A Heuristic Algorithm for Deriving Compact Models of Processor Instruction Sets

Alessandro de Gennaro
School of Electrical and Electronic Engineering
Newcastle University
Newcastle upon Tyne, United Kingdom
Email: a.de-gennaro@ncl.ac.uk

Paulius Stankaitis
School of Computing Science
Newcastle University
Newcastle upon Tyne, United Kingdom
Email: paulius.stankaitis@ncl.ac.uk

Andrey Mokhov
School of Electrical and Electronic Engineering
Newcastle University
Newcastle upon Tyne, United Kingdom
Email: andrey.mokhov@ncl.ac.uk

***Abstract*—Finding a compact formal representation of a processor instruction set is important for easier comprehension by the designer, as well as for synthesis of an efficient hardware implementation of the processor's microcontroller.**

We present a new heuristic algorithm for deriving compact models of processor instruction sets. The algorithm is based on finding similarities between pairs of instructions and assigning similar opcodes (using a Hamming distance metric) to similar instructions (using a newly introduced instruction similarity metric). We demonstrate that this heuristic produces results with an average overhead, in terms of area, of 7.8% in comparison to the global optimum on the benchmarks we studied (subsets of instructions of ARM Cortex M0+, Texas Instruments MSP430 and Intel 8051 processors).

The algorithm is implemented as an open-source plugin for the Workcraft framework and is validated on a case study of a subset of 61 (out of 68) instructions of ARM Cortex M0+ processor. We compare the presented algorithm against a number of other available implementations.

I. Introduction

Over the years, aggressive transistor scaling led to immense microprocessor performance improvement. For instance, microprocessors running frequency has increased 100 times more than theoretically predicted. However, power consumption of chip has been increasing and has became a crucial progress constraint [1]. Moreover, as in recent decades mobile electronics have became more and more affordable to customers, device operation time became vital. Though, battery life has not increased significantly [2], thus other power optimisation approaches are being researched. Reduction of power consumption through instruction set optimisation approach has been known for a while. Nonetheless, it is still an active research area. In this field some important work has been done such as the introduction of a formalism called Conditional Partial Order Graphs [3], and a new instruction set architecture design approach based on that model [4][5]. Reduction of power (as well as area, latency or time to design) of Instruction Set Architecture (ISA), via an automated and rational approach, motivates further exploration of this recently introduced, though promising design approach.

A. Related work

Different approaches have been tried to handle the problem of Conditional Partial Order Graph encoding, targeting the minimisation of parameters such as area, latency of the derived controller which comes up with the synthesis of the representation.

In [6] the so called *Single-literal encoding* has been implemented. It finds the optimal encoding under the constraint that each condition can have at most one literal (positive or negative), it is based on graph colouring and works well in practice when the single-literal constraint is appropriate. Indeed, it does not fit well to instruction set architecture modelling, as it is likely that the opcodes generated by such an approach will be composed by a high number of variables. This parameter in fact might be constrained in this application, as in our case of study: ARM Cortex M0+ architecture has opcodes limited to no more than 8 bits.

In [4] instead, an algorithm which is suppose to find an optimal encoding using a SAT-solver as been developed, named *SAT-based encoding*. Even though it is able to seek an encoding on a fairly reduced number of literals in a short amount of time, it has many constraints which limit the usability of such approach. It doesn't scale well and cannot handle instruction sets with more than 12-15 instructions. The current implementation does not support branching instructions (which are fundamental for the description of more complex instructions behaviours), and finally it does not allow handling opcode constraints such as the definition of the number of variables the opcodes should be composed by, or the definition of any bit for reserved functions.

Another approach, which is not directly related to the Conditional Partial Order Graph model, but which may be extended to it, given the similarity of the problem, is the technique described in [7]. In it, an algorithm for optimally encoding Finite State Machine representation is presented.

In this work we present an algorithm to fill in the gaps described in this Section. We describe a technique able not only to optimise the final controller under different perspectives, but also to customise the final opcodes in order to tailor well the design process of instruction set architectures,

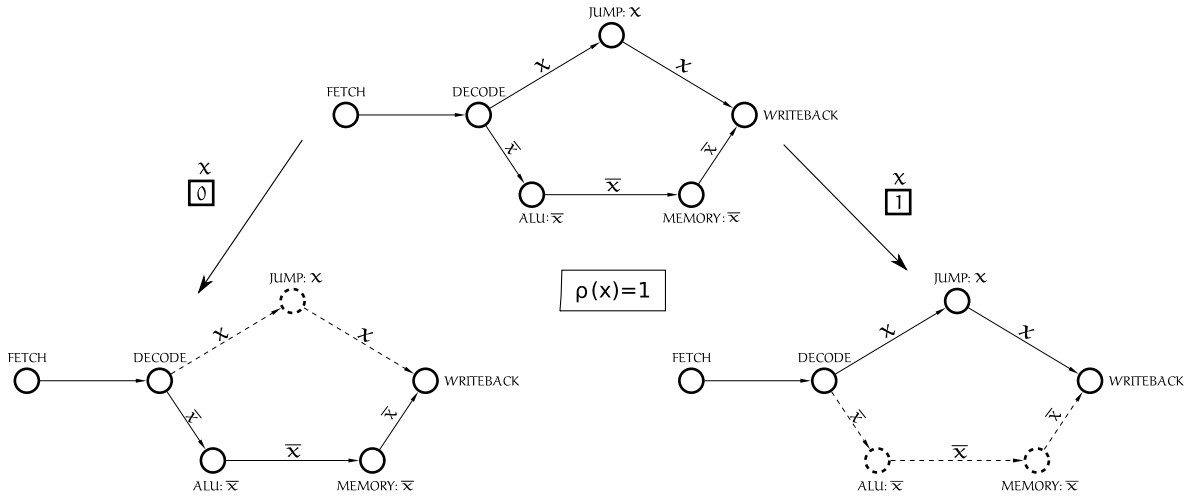

Fig. 1: Example of CPOG with 2 projections: $H|_{X=1}$ on the right, $H|_{X=0}$ on the left side

as well as many other applications this representation might be used to. In *Section II* we briefly introduce the model we have used, to carry out the research. The heuristic algorithm is deeply described in *Section III*. In *Section IV* the model of `ARMv6-M` ISA is shown and used as case of study for our encoding approach. In *Section V* the tool we have developed is presented and a set of benchmarks to evaluate the performance of the new technique is shown. *Section VI* concludes the paper pinpointing what has been achieved, and future work.

II. Background

Conditional Partial Order Graph [3][6][8] is a quintuple H = (V, E, X, ρ, ϕ):

- **V** is a set of vertices which correspond to events (or atomic actions) in a modelled system.
- **E** $\subseteq V \times V$ is a set of arcs representing dependencies between the events.
- Operational vector **X** is a set of Boolean variables. An opcode is an assignment $(x_1, x_2, ..., x_{|X|}) \in \{0,1\}^{|X|}$ of these variables. An opcode selects a particular partial order from those contained in the graph.
- $\boldsymbol{\rho} \in F(X)$ is a restriction function, where $F(X)$ is the set of all Boolean functions over variables in X. ρ defines the operational domain of the graph: X can be assigned only those opcodes $(x_1, x_2, ..., x_{|X|})$ which satisfy the restriction function, i.e. $\rho(x_1, x_2, ..., x_{|X|}) = 1$.
- Function $\boldsymbol{\phi} : (V \cup E) \rightarrow F(X)$ assigns a Boolean condition $\phi(z) \in F(X)$ to every vertex and arc $z \in V \cup E$ in the graph. Let us also define $\phi(z) \stackrel{df}{=} 0$ for $z \notin V \cup E$ for convenience.

This model is based on strict graphical representation, where each event is represented by a circle $\bigcirc$, and each connection between vertices, named as "arc", depicted as an arrow $\rightarrow$. Both the previous elements are labelled with a predefined pattern composed by $vertex/arc$ name, followed by condition $\phi(v/e)$. Next to each graph, a further condition is present called *restriction function* (ρ), composed by *operational variables* X.

An Example of CPOG is shown in Figure 1 with two possible projections at the bottom-side. The purpose of the condition ϕ is to switch on/off vertices and edges, when the conditions on it are satisfied or not respectively. In this representation, dash edges and circles represent nodes and arrows switched off, in such a way as not to affect the behaviour of that particular event class.

III. Encoding of different Partial Orders

The *encoding process* can be performed after all the partial orders have been created and optimised. It consists of associating a unique opcode to each graph in order to be distinguished by the other ones. This is the first step of the synthesis phase. Here, the controller for managing the whole structure is the main target. In order to better describe such problem, it may be worth giving some definitions of the terms used along this research.

An *opcode* points out an array of bits where each element can be a logic `0` or `1`, the following ones are examples of feasible opcodes $\{0010, 000000, 1110001, 0\}$. An *opcode ensemble* is defined as the group of opcodes it is possible to use with a particular number of bits, for instance the opcode ensemble of length 2 is composed by opcodes: $\{00, 01, 10, 11\}$ while the one of length 3 $\{000, 001, 010, 011, 100, 101, 110, 111\}$. *Encoding*, also referred to as a *solution*, is a subset of an opcode ensemble where each opcode is associated to one and only one *partial order* graph. Finally, the *solution space* is represented as the group of all the possible encodings for a particular CPOG.

In Figure 2, each empty circle represents a possible opcode which might be potentially employed to encode one partial order. A number n is present inside a circle when the partial order n has been encoded with the opcode associated to that particular circle. Each opcode, in this case, is composed by 3 bits.

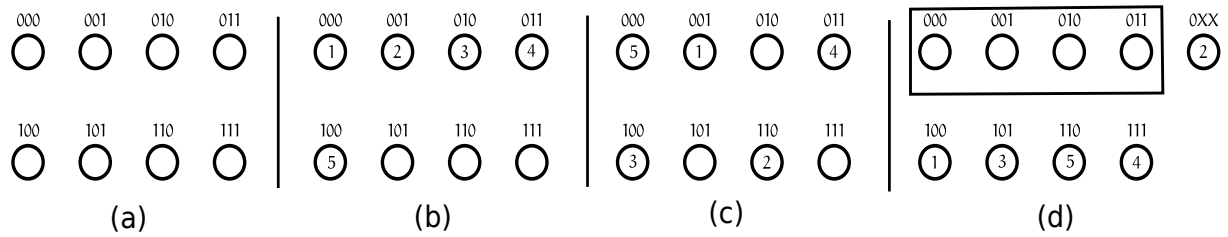

Fig. 2: Some examples of possible encodings for a 5 Partial orders model.

In Figure 2(a), the opcodes available to encode a CPOG that contains up to 8 partial orders are depicted. While in Figure 2(b) and 2(c), two possible solutions, for what concerns a CPOG composed by 5 partial orders, are shown. Designers may also exploit *Don't care* conditions when a group of opcodes might be employed to represent partial orders, as shown in Figure 2(d) for instance. Here, the second partial order is encoded by the opcode 0XX.

In light of the above, we want to demonstrate that the solution space might be really high depending on the number of graphs to encode and on the size of the opcode ensemble (neglecting the encodings including Don't cares conditions). The size of the solution space is of primary importance because potentially, each solution might be used to synthesise a different (with various optimisation degrees) controller, and should be therefore inspected. As a consequence, the higher the number of solutions, the harder it would be to seek the best possible encoding for a particular representation. This is the reason why one of the main concerns at this step is trying to reduce the size of the solution space as much as possible.

For instance, let us consider a very small *Conditional Partial Order Graph* representation, composed by 4 graphs only. As analysed in [6] and briefly above, there could be several approaches to encode various graphs in order to minimise the Boolean functions in each vertex and arc. What we are going to take into account hereby will be opcode ensembles with minimum length related to number of partial orders.

This means that in order to encode 4 different graphs, we need 2 bits (opcode ensemble of length 2). Therefore, in order to get the minimum number of bits needed to encode an entire model one needs to refer to Formula 1, where *k* stands for the number of partial orders which composes the representation.

$$\#\{\mathcal{B}\} = \lceil \log_2(k) \rceil \tag{1}$$

In this case, the number of solutions fits perfectly to the graphs to encode, as with two bits it is possible to encode exactly four elements ($2^2 = 4$). Additionally, in order to compute the entire solution space of such an instance we should take into account all the *permutations* (or *dispositions*, if $k < m$) of the opcode ensemble. Therefore, the total number of solutions one can select by arranging opcodes differently each time in a group of *m* elements is represented in Formula 2 by $\#\{\mathcal{S}\}$, where *k* is the number of graphs to encode.

$$\#\{\mathcal{S}\} = \frac{m!}{(m-k)!} \qquad \Rightarrow \qquad \#\{\mathcal{S}\}' = \frac{(m-1)!}{(m-k)!} \tag{2}$$

Example. If m and k are set to 4:

$$\frac{(4)!}{(4-4)!} = \frac{4!}{1} = 4! = 24$$

Additionally, we managed to reduce the solution space even more by fixing the first element in every solution. This is because each circuit may be derived by two different encodings (adding some inverter gates) complementary to each other. Thus, by taking into account such statement, we modified $\#\{\mathcal{S}\}$ into $\#\{\mathcal{S}\}'$ (Formula 2). It is extremely beneficial for the size of the solution space. By analysing this problem, the ensemble reduces from 24 to 6.

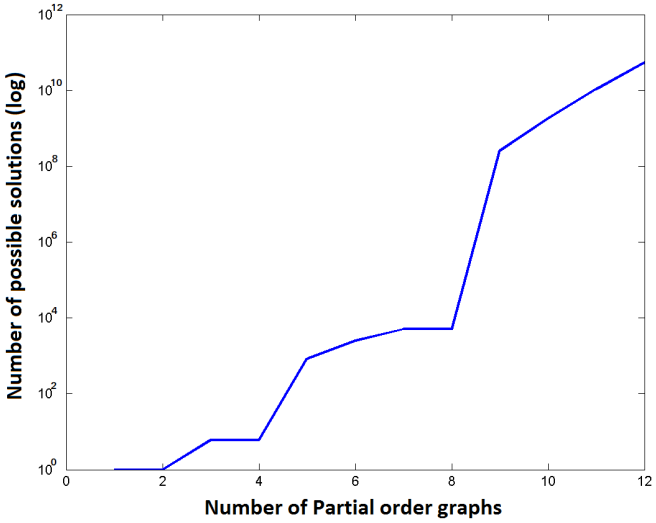

Fig. 3: Number of solutions depending on number of POs.

Figure 3 shows the number of possible solutions (the vertical axis) for encoding a given number of POs (the horizontal axis). The plot is obtained from Formula 2: varying the number of graphs, k, and fixing the size of the opcode ensemble, m, assuming opcodes on minimum number of bits. On y-axis

the $\#\{\mathcal{S}\}'$ is depicted in logarithmic scale. The number of solutions grows up exponentially.

A. Heuristic function

After discussing the number of solutions one should deal with, it is worthwhile mentioning that in the literature of Conditional Partial Order Graphs, a quick way to heuristically evaluate the area of the final controller by looking at the encoding only missed. As a direct consequence, in order to evaluate a solution in terms of whichever constraint (as the area consumption), logic synthesis and technology mapping steps were needed; making the whole process of seeking the best possible solution between the ones available an extremely difficult task. In this research, and in particular in this Section, we therefore aim at presenting a heuristic function for an early evaluation of an encoding, targeting an optimised controller.

In order to describe the heuristic function just mentioned, let us first describe algebraically the differences between the partial order graphs which compose a single model. First parameter for the evaluation is the similarity between couples of graphs, in terms of nodes and arcs. One can set the results, for each couple of partial orders, inside a matrix (named *Difference Matrix*). In particular, it is defined as a strict upper triangular matrix $[\mathcal{N} \times \mathcal{N}]$, where $\mathcal{N}$ is the number of graphs in a model, with all the entries on the main diagonal fixed to 0. Every row (i) is associated to a PO, as well as every column (j). $\mathcal{DM}_{i,j}$ represents the number of vertices and arcs that are present only in one of the two POs i and j (this is similar to the Hamming distance, but on partial orders instead of bit vectors). An example is depicted below, on the left-side of Formula 3. In a similar way it is possible to store all the differences between the opcodes assigned to couples of graphs i and j in the $\mathcal{H}$amming $\mathcal{D}$istance matrix [9] (right-side of Formula 3).

$$\mathcal{DM} = \begin{pmatrix} 0 & 2 & 1 & 4 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad \mathcal{HD} = \begin{pmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{3}$$

A key point to develop the cost function was to associate more similar partial orders to more similar opcodes. Exploiting this knowledge, the cost function we used to evaluate area before the synthesis phase is depicted in Formula 4.

$$\mathcal{F} = \sum_{i,j \in \mathcal{N}}^{i \neq j} (\mathcal{DM}_{ij} - \mathcal{HD}_{ij})^2 \tag{4}$$

Intuitively, minimising $\mathcal{F}$ means encoding similar instruction classes with couples of opcodes with lower $\mathcal{H}$amming $\mathcal{D}$istance.

In order to demonstrate the applicability of such heuristic, we tried to synthesise all the possible solutions of a representation composed by eight graphs (presented in [10], Figure 2.7) and to map the controllers generated with a 90nm simple library composed by basic 1/2 input logic gates (AND, OR, INV, NOR, NAND). The results, taking into account *area* and the *heuristic function*, are plotted in Figure 4 where it is clearly present an average correlation between the area of the final circuit and the cost stem by the encoding.

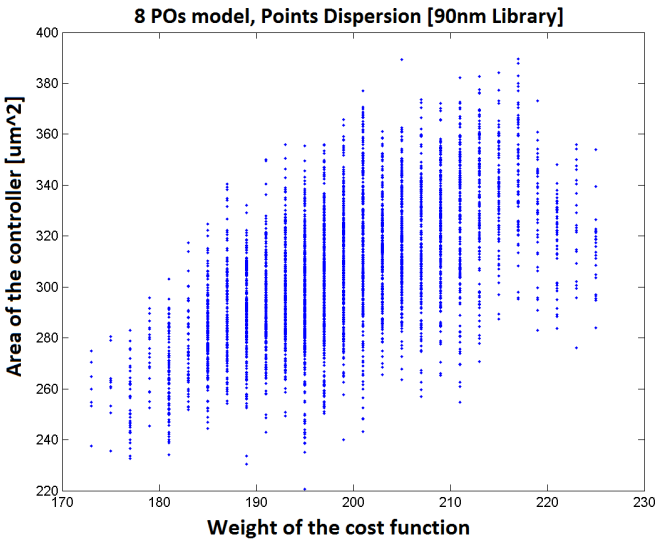

Fig. 4: *Points dispersion* of CPOG composed by 8 graphs.

B. Encoding generation

Exploiting the heuristic function presented previously, designers may achieve quite good results in terms of area (as well as for other constraints) of the final controller without going through the synthesis phase of the model. The last part to complement the encoding association process is to present the encoding generation algorithms that have been developed.

Designers have mostly two choices: trying to generate and synthesise all the possible solutions, or generating few solutions targeting a low cost of function results. It is worth mentioning that, even though the former approach would guarantee a more optimised controller, it is not always possible to apply it to all the models as the number of solutions might exponentially explode for bigger representations. During this work, a tool has been developed (presented in Section V) and it features all the approaches described in this research and in the previous ones. Therefore, both the possibilities are available for the designer who might freely choose between them.

Below we discuss how Simulated Annealing technique can be used to accelerate the search of the optimal encoding. The interested reader is referred to [10] for further information about the so called *Exhaustive encoding*. Briefly, the heuristic algorithm generates as many solutions as the designer wants, trying to associate dynamically couples of opcodes presenting a lower *Hamming distance* to more similar graphs. As some steps of the algorithm are decided randomly, for example when pairs of graphs or opcodes contain the same number of differences, the result is not guaranteed to be optimal, so an additional step is performed in order to minimise the heuristic

TABLE I: Comparison between encodings generations applied to a *CPOG* with eight Partial orders.

Generation approach	**#Encodings inspected**	**Best area** $[\mu m^2]$	**Area overhead**	**Runtime** $[s]$	**Speed-up**
Exhaustive encoding	5040 (all)	220	-	2105	-
Random encoding	10	255	15.73%	2	$\times$1052
	100	247	12.39%	17	$\times$124
	1000	230	4.46%	218	$\times$10
Heuristic encoding (SA)	1	254	15.47%	1	$\times$2105
	10	237	7.65%	2	$\times$1052
	100	225	2.30%	16	$\times$131

function associated to the solution found. Such a step is based on *Simulated Annealing optimisation technique*.

Hereinafter referred to as *SA*, it was introduced by *Kirkpatrick et al.* in 1983 [12]. The aim of this method is to find minimum or maximum values of a function which depends on many independent variables. Commonly, it is named *cost function*, and in our case is the one in Formula 4. Much research was conducted exploiting such an optimisation method, either in computer engineering field or in other subjects [13]. In our case we have used this approach by swapping the opcodes associated to various partial orders attempting to minimise the cost function. We found that the following simulated annealing parameters work well experimentally: *initial temperature* set to 10, *cooldown factor* fix to 0.996 and the *ending temperature* set to 0.1.

The results, applied to the same model mentioned before composed by eight partial orders, are depicted in Table I. In the last row, the outcome of the generation algorithm described so far in this Section is shown. It leads to a quite good result in terms of area-runtime. As one might observe by the Table, the higher the number of solutions inspected, the better the result, no matter which approach is used. Nonetheless, *heuristic encoding* provides better results with fewer solutions inspected with respect to *random encoding* and that is the main reason of the extremely good speed-up it is able to provide with respect to *Exhaustive encoding* runtime.

The results of this Section will be applied to a real Instruction Set Architecture of a microprocessor in the next Section, in such a way to have an extremely sound case of study for demonstrating the applicability of this technique. In Section V instead, this approach will then be applied to a set of different models in order to have a proof of the area overhead which this technique lead to, compared with optimal solutions obtained using the *Exhaustive encoding* method, that is optimal, but slow and computationally expensive.

IV. Case of study: Instruction Set Architecture of ARM Cortex M0+

In order to further explore the CPOG formalism and encoding process, a case study was completed with the ARM Cortex M0+ processor's instructions as a study basis. Methodology for this work was introduced in [14] and was also followed throughout this research. The first step required a derivation of partial orders and it was mainly done by analysing the instruction set description. Although, before partial orders can be constructed, datapath components need to be specified. In [15] five datapath units were obtained, four were reused in this study, Table II describes each of them.

TABLE II: Datapath units that model instructions of ARM Cortex M0+ processor.

Component	**Description**
ALU	Executes mathematical operations.
MAU	Access internal and external memory.
PCIU	Increments Program Counter (PC).
IFU	Provides opcodes to Instruction Register (IR).

Derivation of partial orders involved deep analysis of individual instruction, which was done using the `ARMv6-M` technical manual [16]. This manual provides detailed information about every instructions including encoding, operation pseudo-code and functions of instructions. This process was simplified due to the ability to cluster instructions and give class a single Partial Order (PO). During the derivation process it was noticed that in particular two aspects of instructions affect its graphical representation: *functional similarity* and *addressing mode*. Since, `ARM Cortex M0+` instructions can have two types of addressing modes to specify operands: *register and immediate addressing*. Obtained graphs shared this property in their representation. In particular, addressing mode affected operand fetching process part, instructions with *register type* operand were able to fetch next instruction concurrently with current execution, while *immediate addressing* type instructions required to fetch constant in order to complete instruction.

Functionality difference was another aspect, which differentiated graphs. It is perhaps more intuitive than addressing mode differentiation, as one may understand various purpose instructions are executed in different manner, thus resulting in unique graph. However, instructions can be still categorised under their functionality. General groups are: arithmetic-operations, memory access, control and miscellaneous.

Specification of instruction set procedure extracted 11 different partial orders (which are depicted in Figure 5), which covered in total 61 `ARM Cortex M0+` instructions out of a possible 68. The majority of covered instructions were arithmetical and logical. As just mentioned, not all processor instructions were covered. In particular, hint type instructions,

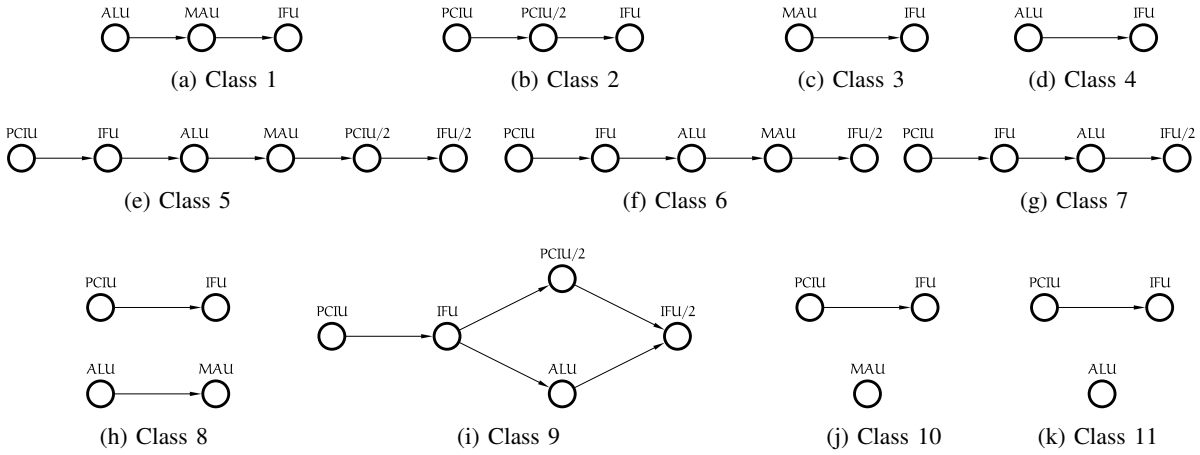

Fig. 5: Conditional Partial Order Graphs model of the ARMv6-M instruction set architecture.

which are associated with interrupts and microprocessor state mode. These instruction required more insightful analysis and due to project time constraints and very high level description of these instruction in [16] they were left out. It is reasonable to predict the effect these instructions could have had on size and complexity of CPOG and control circuit are insignificant. Derived instructions classes are described below.

Class 1 This class models loading data from memory location to Program Counter register instruction. Since this instruction contains the register type operand, constant fetching is not required, which is typically represented by PCIU → IFU. Example `ARM Cortex M0+` instruction: LDR (reg.)

Class 2 NOP instruction, which has no special functionality but proceeding to another instruction, is modelled by this graph. So, after another Program Counter increment next instruction can be fetched by IFU. Example: NOP

Class 3 This class is a representation of Stack instruction: POP. It loads multiple memory locations into registers - represented by MAU and then fetches next instruction IFU. Example: POP

Class 4 This class covers an exotic type of Branch instructions, so called Branch with Link and Exchange, and Branch and Exchange. BLX and BX instructions are a register addressing type, so no constant fetching is needed. We assume that the ALU can perform all functions required to execute these instructions i.e. saving Program Counter address to Link Register. Examples: BLX(reg.), BX

Class 5 Instructions covered by this graph are memory type load/store instructions with immediate addressing mode. Therefore, the constant has to be fetched, which is represented by PCIU → IFU part. After offset is calculated and actual function of instruction executed, next instruction can be fetched PCIU/2 → IFU/2. Examples: LDR (literal), STR(imm.)

Class 6 This class contains a single instruction, which is memory type with immediate operand. The functionality of this instruction is loading memory location to the Program Counter register. Likewise, other immediate operand classes, a constant has to be fetched (PCIU → IFU). Then an instruction could be executed by ALU → MAU and the next instruction fetch IFU/2. Example: LDR (imm.)

Class 7 Unconditional branch instruction is represented by this graph, where address offset to branch is specified as a constant and thus needs to be fetched (PCIU → IFU). Then, ALU updates Program Counter with new value, which is calculated with provided offset, lastly next instruction can be fetched IFU/2. Example: B

Class 8 This graph covers the same instructions as Class 5 just with register addressing mode. Since this addressing mode, instructions do not require operand fetching: PCIU → IFU (next instruction fetch) and ALU → MAU (instruction execution) can be completed in parallel. Examples: LDR (reg.), STR(reg.)

Class 9 This class shares several instructions with Class 11, with a key difference in addressing mode only, which is an immediate addressing mode. Like other instructions with constants, it requires constant fetching to Instruction Register, which is modelled by PCIU → IFU, then ALU can be executed concurrently with another Program Counter increment; lastly the next instruction is fetched. Examples: ADD(imm.),LSR (imm.)

Class 10 Instructions modelled by this class are memory type instructions, although this time multiple registers/address locations are transferred. Stack instruction: PUSH, is an example of this class. Examples: LDM, STM

Class 11 This class covered largest subset of instructions with functionalities including: arithmetical, logical and data copy. Every instruction within this class is a register addressing type. Therefore, likewise this type of class some concurrency exists in this graph, the next instruction can be fetched PCIU → IFU in parallel with latter execution ALU. Examples: CMP (reg.), MOV(reg.)

A. Compositional results

After derivation of partial orders was completed, research proceeded with the next step: encoding and generation of CPOGs. As the solutions space for such a Conditional Partial Order Graph composed by 11 different behaviours is extremely wide, according to Formula 2, we could not apply the approach named *Exhaustive encoding* in order to seek the most optimised controller, targeting in our case area reduction. Therefore, in order to carry out a fair comparison, we focused on comparing the newly introduced *Heuristic encoding (SA)* with several solutions obtained by adopting *Random encoding* technique. The latter one does nothing else but associating random opcodes to the partial order graphs composing the whole representation. Those two algorithms indeed fit well to a CPOG with such a high number of graphs as the one represented in Section IV. Additionally, together they provide a good case of study to give a rough idea to readers about the potential of the heuristic approach, compared with the random one. *Runtime* of the tool and *area* of controller are used as comparison parameters.

The best results, exploiting the algorithm just mentioned, are depicted in Table III, obtained with three different approaches applied to the partial orders representing the `ARMV6-M` Instruction Set Architecture.

TABLE III: Controller size comparison for ARMv6-M ISA model (enc stands for encodings).

Generation approach	**Area** $[\mu m^2]$	**Runtime** $[s]$	**#enc.**
Random encoding	226	18243	100000
SAT-based encoding	204	2	-
Heuristic encoding (SA)	191	169	1000

The worst solution is represented by the controller that comes up with the *Random encoding* approach, it generates the biggest controller. It is pareto-dominated by all the other techniques, but it is useful for giving a general idea about the whole area of the controller without using any optimisation technique. Even though the area of the controller might not look much bigger than the other two shown in the Table, it is worth observing that one hundred thousands of encodings have been tried to reach out such result. It has been required much more time with respect to the other approaches as observable by the *runtime* column.

The *SAT-based encoding* approach, described in [4], gives a very good result compared to the *Heuristic encoding (SA)* technique, which in this case gives the best result by analysing one thousand encodings only. To understanding the benefit of this approach, the reader should take into account that, according to Figure 3, the size of the solution space for a representation composed by 11 POs contains approximately 10^{10} possible encodings. *Single-literal encoding* has not been taken into account for the comparison because of the higher number of variables it requires to synthesise the controller.

Now, let us compare the heuristic technique, which is the one we aim to present in this paper, with the random approach in order to perform a fair comparison. As we mentioned before, we tried to generate an extremely high number of solutions with Random encoding, and a small one with Heuristic encoding (SA). The result is surprisingly good, and it is shown in Figure 7. Though the number of solutions generated by the random encoding technique is much higher than the encodings tried referring to the new approach, they are mostly concentrated between 110 and 220 on the x-axis. The heuristic solutions instead concentrate between 80 and 110 ([10], Figure 6.7) and that is why the probability of finding a better (in terms of area in this case) derived controller is much higher. The results in Table III confirm our hypothesis. The x-axis of the graph is the cost function presented in SectionIII-A, while on the y-axis the bare area $[\mu m^2]$. As we demonstrated, the lower the cost function of the solution, the higher the probability to select a better encoding in terms of area, therefore the main goal is to have a heuristic which returns encodings with a low cost of the function, in Figure 6 the compositional graphs of the `ARMv6-M` model are depicted for sake of completeness.

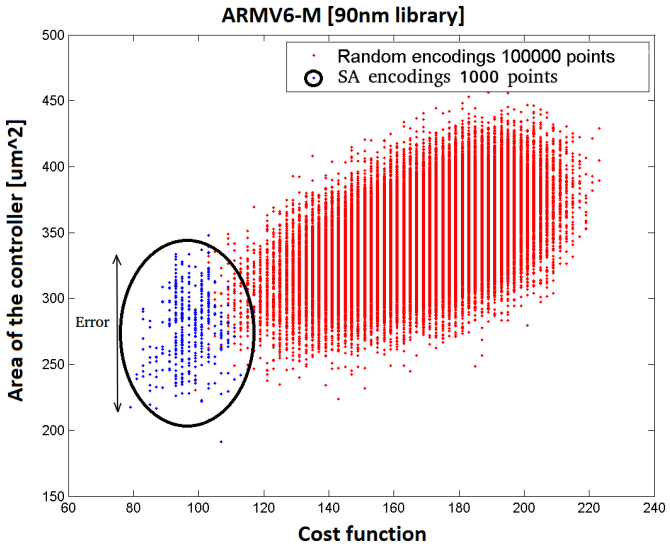

Fig. 7: Comparison between Heuristic encoding (SA) and Random encoding techniques applied to ARMV6-M ISA model.

The reason for such a bad result for random encoding resides on the massive number of possibilities the solution space is composed by. Indeed, even though 100000 may seem a high number of solutions, it represents the 0.0009 % of solution space only, so it is extremely unlikely to find a really good point.

The only drawback depending on the heuristic cost function is related to the non total linearity in the relationship between the heuristic function and the parameter to optimise (area in this case). The error that intrinsically affects the solution space (Figure 7) constrains the tool to analyse the whole range of solutions, as the best controller in terms of area might be present with a slightly higher cost of $\mathcal{F}$ (Formula 4).

In light of the above, one of the main goals of this research has been addressed. The composition of partial order graphs

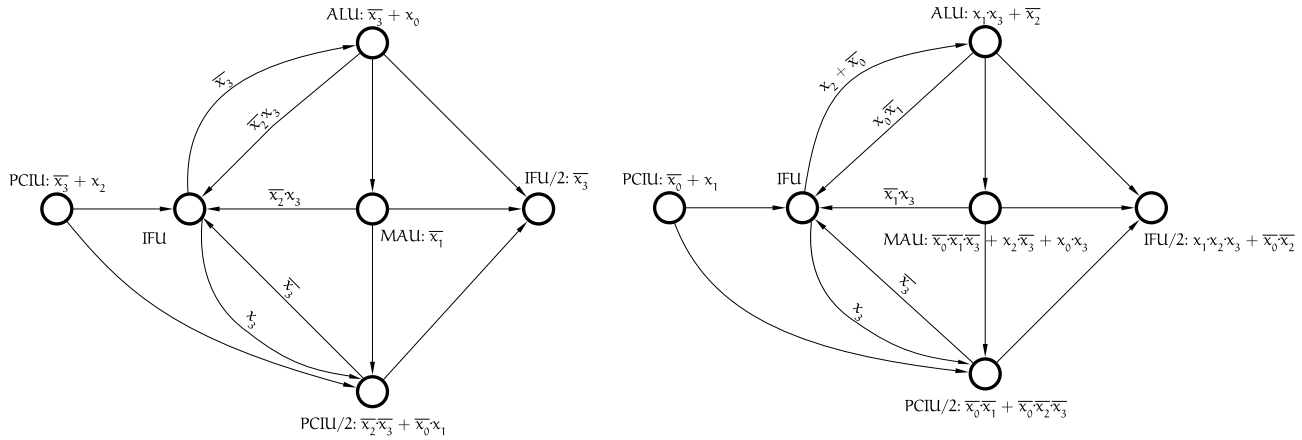

Fig. 6: CPOGs generated with Heuristic encoding (SA) (left side) and Random encoding (right side) algorithms.

has been automated exploiting a heuristic-based algorithm which is able to synthesise the final controller relying on an early-evaluation of the opcodes, prior to the synthesis phase. In the next Section another problem related to instruction set architecture modelling will be addressed: that is the *encoding customisation*. It is needed when one has to deal with particular applications where some bits have to be fixed for reserved functions, or when the opcodes have to be composed by a variable number of bits.

V. Tool implementation

`SCENCO` is the name of the tool which features all the approaches we described in this work, it embeds either the previous techniques (with the advantages and drawbacks highlighted in Section I) and the new algorithm that we aim at presenting hereby. SCENCO, which stands for *SCEN*arios *ENCO*der, has been integrated as an open source tool in `Workcraft` [17], in particular inside the Conditional Partial Order Graphs plugin. It provides a friendly graphical user interface which complements the representative power of the model. All the generation algorithms described in this work can be used.

In order for the tool to provide different optimisation features, such as the ones which might be related to area, power or latency optimisation of the derived controller, `ABC` framework [18] has been integrated into the standard flow of the software. It allows optimisation of the final equations which come up with the synthesis of the representation, constrained by different parameters. It might be extremely useful for having concrete outcome under different constraints, in fact a real gate library (in *genlib* format [19]) can be manually set in order to map the Boolean equations onto gates with real characteristics. Furthermore `Espresso` logic minimiser [25] has been integrated into `SCENCO` in order to solve the Karnaugh Maps for deriving the Boolean equations associated to each vertex/arc. Finally `Clasp`[23]/`Minisat`[24] are used to provide the SAT-based optimal encoding option.

The approaches which may be exploited by using this tool are mainly three. *Heuristic encoding* is the approach we aim to present in this work, it is extremely fast and totally customisable: the designer is allowed to tune the depth of the search (it may be useful to trade off the soundness of the solution with the time for obtaining good encoding, as demonstrated in Table I) as well as totally customise the opcodes. Indeed, one can set the length of the opcodes to use as well as some particular bits in the way described following:

- if an `X` is inserted, the tool will have the total control on that particular bit, therefore it will try to optimise it as much as possible (i.e. an opcode with length 2 set as `XX` may become $\{00, 01, 10, 11\}$);
- if a `0` or a `1` is inserted in the opcode associated with a partial order graph, the tool will look for the optimal encoding without modifying those bits set by the user (i.e. an opcode with length 2 set as `X1` may become $\{01, 11\}$, the second bit will be stuck at `1` in both cases);
- if a `-` is inserted the tool will deal with that particular bit as a *Don't care* condition. Hence, the final opcode containing a `-` will be on a shorter number of variables with respect with the ones without any Don't care (i.e. an opcode with length 3 set as `XX-` might become $\{00, 01, 10, 11\}$, that is because: in the case 01 is chosen both the two opcodes $\{010, 011\}$ will be theoretically assigned to the partial order graph).

The possibility to completely customise the opcodes addresses the need of flexibility which particular applications, as instruction set architectures, need.

Single literal encoding [6] is the fastest, but can use a lot of variables for encoding and does not support customisable opcodes, which might be a problem if one has to deal with instruction set architecture modelling; otherwise it provides a good solution in terms of the derived controller.

SAT-based optimal encoding is the technique described in [4] which, as highlighted in Section I, has many limitations; but might be useful when one does not need to customise the encodings, deal with dynamic CPOGs, or with representations which contain more than 12 - 15 instruction classes.

A. Benchmarks

In this Section, we aim at demonstrating the benefits of the approach we presented in Section III. To do so, we are going to apply *heuristic encoding* by fixing the number of solutions to find to 100, and by constraining the opcodes to minimum number of variables necessary to encode all the POs. This latter imposition is helpful when the number of external I/O pins need to be reduced. Other methods will also be applied to multiple instances of Conditional Partial Order Graph representations composed by different behaviours each. Afterwards, we are going to carry out a comparison, in terms of area overhead and runtime of the tool, with the best possible solution (if available) present in the whole encoding set for each representation. The other approaches will also be considered in the benchmark for the sake of completeness, even though some of them are able to find solutions by using a higher number of variables.

The other two instances we applied these techniques to are subsets of instruction set architectures belonging to `Texas Instruments MSP430` processor which has been modelled in [20], and the `Intel 8051` [14]. The former one contains dynamical partial order graphs, that are particular conditions which, applied to some particular vertices or arcs, modify dynamically the behaviour of the system, while the latter one, as for the model of the `ARMv6-M` does not contain such typology of partial orders.

Table IV depicts all the results in terms of area of the derived controller and runtime of the tool, for seeking an optimal encoding for the models. Many considerations might be done on the data collected in the Table. First of all let us analyse why the results are not present in each box of the Table ("-" character). Regarding Exhaustive encoding, it was not possible to apply this approach to Conditional Partial Order Graph models composed by more than 8 partial orders because of the high number of encodings the whole solution sets are composed of. According to Formula 2 and Figure 3, when the #POs increases beside 8, the size of the solution set tends to be too wide to be inspected entirely. Concerning the SAT-based encoding method instead: when applied to the `Texas Instruments MSP430` instances, it does not produce any result due to the limit of handling branching instructions, as mentioned in Section I.

Another characteristic that is worth analysing is the number of variables which the various approaches use to seek an optimal solution. While the *heuristic* approach does not increment the length of the opcodes in order to reduce the complexity of the derived controller, the Single-literal encoding approach and SAT-based encoding do. The former increments the number of variables in order to target one single Boolean literal per node and arc, while the latter one (regarding the `Intel 8051` subset expressed with 9 POs) cannot solve the problem with 3 variables only, probably due to the high complexity of the graphs. Therefore it needs 4 variables to generate a feasible solution. These approaches are fast and good in terms of the controller optimisation, nonetheless sometimes they might not be exploited either because they do not allow customisation of the opcodes, because of the increased number of variables they may require for the encoding process, or finally due to the limitation in dealing with dynamic behaviours.

On the other hand, heuristic encoding search, improved via simulated annealing optimisation algorithm, provides extremely good results with a slight increase of the search time. In this case we chose to keep the length of the opcode at the minimum, but it may be customised addressing the need of instruction set architectures modelling. In the Table V the *area overheads* and the *speed-up* with respect to the Exhaustive encoding are depicted.

TABLE V: Area overhead and speed-up of the heuristic encoding approach with respect to optimal one.

Model	#POs	Area overhead	Speed-up
ARM Cortex M0+	11	-	-
	8	7.96%	$\times$92
Texas Instruments MSP430	8	11.24%	$\times$124
	7	8.42%	$\times$110
Intel 8051	9	-	-
	8	3.52%	$\times$138
	7	7.69%	$\times$139
Average	-	7.77%	$\times$120

As one might observe, the average overhead in terms of area is 7.77%, which is an extremely good result considering the high average speed-up of the search of $\times$120, which the tool provides considering all the cases taken into account in this research.

VI. Conclusions and Future Research Directions

As was already demonstrated, the *Conditional Partial Order Graph* model provides a compact environment to represent a system that includes different behaviours, both for synchronous and self-timed implementations. As proved in Section IV and V-A additionally, CPOG tailors well to instruction set architecture development, it may help designers find an optimal and efficient encoding as well as to automatically instantiate the decoding module. Furthermore, this work addresses the gap in the synthesis part allowing designers to choose the encoding for the representation via different generation algorithms targeting a better controller (which can be further optimised under different perspectives via `Abc` framework).

In order to bridge this gap, `SCENCO` has been developed and integrated into `Workcraft` [17]. As described in [10], an interesting option provided by this tool is the possibility to select custom encoding either by setting the number of bits the designer wants the opcodes to be composed of, and by fixing some bits inside. Both of these options might be worth applying to particular applications where some particular bits should be fixed for particular functions. This research may be useful to those companies which already design ASIP (Application Specific Instruction-set Processor) such as `Altera` with *Nios*, `Xilinx` with *Microblaze* or `Cadence` (`Tensilica` section) with *XTensa*.

TABLE IV: Benchmark of the encoding approaches applied to seven instructions set architecture subsets. Text in bold represents the smallest controller for each model analysed. Unit of measure: Area: [μm^2] (opcode length in brackets), Runtime [s].

Model	**#POs**	**Exhaustive encoding**		**Single-literal encoding**		**Heuristic encoding (SA)**		**SAT-based encoding**	
		Best area	Runtime	Best area	Runtime	Best area	Runtime	Best area	Runtime
ARM Cortex M0+	11	-	-	**167 (5)**	1	191 (4)	16	204 (4)	2
	8	133 (3)	1292	**126 (5)**	1	143 (3)	14	166 (3)	2
Texas Instruments MSP430	8	**219 (4)**	1982	255 (9)	1	244 (4)	16	-	-
	7	**171 (4)**	1754	176 (9)	1	185 (4)	16	-	-
Intel 8051	9	-	-	**184 (8)**	1	185 (3)	16	195 (4)	3
	8	**160 (3)**	2075	161 (7)	1	165 (3)	15	195 (3)	3
	7	**130 (3)**	2092	140 (6)	1	140 (3)	15	143 (3)	1

There are many *research directions* that our work might lead to. One of the most worthwhile to inspect is surely the enhancement of the heuristic function in order to improve the search of an optimal encoding, maybe making the relationship between the cost and the constrained parameter (area, power, latency, etc.) more linear. In fact, as one might observe in Figure 7, the cost function so far is affected by a certain degree of error (given by the solutions with the same value of the heuristic function, but different area), which may also affect the final result negatively. Secondly, the research on a new generation algorithm could be conducted, as might be the ones inspected in [7][21], which we think may be adapted to our purposes. The work of *Goldberg et al.* (Section 3) additionally, presents an interesting characterisation of the symmetrical solutions, that is surely worthwhile to explore in order to speed-up the search process. Finally, as CPOG tailors well to asynchronous controller design [22], a further research direction might be to integrate analogue specifications into the CPOG representation in order to represent asynchronous circuits able to satisfy analogue requirements.

ACKNOWLEDGMENT

We would like to thank all the μSystems Research Group members at Newcastle University for supporting us and this research. In particular a special acknowledgement goes to: Alex Yakovlev, Danil Sokolov, Maxim Rykunov and Jonathan Beaumont. This research was supported by Royal Society Grant Computation Alive: Design of a Processor with Survival Instincts and EPSRC Grant UNCOVER.

REFERENCES

[1] H. Iwai, *"Roadmap for 22nm and beyond (Invited Paper)"*, Microelectronic Engineering, vol. 86, pp. 1520-1528, 7/2009.

[2] J. Rabaey, *"Low Power Design Essentials: New York, London"*; Springer 2009.

[3] A. Mokhov, A. Yakovlev. *"Conditional partial order graphs: Model, synthesis, and application"*. IEEE Transactions on Computers, Volume 59, Pages 1480-1493, November 2010.

[4] A Mokhov, A Alekseyev, A Yakovlev. *"Encoding of processor instruction sets with explicit concurrency control"*. Computers & Digital Techniques, IET. Volume 5, Pages 427-439. November 2011.

[5] A. Mokhov, M. Rykunov, D. Sokolov and A. Yakovlev. *"Design of Processors with Reconfigurable Microarchitecture"*. J. Low Power Electron. Appl. 2014, 4, Pages 26-43. 20 January 2014.

[6] A. Mokhov. *"Conditional Partial Order Graphs"*. Ph.D. Thesis, Newcastle University, September 2009.

[7] Evguenii I. Goldberg, T. Villa, R. K. Brayton and Alberto L. Sangiovanni-V Incentelli. *"A Fast and Robust Algorithm for Face Embedding"*. Computer-Aided Design, 1997. Digest of Technical Papers., 1997 IEEE/ACM International Conference on 9-13 Nov. 1997, San Jose, CA, USA, Pages 296 - 303.

[8] A. Mokhov, A. Yakovlev. *"Conditional partial order graphs and dynamically reconfigurable control synthesis"*. Design, Automation and Test in Europe, 2008. DATE'08, Pages 1142-1147, 10/03/2008.

[9] R. W. Hamming. *"Error Detecting and Error Correcting Codes"*. The Bell System Technical Journal Vol. XXIX, No. 2, April, 1950.

[10] A. de Gennaro, *"Design of Reconfigurable Dataflow Processors"*. Master's Degree in Computer Engineering (Facolta' di Ingegneria, Politecnico Di Torino), October 2014.

[11] E. L. Lawlor, *"Combinatorial Optimization"*, (Holt, Rinehart & Winston, New York, 1976).

[12] S. Kirkpatrick, C. D. Gelatt Jr and M. P. Vecchi, *"Optimization by Simulated Annealing"*, vol. 220, number 4598, 13 May 1983.

[13] F. Javier Rodriguez-Diaz, Carlos Garcia-Martinez and Manuel Lozano, *"A GA-Based Multiple Simulated Annealing"*, IEEE, Evolutionary Computation (CEC), July 2010.

[14] M. Rykunov. *"Design of Asynchronous Microprocessor for Power Proportionality"*. Ph.D. Thesis, Newcastle University, December 2013.

[15] P. Stankaitis, *"Algebraic Specifications of ARM Cortex M0+ Instruction Set"*, Bachelor Degree thesis, 2014.

[16] ARM Ltd. *"ARMv6-M Architecture Reference Manual"*. ARM DDI 0419C (ID092410), 2010.

[17] I. Poliakov, D. Sokolov, A. Mokhov. *"Workcraft: A static data flow structure editing, visualisation and analysis tool"*. Petri Nets and Other Models of Concurrency - ICATPN 2007. Pages 505-514, 2007.

[18] Berkeley Logic Synthesis and Verification Group. *"ABC, a System for Sequential Synthesis and Verification"*. www.eecs.berkeley.edu/~alanmi/abc/

[19] Richard Rudell. *"Genlib: Combinational Gate Specification"*. Berkeley University, http://www.eecs.berkeley.edu/~alanmi/publications/other/SIS_paper_genlib.pdf.

[20] A. Mokhov, V. Khomenko. *"Algebra of Parameterised Graphs"*. ACM Transactions on Embedded Computing Systems, Volume 13, Issue 4s, 2014.

[21] T. Villa. *"Encoding Problems in Logic Synthesis"*. University of California at Berkeley, Ph.D. Thesis in Engineering. 1995

[22] D. Sokolov, A. Mokhov, A. Yakovlev, D. Lloyd. *"Towards asynchronous power management"*. 2014 IEEE Faible Tension Faible Consommation (FTFC), Pages 1-4, May 2014.

[23] Martin Gebser, Benjamin Kaufmann and Torsten Schaub. *"Conflict-Driven Answer Set Solving: From Theory to Practice"*. Potsdam University, Institut fur Informatik, August-Bebel-Str. 89, D-14482 Potsdam, Germany. May 4, 2012.

[24] Niklas En, Niklas Sorennson. *"An Extensible SAT-solver"*. Chalmers University of Technology, Sweden.

[25] McGeer, P., Sanghavi, J., Brayton, R. and Vincentelli, A.S. *"ESPRESSO-SIGNATURE: A New Exact Minimizer for Logic Functions"*. Design Automation, 1993. 30th Conference on 14-18 June 1993. Pages: 618 - 624. DOI: 10.1109/DAC.1993.204022. Publisher: IEEE.