper is the development of heuristics to predict the optimal amount of parallelism to introduce to a instance-specifc circuit. Some knowledge of the solution design and the target environment is required, as shown by the improvements in Table II. With such knowledge the approach of circuit generation can be optimally applied to other application problems.

REFERENCES

[1] S. Aborhey. Autocorrelation Testing of Combinational Circuits. *Computers and Digital Techniques, IEE Proceedings E*, 136(1):57–61, Jan. 1989.

[2] R. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. on Comp.*, C-35(8):677–691, Aug. 1986.

[3] M. Karpovsky. *Finite Orthogonal Series in the Design of Digital Devices*. John Wiley & Sons, 1976.

[4] H. Lee and F. Ercal. RMESH Algorithms For Parallel String Matching. In *Proceedings of the 3rd International Symposium on Parallel Architectures, Algorithms and Networks (I-SPAN'97)*, pages 223–226, 1997.

[5] J. Rice and K. Kent. Using Instance-Specific Circuits to Compute Autocorrelation Coefficients. In *Proceedings of the First Annual Northeast Workshop on Circuits and Systems (NEWCAS)*, 2003.

[6] J. E. Rice and J. C. Muzio. Methods for Calculating Autocorrelation Coefficients. In *Proceedings of the 4th International Workshop on Boolean Problems, (IWSB P2000)*, pages 69–76, 2000.

[7] J. E. Rice, J. C. Muzio, and M. Serra. The Use of Autocorrelation Coefficients for Variable Ordering for ROBDDs. In *Proceedings of the 4th International Workshop on Applications of the Reed-Müller Expansion in Circuit Design (RM99)*, 1999.

[8] M. Serra and K. Kent. Using FPGAs to Solve the Hamiltonian Cycle Problem. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, 2003.