Published in IET Computers & Digital Techniques
Received on 16th August 2007
Revised on 28th February 2008
doi: 10.1049/iet-cdt:20070120

ISSN 1751-8601

SC Build: a computer-aided design tool for design space exploration of embedded central processing unit cores for field-programmable gate arrays

I.D.L. Anderson M.A.S. Khalid

Department of Electrical and Computer Engineering, Research Centre for Integrated Microsystems (RCIM), University of Windsor, Windsor, Ont., Canada
E-mail: mkhalid@uwindsor.ca

Abstract: A genetic algorithm-based design space exploration technique using parameterised cores is examined. A computer-aided design tool called SCBuild was developed which is capable of applying a genetic algorithm to a core's parameters, and generating hardware description language models of core variants. The tool can also compute estimates of a variant's area and critical path delay on a field-programmable gate array. Using this tool, several experiments were conducted using a soft-core processor with a large design space. It was concluded from these experiments that using a genetic algorithm to explore the design space of a parameterised core can help a designer make intelligent decisions regarding the assignment of values to the parameters of an embedded hardware platform.

1 Introduction

In our modern digital age, devices utilising embedded systems have become very common and enjoy widespread use in our daily lives. An embedded system is an electronic sub-system that utilises computational hardware to perform a small set of tasks that are specific to a particular application [1]. They can be logically broken down into two major components: the embedded software and the digital hardware. The hardware component usually consists of one or more embedded central processing units (CPUs) and their associated application-specific hardware, whereas the software component is a program written and compiled specifically to run on the embedded processors.

As the complexity of embedded system designs increased over time, designing each and every hardware component of the system from scratch soon became far too impractical and expensive for many designers. To meet these challenges, the platform-based design approach [2–4] emerged. Platform-based design entails the idea of using pre-designed and pre-tested hardware components known as intellectual property (IP) cores as a platform upon which to build complete systems.

Soft-cores are a particular class of hardware IP cores that are often used by designers to build their systems. Soft-cores are hardware components which are described using a hardware description language (HDL). In order to increase the reusability of soft-core platforms across a wider range of application domains, many of them are parameterised, meaning that the core's architecture features a number of configurable parameters that can be set by the engineer at design time. These options allow the designer of an embedded system to tailor the core to closely match the requirements of the system's intended application.

When designing an embedded system for any application, it is crucial that a good hardware platform with a suitable combination of parameter values be selected early in the design process. This task is essentially a multi-objective optimisation problem in which design configurations are

doi: 10.1049/iet-cdt:20070120

chosen so that they provide the best balance between a set of competing objectives, most commonly the optimisation of the chip area utilisation, power consumption and performance of the system. This endeavour is complicated by the fact that the design space for any given system is usually very vast. Therefore much research has been conducted into automating the process of design space exploration (DSE) [5]. One widely used automated approach involves applying an exploration strategy based on genetic algorithms, which are a class of optimisation algorithms inspired by the field of biological sciences and which have been found to be effective in solving multi-objective optimisation problems.

This paper is concerned with the question of how to derive a 'good' hardware platform for a given embedded system constructed from a set of parameterised soft-core components. The emphasis of this work is on the design of embedded microprocessors targeted for implementation on field programmable gate arrays (FPGAs) specifically. In this work, the design of a novel software-based computer-aided design (CAD) tool, called SCBuild ('soft-core build') [6], is described. This tool is capable of exploring the design space of a parameterised soft-core in search of the Pareto-optimal set of configurations using a genetic algorithm-based approach, and is able to generate synthesisable VHDL descriptions of instances of a core given specific parameter values. Several DSE experiments are conducted using this tool, and the results obtained from these experiments are discussed.

The outline of this paper is as follows. Section 2 provides a summary of some of the previous work that has been done by other researchers in this area of study. In Section 3, the design and implementation of SCBuild are discussed in detail. Section 4 presents the results obtained through experimentation using SCBuild and a simple parameterised reduced instruction set computer (RISC) processor. Finally, Section 5 concludes this paper and discusses possible future work in this area.

2 Previous work

Yiannacouras [7, 8] developed SPREE, the Soft Processor Rapid Exploration Environment, in order to facilitate the exploration of the design space for soft-core processors targeted for implementation on an FPGA. SPREE consists of a hardware component library and an register transfer level (RTL) generator. The RTL generator fetches hardware components from the library and builds a datapath according to special datapath and instruction set architecture descriptions which are given to the RTL generator as an input. The RTL generator then creates the corresponding control logic, either pipelined or unpipelined, yielding a complete soft-core processor. The generated processors were based on the MIPS-I [9] instruction set architecture. The SPREE system was used to investigate several soft-core processor architectural alternatives including hardware against software multiplication, different shifter implementations, varying pipeline depths and organisations, as well as some other interesting architectural tradeoffs. The generated processors were evaluated using a set of benchmark applications and compared against several Altera Nios II [10] implementations. One major difference between the SPREE system and this present work is the exploration methodology. The SPREE system uses an exhaustive exploration strategy, in which the user must manually explore the various design tradeoffs by developing different architectural descriptions for each processor variant. In contrast, this research applies an automated approach based on a genetic algorithm to explore the design space of a heavily parameterised soft-core description.

Sheldon *et al.* [11] present a methodology for customising FPGA-based soft-core processors for a specific application. They customise a Xilinx MicroBlaze soft-core processor with five parameters: optional multiplier, barrel shifter, divider, floating-point unit and data cache support, yielding a total of 32 configurations. A set of 11 EEMBC [12] benchmarks, plus one additional custom-designed benchmark application, were chosen for experimentation. Performance was measured by the runtime of a benchmark application on the processor, and size was measured as the number of equivalent look-up table (LUTs), the weighted sum of the number of LUTs, multipliers and RAM blocks used by the processor on a Xilinx FPGA. The authors employed two approaches to explore the design space. The first approach involved characterising the soft-core DSE problem as a 0–1 knapsack problem, which they then solve using a dynamic programming algorithm [13]. First, the effects of each of the five parameters on the area and runtime were studied by synthesising the processor with each option separately and running the benchmark on the processors. Then a dynamic programming algorithm was applied to determine the optimal configuration. The second approach used real synthesis and application execution data during exploration and a greedy search strategy. The impact of each parameter on the area and runtime was first predeterimined by performing a synthesis/execution run for each parameter separately. Then, each of the parameters was searched in sequence, starting from the most influential to down to the least. If a parameter improved on the application runtime while meeting area constraints, then the parameter value was selected. The drawback of these methodologies is that only a small design space can be explored in a reasonable amount of time, thus making it infeasible for larger systems with many parameters. By contrast, this research utilises a genetic algorithm-based approach which allows for a much larger design space – on the order of millions of design configurations – to be explored in a relatively short amount of time.

The Platune system [14] is an environment that allows an embedded system designer to tune the parameters of a parameterised hardware platform. The system provides a set of simulation and power models for the components of a parameterised system consisting of a MIPS R3000 processor

[10] with instruction and data caches, on-chip memory and a set of interconnecting buses. The simulation models are used to compute the execution time of a specific program running on the processor and to gather information on the power consumption of the system. Platune also features a DSE framework that uses a parameter interdependency model and an exhaustive approach to determine the Pareto-optimal set of configurations. A graph of parameter interdependencies is created, and interdependent parameters are gathered together into clusters. Then an exhaustive exploration approach is applied to each cluster to determine its local Pareto-optimal set. Once each cluster has been searched, pairs of clusters are merged together and the exhaustive search is applied to the merged clusters. This process continues until a single cluster remains, and the Pareto-optimal set of configurations is determined. In contrast to the Platune system, this work features a GA-based approach to search the design space for the Pareto-optimal set. Additionally, the Platune system is directed specifically towards the use of the MIPS R3000 processor model, which is not designed specifically for FPGA implementation. In this research, a general framework for the DSE of any parameterised core has been established, with emphasis on soft-cores targeted for implementation on an FPGA.

Palesi and Givargis [15] present an approach to explore the design space of heavily parameterised systems using a genetic algorithm, namely the Strength Pareto Evolutionary Algorithm 2 (SPEA2) [16]. The approach combines the parameter dependency clustering and exhaustive search method used by Platune with the genetic-based SPEA2 algorithm to reduce the time needed to find the Pareto-optimal set of configurations. Their results indicate that an approximated Pareto-optimal set that is within 1% of the actual set can be obtained using the combined approach while reducing the amount of simulation time required to determine the set by 80%. Ascia *et al.* [17, 18] later use the SPEA2 algorithm directly with the Platune system to search for the Pareto-optimal set of configurations. These researchers provide some useful conclusions about the use of genetic-based algorithms in platform-based design problems. However, they do not focus specifically on the exploration of the design space of soft IP cores targeted for FPGAs.

The PEAS-III system by Itoh *et al.* [19] is a system-level design environment that enables designers to quickly explore the design space of pipelined embedded processors. The system is based on the micro-operation description of instructions, which allows designers to concentrate on the design of a processor's instruction set. A pipelined processor is built from a series of pipeline stage models. Each stage model represents a single stage in the pipeline and consists of pipeline resources such as arithmetic logic units (ALU) and other functional units, inter-stage pipeline registers, a stage controller and the interconnections between them. The PEAS-III system creates a datapath and associated control logic by cascading the stage models in series. Two VHDL descriptions of the processor are generated by the system: a non-synthesisable model used purely for simulation and a version intended for synthesis. In order to evaluate the effectiveness of PEAS-III, several processors were built using the system, including a MIPS R3000 processor, a DLX processor [20] and a simple RISC controller. Later, Kitajima *et al.* [21] designed a complex instruction set computer processor with a deep pipeline using the PEAS-III system, and in 2003, Kobayashi *et al.* [22] used PEAS-III to implement a JPEG encoder. The PEAS-III system does not utilise any form of automated DSE, thus distinguishing it from this present research.

3 SCBuild

3.1 SCBuild system environment

A diagram of the SCBuild system environment is shown in Fig. 1. SCBuild is a software-based CAD tool which facilitates the rapid exploration of the design space of parameterised soft-core components. This tool accepts a template description of a parameterised core, which is a blueprint for the core that contains information on its parameters and tells the tool how to generate HDL code for the core given certain parameter values. It uses the simple evolutionary algorithm for multi-objective optimisation (SEAMO) [23], a genetic-based algorithm, to prune the design space of the parameterised core and determine an approximation of its Pareto-optimal set of configurations. An estimate of each configuration's objective values, such as its FPGA logic resource utilisation and critical path delay, are also computed by the program. Additionally, SCBuild can generate synthesisable VHDL descriptions of core instances with selected features by instantiating components from the VHDL component library. It is also able to generate a Tool Command Language (Tcl) [24] script file and invoke Altera's Quartus II software [25] to run the script, creating a new quartus project file (.qpf), compiling the generated VHDL code and saving the synthesis results in a text file for later processing.

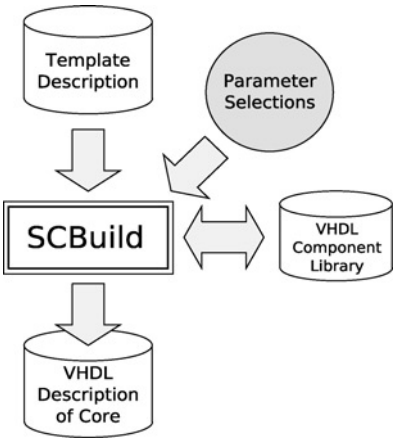

Figure 1 *SCBuild system environment*

doi: 10.1049/iet-cdt:20070120

3.2 CAD flow for SCBuild

The SCBuild CAD flow is depicted in Fig. 2. Each step in the flowchart will be discussed in the sections that follow. For additional information, see [6].

3.2.1 Design entry: The first step in the CAD flow for SCBuild is design entry. At this initial stage, a template description for a parameterised core is created by the end-user. At present, this template description is created manually by the template designer, although in future research work, an additional software tool may assist the designer in creating this description.

As its name implies, the SCBuild template description serves as a template or blueprint for a parameterised core from which many different variant cores can be generated given a set of parameter values. The complete template description for a parameterised core is made up of a set of text files containing extensible markup language (XML) [26] code describing the set of template components which make up the core. This description contains information on the core's parameters, the hierarchy of sub-components that make up the core, the connectivity of these sub-components, the set of possible physical implementations that a sub-component may have and the ways in which the core's parameters affect its underlying structure.

3.2.2 Collect system-level parameters: One of the template component files is then chosen as the top-level entity of the core, and is denoted as 'system' component. This file is read and all of the parameter information is collected and stored in a data structure internal to SCBuild. All of these parameters then become visible to the user. If a core with a specific set of features is required, then any or all of the system-level parameters can be set and locked to their desired values. Parameters that are not locked are considered 'free' and will be used to explore the design space of the parameterised core, whereas those that are locked will be considered constant during the process of automated DSE.

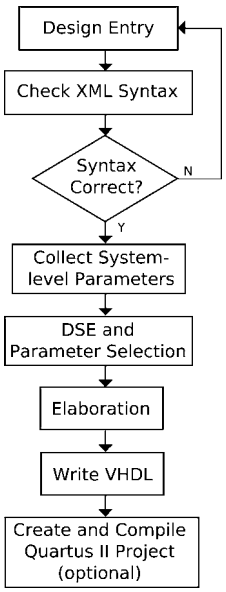

Figure 2 *SCBuild CAD flow*

3.2.3 DSE and parameter selection: In this step, the SEAMO genetic algorithm is applied to the free parameters of the system, resulting in a set of configurations that approximates the Pareto-optimal set. The algorithm as it is applied by SCBuild is briefly summarised as follows: each system-level parameter is represented as a gene – a discrete variable with a finite set of possible integer values. The integer values correspond to the possible system-level parameter values. The chromosome is simply a string of genes.

The first step of the algorithm is to generate an initial population of N chromosomes randomly. After this is done, each chromosome is evaluated individually one-by-one in terms of its objectives. The method used for determining these values will be discussed below. The estimated objective values for area and delay are stored separately in an 'objective vector' for each chromosome. Once every chromosome has been evaluated, the 'best-so-far' values for area and for delay are recorded.

Then, each member in the population is paired with another randomly selected member from the population and is given a chance to produce an offspring using the crossover operator. The crossover operator selects a cut-point at random and combines the left half of one parent with the right half of the other. The crossover operator is only a certain percentage of the time, according to the specified crossover rate, r_c. The offspring is then mutated, which involves randomly selecting one gene within the offspring chromosome and changing it to another random value. Only a certain percentage of the offspring produced are mutated, the proportion of which is determined by the mutation rate, r_m.

At this point, the mutated offspring is evaluated in terms of its objectives and replaces one of the parents if one of the several conditions is met. If one of the parents is dominated by the offspring (i.e. is inferior to the offspring across all of the objectives), then the dominated parent is replaced by the offspring. If the offspring improves on one or more of the best-so-far values, then the offspring replaces one of its parents (the parent to be replaced is randomly chosen). Finally, if an identical copy of the offspring already exists within the population, then the offspring is discarded.

After the algorithm has visited each member of the population and paired it with another to produce offspring, one generation has passed. The algorithm will generally iterate through a number of generations before the population converges towards an approximation of the Pareto-optimal set of configurations. The size of the population, N, and the number of generations, G, the crossover rate r_c and the mutation rate, r_m, constitute the parameters of the algorithm

and appropriate values for them are often determined through experimentation.

Prior to running the algorithm, the user must specify the population size to use, the number of generations for which to run the algorithm and the crossover and mutation rates. Once the algorithm has run its course and the final set of configurations has been determined, the user selects one of these configurations as the final configuration, and all of the system-level parameters are set and locked to these values.

SCBuild uses the objective estimation approach proposed by Jha and Dutt [27] to evaluate configurations during the execution of the SEAMO algorithm. This approach involves establishing fast equations for estimating area and critical path delay – the two objectives used in this research – using least-squares regression analysis on actual synthesis data for a number of representative configurations. These equations relate the area and delay objectives to the P (the total number of parameters) parameters of the system, p_1, p_2, …, p_P and have the general form

$$F_k(p_1, p_2, \ldots, p_P) = a_{0,k} + \sum_{i=1}^{P} (a_{i,k} \cdot f_{i,k}(p_i)) \qquad (1)$$

where $a_{0,k}$, $a_{1,k}$, …, $a_{P,k}$ are the constant coefficients determined using regression analysis and the functions $f_{i,k}(p_i)$ approximate the relationships between the parameters p_i and the objective values (in this case, FPGA area utilisation and delay). The area of the circuit is given in terms of equivalent logic elements (LE) used on a Stratix EP1S40F780C5 FPGA [28]. This value is determined by summing the number of LEs, DSP blocks (9 × 9 bit), M512s, M4Ks and M-RAM memory blocks multiplied by their relative sizes on the Stratix FPGA [7]. Delay is a measure of the critical path delay of the circuit in nanoseconds (ns) reported by the Quartus II Timing Analyzer [29].

3.2.4 Elaboration: After a configuration is selected, SCBuild proceeds with the most important step of the CAD flow: elaboration. At this stage, a final VHDL model of a given core with the set of features specified by the user-selected parameter values is constructed. At this step, SCBuild processes the input template description files and uses the assigned parameter values to build the final VHDL description of a core instance. SCBuild instantiates components from the VHDL component library and connects their ports together as specified in the template description.

3.2.5 Quartus II project creation and compilation: SCBuild can optionally generate a simple Tcl [24] script file that directs the Altera Quartus II software tool to create a new project file, take the newly generated set of VHDL files and perform a complete compilation including analysis and synthesis, fitting, assembly and timing analysis and write the pertinent compilation report information into a text file that can be subsequently read back by SCBuild for later use.

3.3 The VHDL component library

The SCBuild VHDL component library is an open-ended collection of pre-designed and pre-tested soft-core components described in VHDL that can be used to build more complex designs using SCBuild. The library can be expanded limitlessly by adding new hardware components to it. For the purposes of this research, the VHDL component library consists of the hardware modules necessary to construct a simple pipelined RISC processor, as will be discussed later in Section 4.1.

4 Experiments

In this section, the results of several DSE experiments are presented. For these experiments, a template description model of a simple parameterised pipelined RISC processor core was created. Then the SCBuild CAD tool was used to explore the design space of the processor core and generate and compile a number of variant implementations.

4.1 Target core

The parameterised RISC processor template used in this research is a modified version of the pipelined RISC CPU presented by Mano and Kime [30]. The core is a simple microprocessor with a load-store architecture consisting of a datapath, a control unit and separate data and instruction memories. It features a total of 38 instructions, including instructions for performing arithmetic, logical, shift, branch and memory operations with integer data.

In order to facilitate experimentation, a set of parameters was added to the processor, a template description model of the core was created and a library of VHDL building-block components was constructed. The core features the parameters listed in Table 1. Given these parameters, there are a total of exactly 1 672 704 possible configurations for this core.

4.2 Establishing the objective estimation equations

In order to establish a set of objective estimation equations (1) for the RISC processor core, it was first necessary to synthesise a set of configurations that are representative of core's design space. To this end, a parameter sweep was performed on each of the core's 11 parameters. Starting from a base configuration (in which all of the parameter values are set to 1), each of the core's parameters was varied across their entire range of values, whereas the other parameters were held constant at their base values. This yielded a total of 41 'sweep' configurations, each of which was generated by SCBuild and compiled using Quartus II [29]. All of these configurations were targeted for the Altera Stratix EP1S40F780C5 FPGA [28] and were compiled using the default compiler settings. The following pieces of information were collected from the Quartus II compilation reports for each configuration: the number of LEs, DSP blocks, M512, M4K and M-RAM memory

IET Comput. Digit. Tech., 2009, Vol. 3, No. 1, pp. 24–32
doi: 10.1049/iet-cdt:20070120

Table 1 RISC processor hardware parameters

Parameter	Possible values
ALU adder implementation (p_1)	ripple-carry, carry-lookahead
arithmetic shifter implementation (p_2)	none, basic, barrel
branch adder implementation (p_3)	ripple-carry, carry-lookahead
data address width (p_4)	5 to 15 bits
data width (p_5)	8, 16, 32, 64 bits
include multiplier (p_6)	false, true
instruction address width (p_7)	5 to 15 bits
logical shifter implementation (p_8)	none, basic, barrel
operand width (p_9)	2 to 9 bits
pipelined (p_{10})	false, true
rotator implementation (p_{11})	none, basic, barrel

blocks used by the configuration, and the critical path delay of the processor in nanoseconds reported by the Timing Analyzer.

A second set of sweep configurations was also produced. In order to study the relationship between the data width parameter (p_5) and the ten remaining parameters of the core, an additional 105 sweep configurations were generated and compiled. In total, 146 different sweep configurations were generated and compiled successfully. These configurations served as the basis for establishing area and delay objective estimation equations for the RISC processor.

Some trial and error was necessary to determine the exact forms for all of the functions $f_{i,k}(p_i)$ for each parameter. After these functions were determined, P-dimensional regression analysis was applied to the parameter sweep data in order to compute the $a_{i,k}$ coefficients in (1).

4.3 Testing the objective estimation equations

The accuracy of the area and delay estimation equations was first tested using the 146 parameter sweep configurations used to establish the equations. The 'actual' values obtained from compilation data collected from Quartus II were compared with the estimated values that were computed using the established objective estimation equations. Graphs comparing the actual values to the estimated values of area and critical path delay for the 146 sweep configurations are shown in Fig. 3. As can be seen in Fig. 3*a*, the estimated area values correlate well with the actual values, showing an average percentage error of 10.1%. The delay estimates also track the actual values reasonably well, with an average percentage error of 11.7%, as shown in Fig. 3*b*.

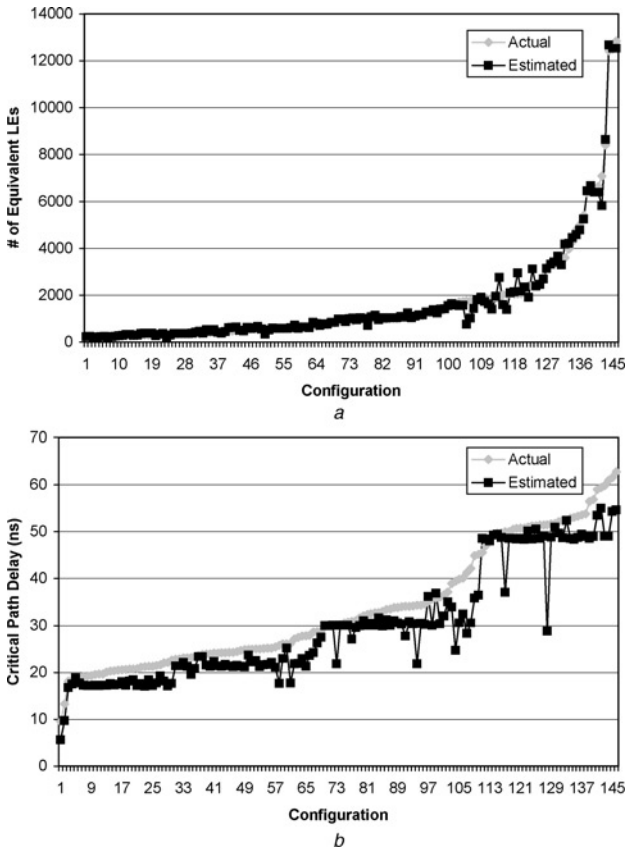

Figure 3 *Actual and estimated values for sweep configurations*

a Area
b Delay

Next, the equations were tested to determine their accuracy for any arbitrary configuration. For this test, a set of 41 individuals was randomly generated and compiled with Quartus II. Again, the compilation data obtained for these

41 configurations were compared with the approximated values that were determined using the estimation equations. Graphs comparing the actual and estimated values for area and delay are shown in Fig. 4. As can be seen from the figure, the area equation provides better overall estimates than does the delay equation; however, they are both still within reasonable tolerances. The average error was 13.3% for the area estimates and 16.4% for the delay estimates.

These experiments serve to demonstrate the inherent difficulty with estimating the critical path delay of an arbitrary parameterised core. This difficulty is due to the fact that a number of different factors affect the critical path delay of a core, including the implementation of the core's underlying components, the placement of the circuit on the FPGA, the routing running between the various parts of the core and so on. By contrast, the FPGA area utilisation of a core is easier to estimate accurately, because many of the parameters affect the total area of a core in a regular and predictable manner.

Another comment that can be made about these results is that a tradeoff exists between the accuracy of the estimated objective values and the amount of computation required to obtain those values. In general, more accurate estimations can be made at the expense of longer computation times. The goal of the regression-based objective estimation technique used in this research is to produce a set of estimation equations that can be evaluated quickly and easily. However, the light computation workload required by this method does come at the cost of reduced estimation accuracy. In addition, an up-front investment of time and effort is also necessary to produce enough real sythesis data in order to establish these equations. In future work, different objective estimation techniques may be applied in order to increase the accuracy of the estimates and remove the need to generate a set of sweep configurations.

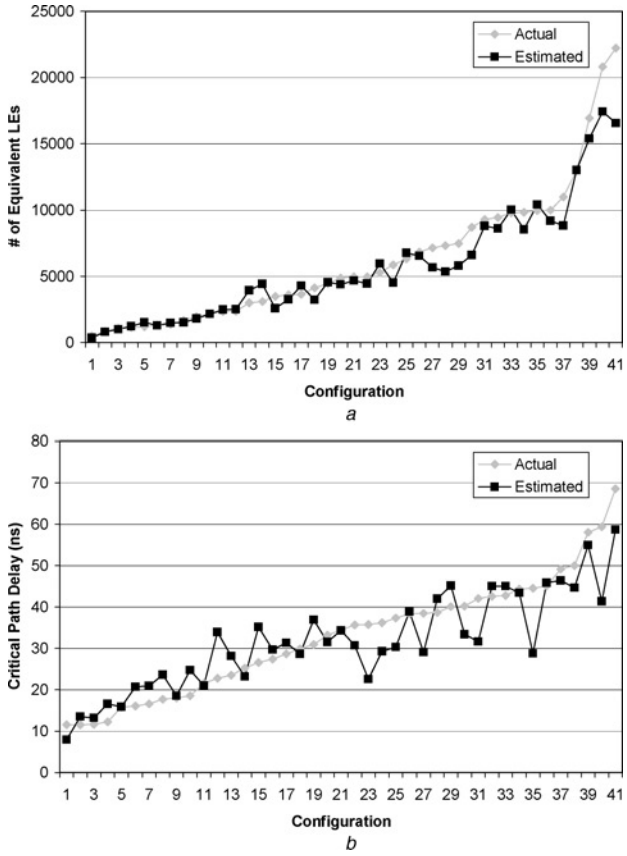

Figure 4 *Actual and estimated values for random configurations*

a Area
b Delay

4.4 Design space exploration

The SEAMO algorithm was applied to a population of randomly generated configurations in order to determine an approximation of the Pareto-optimal set. In this section, the results of this experiment are presented.

4.4.1 Algorithm parameters: The set of 50 random configurations was used as the initial population for this exploration experiment. Suitable values of the crossover and mutation rates were determined experimentally. The crossover and mutation rates were both varied between 0.1 and 1.0, and the resulting evolved population was observed. It was determined that a value of 0.1 for the crossover rate and 0.2 for the mutation rate provided the greatest diversity of configurations in the evolved population. Finally, the number of generations of the algorithm was set to 20.

4.4.2 Results: Using the algorithm parameters mentioned above, the SEAMO algorithm was applied to the initial population of 50 configurations. After 20 generations of the algorithm, the population began to converge towards an

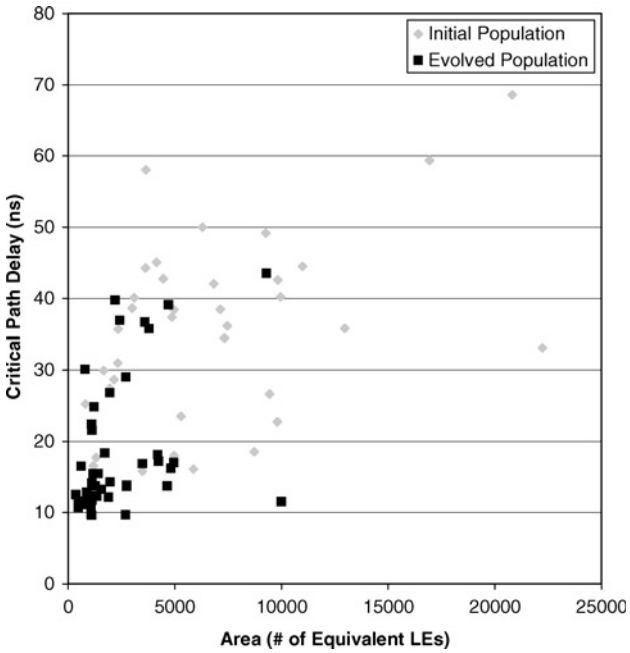

Figure 5 *Initial and evolved populations*

IET Comput. Digit. Tech., 2009, Vol. 3, No. 1, pp. 24–32
doi: 10.1049/iet-cdt:20070120

approximation of the Pareto-optimal set. Each of the configurations in the resulting evolved population was compiled using Quartus II. A visual comparison of the initial and the evolved populations (using the actual values collected from the compilation reports), is shown in Fig. 5.

As can be seen in the figure, the majority of the configurations in the evolved population tend to cluster around the lower-left corner of the design space, approximating the Pareto-optimal front. The variability in the evolved population is due to the inaccuracies present in the objective estimation equations used to evaluate each configuration. If more accurate estimation methods were used, then the evolved population would form a smoother curve along the lower-left boundary of the design space.

5 Conclusions and future work

These experiments have shown that utilising a genetic-based approach to prune the design space of a parameterised core can be helpful in assisting a designer making decisions regarding the selection of parameter values for a parameterised hardware platform. Coupled with an accurate configuration evaluation methodology, the genetic algorithm helps to eliminate the sub-optimal configurations from consideration, yielding a much smaller set of configurations from which to choose. Designers can then choose one configuration from the pruned set that satisfies their application-specific design constraints.

In future work, a number of interesting issues may be explored. SCBuild could be extended so that it is able to automate a greater number of design tasks, including the generation of control logic and automatic compilation of cores using Tcl scripts. More accurate objective estimation techniques could be examined, thus improving the quality of the DSE results. High-level synthesis support may be another significant addition. Finally, a broader exploration of the soft-core processor design space targeting FPGAs can be conducted. A wider variety of architectural features can be examined, including cache sub-systems, branch prediction schemes, floating-point support, various pipeline architectures, custom instruction support, functional unit implementations, interrupt and exception handling and others. Also, parameterised multi-processor hardware platforms and the various accompanying implementation issues would prove to be a useful course of study.

6 References

[1] JERRAYA A.A.: 'Long term trends for embedded system design'. Proc. EUROMICRO Systems Digital System Design (DSD'04), Rennes, France, 31 August–3 September 2004, pp. 20–26

[2] CHANG H., COOKE L., HUNT M., MARTIN G., MCNELLY A., TODD L.: 'Surviving the SOC revolution: a guide to platform-based design' (Kluwer, Norwell, MA, USA, April 1999)

[3] MARTIN G., BRUNEL J.-Y.: 'Platform-based co-design and co-development: experience, methodology and trends'. Proc. 9th IEEE/DATC Electronic Design Processes Workshop, April 2002

[4] POP P., ELES P., PENG Z.: 'Analysis and synthesis of distributed real-time embedded systems' (Kluwer Academic Publishers, Boston, Dordrecht, London, 2004)

[5] GRIES M.: 'Methods for evaluating and covering the design space early in design development', RFC UCB/ERL MO3/32, Electronics Research Lab, University of California at Berkeley, August 2003

[6] ANDERSON I.D.L.: 'A CAD tool for design space exploration of embedded CPU cores for FPGAs', Master's Thesis, University of Windsor Windsor, Ontario, Canada, 2007

[7] YIANNACOURAS P.: 'The microarchitecture of FPGA-based soft processors', Master's Thesis, University of Toronto Toronto, Ontario, Canada, 2005

[8] YIANNACOURAS P., STEFFAN J.G., ROSE J.: 'Exploration and customization of FPGA-based soft processors', *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 2007, **26**, (2), pp. 266–277

[9] HENNESSY J.L., JOUPPI N.P., GILL J., *ET AL.*: 'The MIPS machine'. COMPCON, 1982, pp. 2–7

[10] Nios II, available at: http://www.altera.com/products/ip/processors/nios2/ni2-index.html, January 2007

[11] SHELDON D., KUMAR R., LYSECKY R., VAHID F., TULLSEN D.: 'Application-specific customization of parameterized FPGA soft-core processors'. Proc. IEEE/ACM Int. Conf. Computer-Aided Design, 2006, San Jose, CA, USA, November 2006, pp. 261–268

[12] The Embedded Microprocessor Benchmark Consortium. Available at: http://www.eembc.org/, March 2007

[13] TOTH P.: 'Dynamic programming algorithms for the zero-one knapsack problem', *Computing*, 1980, **25**, pp. 29–45

[14] GIVARGIS T., VAHID F.: 'Platune: a tuning framework for system-on-a-chip platforms', *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 2002, **21**, (11), pp. 1317–1327

[15] PALESI M., GIVARGIS M.: 'Multi-objective design space exploration using genetic algorithms'. Proc. 10th Int. Symp. Hardware/Software Codesign, 2002 (CODES 2002), Estes Park, Colorado, USA, 6–8 May 2002, pp. 67–72

[16] ZITZLER E., LAUMANNS M., THIELE L.: 'SPEA2: improving the strength Pareto evolutionary algorithm', Technical Report Computer Engineering and Communication Networks Lab, Swiss Federal Institute of Technology (ETH) Zurich Gloriastrasse, CH-8092, May 2001

[17] ASCIA G., CATANIA V., PALESI M.: 'A GA-based design space exploration framework for parameterized system-on-a-chip platforms', *IEEE Trans. Evol. Comput.*, 2004, **8**, (4), pp. 329–345

[18] ASCIA G., CATANIA V., PALESI M.: 'A multiobjective genetic approach for system-level exploration in parameterized systems-on-a-chip', *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 2005, **24**, (5), pp. 635–645

[19] ITOH M., HIGAKI S., SATO J., *ET AL.*: 'PEAS-III: an ASIP design environment'. Proc. Int. Conf. Computer Design, Austin, TX, USA, September 2000, pp. 430–436

[20] HENNESSY J.L., PATTERSON D.A.: 'Computer architecture: a quantitative approach' (Morgan Kaufmann, San Fransisco, CA, USA, September 2006, 4th edn.)

[21] KITAJIMA A., SASAKI T., TAKEUCHI Y., IMAI M.: 'Design of application specific CISC using PEAS-III'. Proc. 13th IEEE Int. Workshop on Rapid System Prototyping (RSP'02), Darmstadt, Germany, July 2002, pp. 12–19

[22] KOBAYASHI S., MITA K., TAKEUCHI Y., IMAI M.: 'Rapid prototyping of JPEG encoder using the ASIP development system: PEAS-III'. Proc. Int. Conf. Multimedia and Expo, 2003 (ICME'03), vol. 1, Baltimore, MD, USA, 6–9 July 2003, pp. I-149–152

[23] VALENZUELA C.L.: 'A simple evolutionary algorithm for multi-objective optimization (SEAMO)'. Proc. 2002 Congress on Evolutionary Computation (CEC'02), vol. 1, Honolulu, HI, USA, 12–17 May 2002, pp. 717–722

[24] Tcl Developer Xchange, available at: http://www.tcl.tk/, January 2007

[25] Altera Corporation: Quartus II software, available at: http://www.altera.com/products/software/products/quartus2/qts-index.html, January 2007

[26] Extensible Markup Language (XML), available at: http://www.w3.org/XML/, January 2007

[27] JHA P.K., DUTT N.D.: 'Rapid estimation for parameterized components in high-level synthesis', *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 1993, **1**, (3), pp. 296–303

[28] Altera Corporation: 'Stratix device handbook', (July 2005)

[29] Altera Corporation: 'Quartus II version 5.0 handbook', Version 5.0.0, May 2005

[30] MANO M.M., KIME C.R.: 'Logic and computer design fundamentals' (Prentice Hall, Upper Saddle River, NJ, USA, 2001, 2nd edn. updated)

doi: 10.1049/iet-cdt:20070120