# Modelling and FPGA-based Implementation of Graph Colouring Algorithms

Valery Sklyarov, Iouliia Skliarova, Bruno Pimentel
Department of Electronics, Telecommunications and Informatics, IEETA
University of Aveiro, Aveiro, Portugal
skl@det.ua.pt, iouliia@det.ua.pt, pimentel@ieeta.pt

## Abstract

The paper discusses an effective matrix-based exact algorithm for graph colouring that is well-suited for implementation in FPGA. It is shown that the algorithm can also be used as a base for a number of approximate algorithms. Software models (described in C++) and hardware circuits (synthesised from Handel-C specifications) for the exact algorithm are discussed, analysed and compared. Characteristics and capabilities of the approximate algorithms are also examined. It is shown that such algorithms can be used for a wide range of practical applications, including robotics and embedded systems.

**Keywords**: graph colouring, Handel-C specification, FPGA implementation

## 1 Introduction

Graph colouring algorithms are widely used for solving different engineering problems in robotics and embedded systems. One of the most common examples is register allocation [1], where the compiler assigns intermediate computation values to storage units in an embedded processor. Another example can be found in [2] for memory-constrained networked embedded systems. One of the tasks, considered in [2] was scheduling the activities of so called ping nodes (actuators), which can be formulated as a distributed graph colouring problem. A colour corresponds to a specific time slot in which a ping node vibrates. Two adjacent nodes in the graph, each representing an actuator, cannot have the same colour since the vibrations from these actuators would then interfere with each other. The number of colours is therefore the length (in distinct time slots) of a schedule. The objective is to find a shortest schedule such that the ping nodes do not interfere with one another, in order to minimise damage detection and response times. A key open problem in mobile robotics is to find a way to program a mobile robot to explore an environment and visually determine its position (coordinates) [3]. A promising approach is to have the robot first determine an optimal set of visual landmarks for navigation, and then use the landmarks to find its position. The problem of finding the set of landmarks can be formulated as a graph colouring problem. Another potential scope is microprogramming for application-specific embedded microprocessors (ASEMP). Let $M = \{m_1, m_2, \ldots m_N\}$ be a set of micro-operations for an ASEMP. Consider a graph G, which has N vertices. Two vertices $m_i \in M$ and $m_j \in M$ are connected with an edge if and only if $m_i$ and $m_j$ can be executed in ASEMP at the same time. Colouring the graph G enables us to code the micro-instructions (composed of micro-operations from M) using the minimal number of fields. A similar problem has to be solved for resource distribution in parallel systems.

There are many more general problems. For example, it is known that any geographical map can be painted in four colours in such a way that any two countries having common boundary will be painted in different colours. This follows from the theorem of Haken-Appel [4,5] that any planar graph can be painted in the maximum of four colours. Many other practical applications of graph colouring problem can be found in [6,7].

The considered above different tasks can be formulated mathematically and solved through applying the same model and methods. This shows the importance of the graph colouring problem. It should be noted that the graph colouring is a very computationally complex task (this is NP-hard problem [5]). Often it is necessary to solve this problem in run-time (for example, for register allocation [1]), which increases essentially the total execution time for the relevant applications. However, acceleration can be achieved with the aid of graph-colouring targeted hardware [8,9]. This makes possible the number of the required clock cycles for the respective applications to be decreased significantly.

The paper suggests algorithms for solving the graph colouring problem in FPGA. To compare the results two different models have been explored:

- C++ projects running on general purpose computers;
- Handel-C based projects for FPGA.

It is known [10] that exhaustive search (complete enumeration) for the considered problem requires $N^K$ iterations, where N is the number of vertices in a graph and K is the number of colours. Thus, this might involve significant computational resources. In

order to find out a compromise between the execution time and the required accuracy two types of algorithms have been implemented and analysed. They are an exact algorithm producing the optimal solution and approximate algorithms enabling us to stop the execution as soon as an appropriate for the considered problem solution has been found.

## 2 Matrix Specification

Binary and ternary matrices are very well suited for processing them in FPGAs [11]. This section demonstrates that any graph, that has to be coloured, can be converted to matrix specification in such a way that solving the problem over the matrix is equivalent to solving the problem over the graph. The graph [10] in figure 1 will be used as an example to explain all the steps for the conversion.

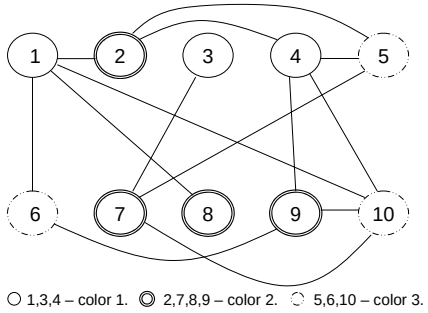


**Figure 1**: Graph G and its optimal colouring.

For solving the graph-colouring problem it is required to paint graph nodes in such a way that: any two nodes connected with an edge are painted in different colours and the number of used colours is minimal.

There are several optimal solutions for the considered graph (with the minimum number of colours, 3) and one of them is shown in figure 1: {{1,3,4},{2,7,8,9},{5,6,10}}.

Let us consider a matrix, which has the same number of rows as the number N of vertices of the graph G. If and only if two vertices $m_i$ and $m_j$ are connected with an edge in G then the matrix rows i and j must be orthogonal. Orthogonality of two ternary vectors is defined as follows [10]:

$(m_i \text{ ort } m_j) \Rightarrow \{m_i\} \cap \{m_j\} = \varnothing$,

where $\{m_i\}$ ($\{m_j\}$) is a set of binary vectors that correspond to the ternary vector $m_i$ ($m_j$) by replacing the don't care values (-) with all possible combinations of 0s and 1s. For example, the following two ternary vectors 0-11- and -1110 are not orthogonal, because $\{00110,00111,01110,01111\} \cap \{01110,11110\} = \{01110\} \neq \varnothing$.

If and only if two ternary vectors are not orthogonal (let us designate this as $m_i$ ~~ort~~ $m_j$) they do intersect in the Boolean space:

$(m_i \text{ ~~ort~~ } m_j) \Leftrightarrow (m_i \text{ ins } m_j)$.

Table 1 depicts matrix μ(G) that was built for the graph G in figure 1. This is a symmetric matrix with respect to the main diagonal that is why just a lower triangle of the matrix is sufficient. The second (upper) part might contain all don't care values because the first (lower) part provides all the necessary and sufficient conditions.

**Table 1:** Matrix μ(G) with rows 1, 2,…,10 and columns I, II,…,VIII.

| | I | II | III | IV | V | VI | VII | VIII |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | - | - | - | - | - | - | - |
| 2 | 1 | 0 | - | - | - | - | - | - |
| 3 | - | - | 0 | - | - | - | - | - |
| 4 | - | 1 | - | 0 | - | - | - | - |
| 5 | - | 1 | - | 1 | 0 | - | - | - |
| 6 | 1 | - | - | - | - | 0 | - | - |
| 7 | - | - | 1 | - | 1 | - | 0 | - |
| 8 | 1 | - | - | - | - | - | - | - |
| 9 | - | - | - | 1 | - | 1 | - | 0 |
| 10 | 1 | - | - | 1 | - | - | 1 | 1 |

The matrix was built using the following rules:

Insert the first row corresponding to the vertex 1 (see figure 1), which has the first element zero and all the remaining elements equal to *don't care* (note that at the beginning we do not know the exact number of remaining don't cares, but this does not give rise to any problem);

Provide orthogonality of the 1st vertex with the vertices connected to the 1st vertex by edges. For our example they are 2, 6, 8 and 10 (see figure 1). It is done by recording the value "1" in the first column (I) of the rows 2,6,8 and 10 of table 1. All empty elements of the first column are assigned to *don't care* ("-"). Correctness of these operations is evident from the definition of orthogonality;

If the vertex 2 has connections with succeeding vertices, i.e. with the vertices 3,4,5,… then use column 2 to indicate orthogonality; otherwise, skip vertex 2 and consider the next one. For our example the vertex 2 is connected with the succeeding vertices 4 and 5. Thus the column II contains "1"s in the rows 4 and 5 and "0" in the row 2.

Use the same rules for all the remaining vertices. For our example it permits to fill in table 1.

Obviously, the maximum number of columns for the considered matrix μ(G) cannot exceed the value N-1, where N is the number of vertices in the graph G. The method of conversion considered above makes possible to formulate the graph colouring problem over a matrix μ(G). Then it is necessary to find out the minimal number K of such subsets of rows that any subset does not contain mutually orthogonal rows. In other words, any two of the rows belonging to the same subset must intersect in the Boolean space. The number K will be the minimal number of colours for

graph G and rows from each subset will correspond to the vertices of graph G that can be painted in the same colour.

For example, the following sets give a feasible solution: {{1,3,4}, {2,6,7,8},{5,9},{10}}. However K = 4 > 3 and this is a non-optimal solution. Let as consider any subset, for example the first one {1,3,4}:

0 - - - - - - -
- - 0 - - - - -
- 1 - 0 - - - -

Any two vectors of this subset do intersect and the result of intersection for all three vectors is: 0 1 0 0 - - - - - . It is important to note that the considered above set cannot be expanded. If we add any of the remaining rows from table 1 the requirements will be violated. In other words, the vector 0 1 0 0 - - - - - is orthogonal to all the remaining vectors (i.e. to rows 2,5,6,7,8,9,10) of table 1.

## 3 The Exact Algorithm

We have already mentioned that exhaustive search algorithms for the considered problem are very time consuming. They are not efficient and sometimes even cannot be used at all due to time limits. The considered exact backtracking search algorithm is based on the following four steps that are common for combinatorial search methods [10] and that are repeated sequentially until the solution is found.

1. *Reduction*. The matrix is reduced as much as possible applying some pre-established rules.

2. *Splitting*. The problem is decomposed in less complicated sub-problems.

3. *Termination*. The current step is terminated as soon as a new incomplete solution is of the same quality (i.e. contains the same number of colours) as any previous solution that has already been found.

4. *Search for the optimal result*. The steps 1-3 have to be repeated until all possible solutions have been implicitly examined.

Some of the reduction and splitting rules can be directly borrowed from the method of condensation, proposed in [10].

The algorithm will be demonstrated on the example of graph G in figure 1 and the respective matrix μ(G) in table 1. The following reduction rules will be used:

1. If, after selecting a new colour, the matrix μ(G) contains a column *i* without values 0 (1) then this column *i* can be deleted;

2. If, at any intermediate step of the algorithm, the matrix μ(G) contains a row *j* with just don't cares ("-") then this row *j* can be deleted from the matrix and included in the constructed subset;

3. All the rows that have already been included in the constructed subsets are removed from the matrix μ(G).

Figure 2 depicts a part of the search tree, which will be built sequentially in accordance with the following steps:

1. Choose a new colour (i.e. create a new initially empty subset);

2. Apply the reduction rules;

3. Consider the topmost row $m_i$ in the matrix;

4. Include the row $m_i$ in the constructed subset and delete it from the matrix;

5. Find out all other rows intersecting (i.e. not orthogonal) with the vector $m_i$;

6. Select the first not tried yet row $m_j$ from point 5, include it in the constructed subset and then delete it from the matrix;

7. Assign $m_i = m_i \; ins \; m_j$ and repeat the steps 5-7 if this is possible. If this is not possible go to the step 8;

8. If the intermediate matrix is not empty repeat the steps 1-7. Otherwise, store the solution found and then backtrack to the nearest branching point (set at the step 6 and try to find a better solution by repeating the steps 6-8.

Let us consider how to discover an exact solution for the graph depicted in figure 1 and the relevant matrix μ(G) shown in table 1. The subsequent list contains all the required operations. The first level of headings, such as 1, 2,… indicate the numbers of subsets of vertices (rows) that are being constructed. The second level of headings, such as 1.1, 1.2,…,2.1… indicate the sequence of operations needed to construct any subset of vertices (rows) that have to be painted in the same colour.

The list is given below.

1. Executing operations for discovering the first subset of vertices (rows of table 1) that have to be painted in the first colour.

1.1. The reduction rules cannot be applied.

1.2. Selecting the first row (1) of table 1. This row forms the root of the search tree or the level 0 (see figure 2).

1.3. Removing the row 1 from the matrix μ(G) and including this row in the first subset $S_1 = \{1\}$.

1.4. Building the first group of nodes of the tree that correspond to such rows of table 1 that are compatible with the row 1. The row $m_j$ is compatible with the row 1 if and only if the condition $m_1 \; ins \; m_j$ is satisfied. Since all the rows 3, 4, 5, 7, 9 are compatible with the row 1 they form the first level of the tree in figure 2.

1.5. Selecting the first vertex (row) from the first level. Let us agree to order vertices of the same level from the left (the smallest number) to the right (the greatest number). Thus, the row 3 is selected, included in $S_1$ ($S_1$ = {1,3}) and then deleted from the matrix μ(G).

1.6. Building the second group of nodes of the tree that correspond to such rows of table 1 that are compatible with the vector (*1 ins 3 = 0------- ins --0------ = 0-0-----*). Since the rows 4, 5, 9 are compatible with the vector (*1 ins 3*) they form the second level of the tree in figure 2 and $S_1$ = {1,3,4}. The row 4 is deleted from the matrix.

1.7. Selecting the first vertex (row) from the second level and repeating the same operations (forming the $3^d$, the $4^{th}$, etc. levels) while this is possible. For our example, the vector (*1 ins 3 ins 4*) is not compatible with the remaining rows of table 1, thus, the first subset is $S_1$ = {1,3,4}. Note that later this subset might be changed.

1.8. The resulting matrix $\mu(G)\backslash S_1$ (where all the rows included in $S_1$ have been deleted) is shown in table 2.

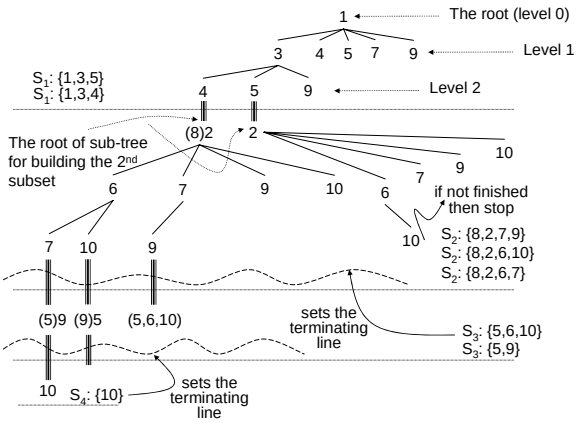


**Figure 2**: Execution steps of the exact algorithm.

2. Executing operations for discovering the second subset of vertices (rows of table 1) that have to be painted in the second colour.

2.1. Applying the reduction rules. The rule 1 can be applied to columns I, III, IV and then the rule 2 can be applied to row 8. The resulting matrix is shown in table 3.

2.2. Executing operations that correspond to points 1.2-1.8 above. As a result, the subset $S_2$ will be built ($S_2$ = {8,2,6,7} – see figure 2). The resulting matrix $\mu(G)\backslash(S_1 \cup S_2)$ is shown in table 4.

3. Executing operations for discovering the third subset of vertices (rows of table 1) that have to be painted in the third colour. Finally, $S_3$ = {5,9} – see figure 2.

4. Executing operations for discovering the fourth subset of vertices (rows of table 1) that have to be painted in the fourth colour. Finally $S_4$ = {10} – see figure 2. After this step the resulting matrix does not contain any row and we have got the first solution ($S_1$, $S_2$, $S_3$, $S_4$) and K=4 colours.

**Table 2:** The matrix $\mu(G)\backslash S_1$.

| | I | II | III | IV | V | VI | VII | VIII |
|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 0 | - | - | - | - | - | - |
| 5 | - | 1 | - | 1 | 0 | - | - | - |
| 6 | 1 | - | - | - | - | 0 | - | - |
| 7 | - | - | 1 | - | 1 | - | 0 | - |
| 8 | 1 | - | - | - | - | - | - | - |
| 9 | - | - | - | 1 | - | 1 | - | 0 |
| 10 | 1 | - | - | 1 | - | - | 1 | 1 |

**Table 3:** The matrix $\mu(G)\backslash S_1$ after applying the reduction rules.

| | II | V | VI | VII | VIII |
|---|---|---|---|---|---|
| 2 | 0 | - | - | - | - |
| 5 | 1 | 0 | - | - | - |
| 6 | - | - | 0 | - | - |
| 7 | - | 1 | - | 0 | - |
| 9 | - | - | 1 | - | 0 |
| 10 | - | - | - | 1 | 1 |

**Table 4:** The matrix $\mu(G)\backslash (S_1 \cup S_2)$.

| | II | V | VI | VII | VIII |
|---|---|---|---|---|---|
| 5 | 1 | 0 | - | - | - |
| 9 | - | - | 1 | - | 0 |
| 10 | - | - | - | 1 | 1 |

Now we will use the value K as a constraint for future steps. In other words, if the reduced matrix is not empty and at least K-1 colours have already been used we have to stop forward propagation steps and backtrack to the nearest branching point in order to try to find out a better solution. Thus, the first solution ($S_1$, $S_2$, $S_3$, $S_4$) for our example sets the terminating line shown in figure 2. For our example the nearest branching point is 6 (see figure 2). Now we can construct a new set $S_2$ = {8,2,6,10} for the level 2 (see figure 2). For the next step the reduced matrix contains three rows (5,7,9). After constructing a new set $S_3$ = {9,5} for level 3 the matrix is not empty. Thus, not less than 4 colours will be needed and we have to: 1) stop forward propagation; 2) backtrack to the row 2; and 3) select the first row (this is 7 – see figure 2) after the row 6 that has already been considered. Using this tree pruning method together with the reduction rules enables the total number of visited nodes in the search tree to be decreased significantly.

The subsequent steps are shown in the list below:

2. Executing operations for discovering the second subset of vertices (rows of table 1) that have to be painted in the second colour. Finally, $S_2$ = {8,2,7,9} – see figure 2.

3. Executing operations for discovering the third subset of vertices (rows of table 1) that have to be painted in the third colour. Finally $S_3 = \{5,6,10\}$ and the reduced matrix is empty. Thus, ($S_1$, $S_2$, $S_3$) is a better solution and K=3 colour.

Now the constraint for the future steps is changed to 3. Thus, the new terminating line in figure 2 is being set up. It is easy to check that the subsequent steps do not permit to improve the solution.

## 4 Derived Approximate Algorithms

The considered algorithm possesses the following primary advantage. Beginning from the first iteration we have got a solution of the problem. As a rule, this solution is not far away from the optimal solution and therefore might be appropriate for some practical applications. It is important that any new iteration does not make worse the first result and it can only improve it. Thus, either the number of iterations might be limited or any intermediate complete result can be checked for adequacy. This permits to suggest a number of approximate algorithms derived from the exact algorithm. It is very interesting to estimate how the optimal result depends on the number of iterations; how far is the first result from the optimal result, etc. The details of relevant experiments for real-world and randomly generated graphs will be reported in section 6.

## 5 General Description of FPGA-targeted Projects

Initially, the exact algorithm for graph colouring was described in C++ language and carefully debugged and tested in PC. Then C++ code was converted to a Handel-C project using the rules established by *Celoxica* [12]. The basic blocks of the hardware project are shown in figure 3.

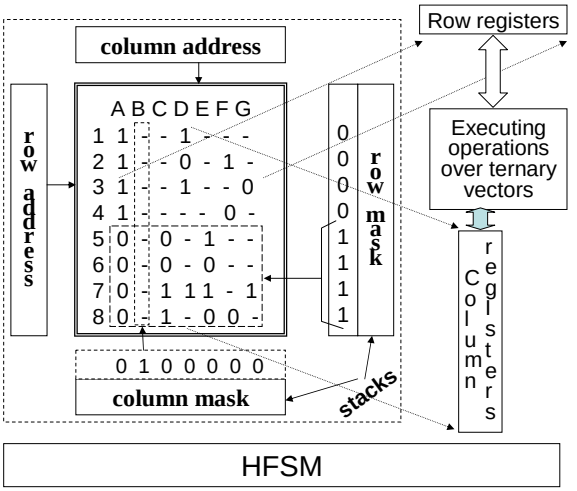


**Figure 3**: General structure of the hardware project for solving the graph colouring problem.

The initial and all the intermediate matrices are kept in the same RAM (constructed from embedded in FPGA memory blocks). Mask registers (for columns and for rows) make possible some rows and columns to be excluded from the matrix in order to select a minor (i.e. to specify the reduced matrix). The column/row address registers provide dual access allowing either a row or a column to be read (to a column/row register). The sequence of the required operations is generated by a hierarchical finite state machine [13].

## 6 Experiments and Comparison

The synthesis and implementation of circuits from the specification in Handel-C was done in Celoxica DK4 design suite [12] and implementation was finalised in Xilinx ISE 8.1 for xc2v1000-4fg456 FPGA (Virtex-II family of Xilinx [14]) available on RC200 prototyping board [12]. Note that each circuit includes not only components that are needed for problem analysis, but also auxiliary blocks for entering input data and visualising the results.

The results of experiments are presented in table 5 for the exact algorithm and in table 6 for approximate algorithms.

**Table 5:** The results for the exact algorithm.

| P(K) | N | L | $N_s$ | $F_h$ | $N_{clk}^h$ | $N_{clk}^s$ |
|---|---|---|---|---|---|---|
| 1(4) | 32 | 53 | 3481 | 26 | 587103122 | $7.3568*10^9$ |
| 2(3) | 18 | 21 | 4028 | | 166755 | $3.99*10^6$ |
| 3(4) | 28 | 50 | 4013 | | 102509429 | $1.77*10^9$ |
| 4(3) | 18 | 12 | 4028 | | 24122 | $0.426*10^6$ |
| 5(8) | 32 | 258 | 3481 | | 97356959 | $2.627*10^9$ |
| 6(7) | 32 | 241 | 3481 | | 122103423 | $2.534*10^9$ |

**Table 6:** The results for approximate algorithms.

| P(K) | N | L | $N_{clk}^h$ | $N_{clk}^s$ | $K_a$ | R |
|---|---|---|---|---|---|---|
| 1(4) | 32 | 53 | 24770 | $0.41 * 10^6$ | 4 | 1 |
| 2(3) | 18 | 21 | 4885 | $0.12 * 10^6$ | 3 | |
| 3(4) | 28 | 50 | 17612 | $0.31 * 10^6$ | 4 | |
| 4(3) | 18 | 12 | 3861 | $0.067 * 10^6$ | 3 | |
| 5(8) | 32 | 258 | 20388 | $0.506 * 10^6$ | 10 | |
| 6(7) | 32 | 241 | 21578 | $0.53 * 10^6$ | 10 | |
| 5(8) | 32 | 258 | 25877 | $1.04 * 10^6$ | 9 | 2 |
| 6(7) | 32 | 241 | 24803 | $0.625 * 10^6$ | 9 | |
| 5(8) | 32 | 258 | 2680182 | $1.11 * 10^9$ | 8 | 3 |
| 6(7) | 32 | 241 | 3318341 | $72.67 * 10^6$ | 8 | |
| 6(7) | 32 | 241 | 106231355 | $2.24 * 10^9$ | 7 | 4 |

Rows/columns of the tables contain the following data: P(K) – the number of problem instance, which can be optimally painted with K colours, N – the number of graph vertices; L – the number of graph edges; $N_s$ – the number of occupied FPGA slices; $F_h$ – the resulting circuit clock frequency in MHz; $N_{clk}^s$ – the number of clock cycles required for solving a given problem instance in software; $N_{clk}^h$ – the number of clock cycles required for solving a given problem instance in hardware; $K_a$ – the number of

colours obtained by the approximate algorithm, R – the number of respective iterations for $K_a$ colours.

The first problem (1) is the colouring of central and western European map. The next three problems (2-4) are encodings of micro-operations in microinstructions. The last four examples (5,6) were generated randomly.

We can summarise the results of experiments and analysis of alternative realisations as follows:

1. In average the FPGA-based implementations require one order of magnitude less clock cycles comparing to software implementations.

2. Approximate algorithms make possible to obtain either optimal or near-optimal results after just a few iterations. Thus, they are very well-suited for many practical applications. Besides, they enable us to reduce very significantly the execution time (up to four orders of magnitude, which can easily be calculated comparing data from tables 5 and 6).

3. The analysis of execution time has shown that approximate algorithms are very well suited for run-time solution of the graph colouring problem.

4. Preliminary tests of the considered algorithms in hardware description language based projects (VHDL, in particular) show that the number of FPGA slices can be significantly reduced.

5. The considered projects are easily scalable making possible to customise the required complexity and to link the projects with embedded systems for robotics working in real-time environment.

6. The used architecture (see figure 3) can easily be retargeted to solving some other combinatorial problems, such as matrix covering or Boolean satisfiability [11].

## 7 Conclusion

The first important contribution of the paper is the presentation of the known graph colouring problem in such a way that makes possible easy implementation of projects for solving this problem in commercially-available FPGAs. The second contribution is the analysis which permits to conclude that FPGA-based projects are more advantageous than relevant software projects executing on general-purpose computers (in terms of the number of clock cycles). The remaining significant issues considered in the paper are the exact algorithm for graph colouring and the derived approximate algorithms, which can be used for solving numerous engineering problems in the scope of robotics and embedded systems.

## 8 Acknowledgements

The authors would like to acknowledge Ivor Horton for his very useful comments and suggestions.

## 9 References

[1] G. Goossens, J. Van Praet, D. Lanneer, W. Geurts, A. Kifli, C. Liem, P.G. Paulin, “Embedded software in real-time signal processing systems: design technologies”, Proceedings of the IEEE, 85(3), pp 436-454 (1997).

[2] V. Subramonian, H.M. Huang, G. Xing, C. Gill, C. Lu, R. Cytron, “Middleware specialization for memory-constrained networked embedded systems”, http://www.cs.wustl.edu/~venkita/publications/rtsj_norb.pdf, visited on 25/09/2006.

[3] J. Ezick, “Robotics”, http://dimacs.rutgers.edu/REU/1996/ezick.html, visited on 25/09/2006.

[4] K.H. Rosen (ed.), *Handbook of Discrete and Combinatorial Mathematics*, CRC Press, USA (2000).

[5] R. Thomas, “The four color theorem”, http://www.math.gatech.edu/%7Ethomas/FC/fourcolor.html, visited on 25/09/2006.

[6] J. Culberson, “Graph coloring page”, http://www.cs.ualberta.ca/~joe/Coloring/index.html, visited on 25/09/2006.

[7] Y.L. Wu, M. Marek-Sadowska, “Graph based analysis of FPGA routing”, Proceedings of the European Design Automation Conference, Germany, pp 104-109 (1993).

[8] T.K. Lee, P.H.W. Leong, K.H. Lee, K.T. Chan, S.K. Hui, H.K. Yeung, M.F. Lo, J.H.M. Lee, “An FPGA implementation of GENNET for solving graph coloring problems”, Proc. of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), USA, pp 284-285 (1998).

[9] L.M. Pochet, M. Linderman, R. Kohler, S. Drager, “An FPGA based graph coloring accelerator”, http://klabs.org/richcontent/MAPLDCon00/Papers/Session_A/A2_Pochet_P.pdf, visited on 25/09/2006.

[10] A.D. Zakrevskij, Logical Synthesis of Cascade Networks, Science, Moscow (1981).

[11] I. Skliarova, *Reconfigurable Architectures for Combinatorial Optimization Problems*, Ph.D. thesis, University of Aveiro, Portugal (2004).

[12] Celoxica, “Celoxica products”, http://www.celoxica.com, visited on 25/09/2006.

[13] V. Sklyarov, “FPGA-based implementation of recursive algorithms,” Microprocessors and Microsystems, Special Issue on FPGAs: Applications and Designs, 28(5-6), pp 197–211 (2004).

[14] Xilinx, “Xilinx products”, http://www.xilinx.com, visited on 25/09/2006.