# High Speed Merged-datapath Design for Run-Time Reconfigurable Systems

Mahmood Fazlali[#*1], Ali Zakerolhosseini[#2], Asadollah Shahbahrami[#†3] and Georgi Gaydadjiev[*4]

[#]*Department of Computer Engineering, Shahid Beheshti University G.C, Tehran, Iran*
[1]fazlali@cc.sbu.ac.ir
[2]a-zaker@sbu.ac.ir

[*]*Computer Engineering Lab., Delft University of Technology, Delft, The Netherlands*
[†] *Department of Computer Engineering, University of Guilan, Rasht, Iran*
{[3]a.shahbahrami, [4]g.n.gaydadjiev}@tudelft.nl

***Abstract*—Datapath merging is an efficient high level synthesis method to merge Data Flow Graphs (DFGs), corresponding to two or more computational intensive loops. This process creates a general purpose datapaths (merged datapaths) instead of multiple datapaths that results in shorter bit-stream length and therefore reduces the configuration time in reconfigurable systems. The merged datapath, however has worse loop execution time. This paper represents two datapath merging algorithms to address this problem. These algorithms consider the impact of adding multiplexer's latency to the critical path delay of the merged datapath. The former algorithm merges DFGs from the biggest DFG to the smallest one to make high speed merged datapath. The latter merges DFGs in steps, and in the final step, it combines the resources inside the merged datapath to achieve additional reduction in configuration time. The proposed techniques are evaluated using several Mediabench applications. The experimental results show a significant reduction, up to 35% in loops execution time for the first algorithm and up to 27% reduction for the second algorithm in comparison to previous datapath merging algorithm.**

## I. Introduction

Many applications contain computational intensive loops, which in some cases can be accelerated by reconfigurable devices such as FPGAs. On the other hand, the FPGA resources are limited. In order to share FPGA resources among different applications, run-time reconfiguration is employed when the hardware is needed [1]. However, the run-time reconfiguration imposes a considerable overhead to the performance of the system. Therefore, the configuration should be done as efficient as possible.

The bit-stream length and the configuration time of the hardware are directly proportional. In fact, the time of transmitting bit-stream into FPGA corresponds to the configuration time [2] and therefore, reducing the bit-stream length amortizes the configuration time. Previous research has been carried out to reduce the configuration time by using compression and caching techniques. For instance, the authors in [3,4] and [5] used compression and caching techniques to reduce the bit-stream length, respectively. Although these techniques reduce the configuration time, they are costly. In order to prevent the additional cost, the configuration time reduction can be addressed during High Level Synthesis (HLS). Mostly HLS is used to create the Data Flow Graphs (DFG) with a number of iterations for the computational intensive loops [6, 7]. The HLS shares resources of the DFGs to make a more generic datapath. Therefore, it reduces the hardware cost of the datapath. The synthesis process comprises the major tasks of scheduling, resource allocation, resource binding, and interconnection binding [8].

Making a multimode datapath instead of multiple datapaths can reduce the hardware cost [9]. Datapath merging is an efficient HLS approach that makes a multimode datapath for partially reconfigurable systems [10,11]. We showed in [12] that datapath merging is a suitable method to reduce the datapath configuration time. The method in [12] heuristically chooses a sequence of DFGs to merge together, consequently, it cannot optimize the configuration time of the merged datapath. On the other hand, datapath merging algorithms add multiplexers in the input port of the functional units in the merged datapath, and as such, increase the execution time of the loops via the merged datapath. Merging more DFGs also causes sharing more functional units in merged datapath. This means that the final merged datapath employs larger multiplexers. If the multiplexer is on the critical path of the merged datapath, it will increase the loops execution time. In case of loops with many iterations; this time-overhead will be unacceptable.

In order to provide a high speed merged datapath, we present two new datapath merging algorithms. The former merges DFGs to reduce the configuration time while it avoids large multiplexer on the critical path of the merged datapath. The latter algorithm merges DFGs in steps in the same way, and in the final step it combines the resources inside the merged datapath to achieve the additional reduction in configuration time. The rest of the paper is organized as follows. In section II, trade-off between the conflicting factors, configuration time reduction, and increase in loop execution time, in datapath merging is presented. The proposed datapath merging algorithms are presented in section III. We evaluate the proposed algorithms in section IV. Ultimately, section V concludes the paper.

## II. Loop Execution Time via the Merged Datapath

Although datapath merging reduces the datapath configuration time, it increases the loop execution time. Fig. 1 illustrates two DFGs ($G_1$ and $G_2$) and two merged datapaths ($MDP_1$ and $MDP_2$) corresponding to merging $G_1$ and $G_2$. Due to the reduced hardware usage in $MDP_1$, it has shorter configuration time in comparison to $MDP_2$. On the other hand, $MDP_1$ has a multiplexer on the input port of node $a_5/b_3$. The divider in $a_5/b_3$, has the longest hardware delay among the functional units and $a_5/b_3$ is on the critical path delay of the merged datapath. Therefore the clock rate of the merged data path $MDP_1$ is lower than $MDP_2$ and consequently, execution time of the loops via the $MDP_1$ is longer than their execution time via $MDP_2$.

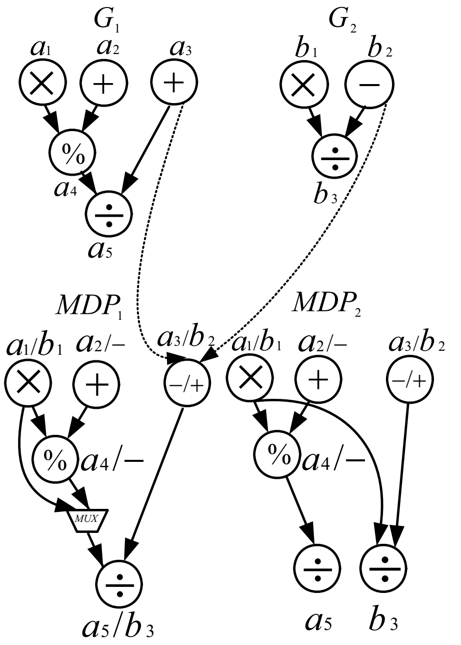

Fig. 1. Merging $G_1$ and $G_2$ to create merged datapaths $MDP_1$ and $MDP_2$.

Hence, datapath merging results in slowing down the hardware. Although pipeline implementation of the merged datapath can reduce this overhead, it can be unacceptable for the loops with many iterations. So, to avoid prohibitive increase in loop execution time, we should consider hardware latency in datapath merging.

## III. The Proposed Datapath Merging Algorithms

Since merging DFGs all at once is an *NP*-complete problem [10], DFGs should be merged in steps to reduce the configuration time. In the proposed algorithms, we merge DFGs in such a way multiplexers are not added to the critical path of the merged datapath. The DFGs are merged together starting from the biggest DFG to the smallest one. In the first proposed algorithm (High speed-1), the merging is performed in a number of steps. Each step corresponds to merging a DFG onto the merged datapath that includes the three stages below.

1- Making the compatibility graph between *MDP* and DFGs;
2- Finding the bounded execution time-weighted clique;
3- Reconstructing the pipelined merged datapath.

In the first stage, we should find the similarity between the functional units of the DFG and the functional units of the merged datapath. To this end, we make the compatibility graph in each stage of datapath merging.

A compatibility graph $G_c = (N,A)$ is an undirected weighted graph. Each node $n \in N$ shows the merging possibility between a vertex from *MDP* and a vertex from *G*, or a merging possibility between an edge from *MDP* and an edge from *G*. Each arc $a=(n,m) \in A$ illustrates that having both nodes, *n* and *m,* together in merging, do not merge two vertices from a DFG $G_i$ together. It means they are compatible. The weight for each node, $w_i$, represents the reduction in merged datapath configuration time resulted from merging indicated by the node. For example as illustrated in Fig. 2, two vertices $a_1 \in G_1$ and $b_1 \in G_2$ can be merged together. So, the node $a_1,b_1$ is created in $G_c$ for this merge that its weight indicates to the 360 *ns* reduction in configuration time resulted from merging the vertices. This weight is the difference between the configuration time of the obtained vertex in *MDP* and configuration time of the original vertices before merging [12]. In this way, the compatibility graph "$G_c$" for two DFGs $G_1$ and $G_2$ is created by using the method was explained above.

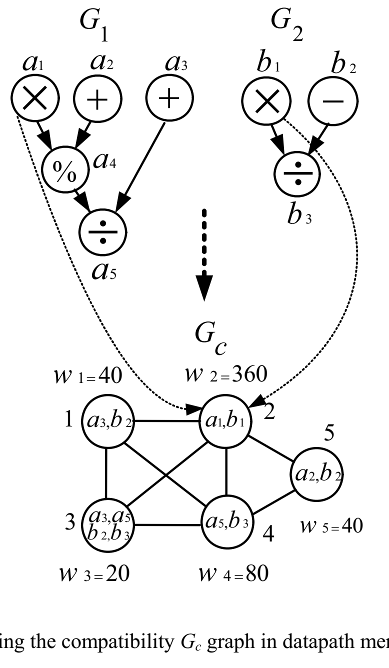

Fig. 2. Making the compatibility $G_c$ graph in datapath merging [12].

In the second stage of the High speed-1 algorithm, we should find a clique in the compatibility graph to make a high speed merged datapath. By finding the maximum weighted clique from the compatibility graph in this stage, the merged datapath which has minimal configuration time is made [12]. Fig. 3 illustrates two cliques and their corresponding merged datapath for the compatibility graph in Fig. 2. The first one is the maximum weighted clique and its corresponding merged datapath that are illustrated in Fig.3.a. The second one is another clique and its corresponding merged datapath which are illustrated in Fig.3.b. Although the merged datapath configuration time in Fig.3.a is less than the merged datapath configuration time in Fig.3.b, the execution time of loops in Fig3.a is longer than their execution time in Fig.3.b. The clique in Fig.3.b avoids adding multiplexer on the input ports of the functional unit on the critical path delay of the merged datapath.

To find the high speed merged datapath, we need a clique as the one illustrated in Fig.3.b. By using this clique the desired merged datapath is obtained. So, we need a similar clique to solve the problem. Therefore:
Given a compatibility graph $G_c$ for $k$ number of DFGs, $G_i$ $i=1...k$, Bounded execution-time weighted clique, $M_b$, is a maximal weighted clique in $G_c$ that has bounded critical path delay for the corresponding merged datapath.

The algorithm which finds the Bounded execution time-weighted clique, $M_b$, in $G_c$ should take into consideration every increase of the execution time of the loops via the merged datapath. Finding $M_b$ in $G_c$ is similar to finding the maximum weighted clique. The maximum weighted clique problem is known to be an *NP*-hard problem [13]. To find the maximum weighted clique from a graph, [14] employs the Branch&Bound algorithm and optimizes its execution time. This method for searching the problem space and its optimizations is suitable for solving our problem. This way, we modified the Branch& Bound algorithm in [14] to find $M_b$. Our algorithm to find bounded execution-time weighted clique is a recursive Branch&Bound function that searches the nodes in a compatibility graph to find the desired clique. It considers all nodes and also decides which node is probable to be in the bounded-execution time-weighted clique in each branch of recalling the function. Our modifications are as follows:

- Considering the loops execution in each step of Branch& Bound algorithm by employing a bound for the loops execution time via the merged datapath.
- Using configuration time bound and execution time bound in Branch&Bound Algorithm to cut the branches and limit problem search space.

The former modification chooses the clique among all cliques in each branch that has the bounded execution time for the loops. It prevents from adding big multiplexers on the critical path of the merged datapath corresponding to this clique. The latter reduces the search space of the algorithm by cutting branches which are improbable to find the bounded execution time-weighted clique.

In the last stage of High speed-1 algorithm, the bounded-execution time-weighted clique, $M_b$, is used to reconstruct *MDP*, and produce the next merged datapath. Each node from $M_b$ indicates a merging possibility between an edge (a vertex) from *MDP* and an edge (a vertex) from $G_j$. Other vertices and edges which cannot be merged are added to *MDP* without merging. It should be notified *MDP* is pipelined to reduce the overhead of loops. These processes are repeated for merging all DFGs onto the merged datapath until the last merged datapath resulted from algorithm High speed-1 is made.

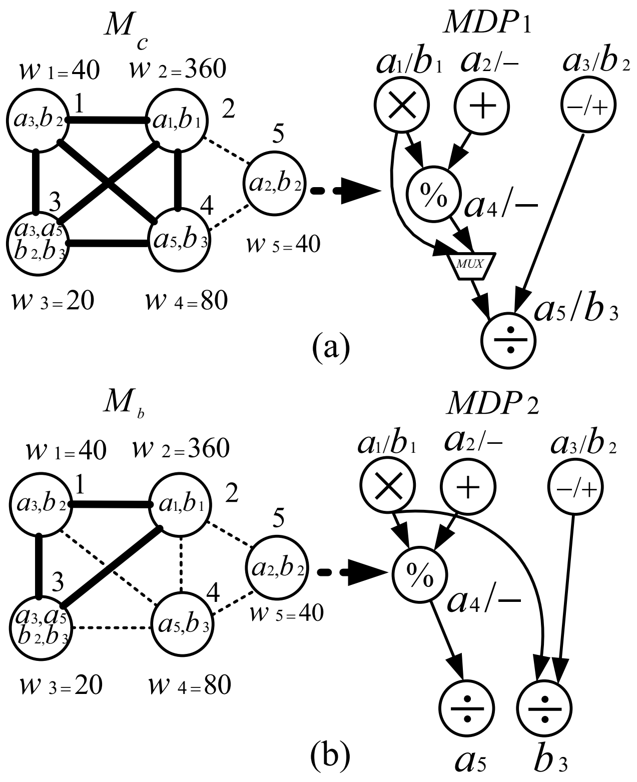

Fig. 3. Reconstructing the merged datapath *MDP* using the maximum weighted clique, and the bounded execution-time weighted clique.

The second proposed datapath merging algorithm (High speed-2), includes all steps of the High speed-1 merging algorithm plus an extra step. In this step all merging possibility among the resources inside the merged datapath is considered as a compatibility graph (intra-merged datapath resource sharing). To do this, we used the compatibility graph in [11]. Then the bounded execution time-weighted clique in this graph, $M_b$, is determined. By finding $M_b$ in the compatibility graph and reconstructing the *MDP* using this clique, the merged datapath resulted from the High speed-2 merging algorithm is created.

## IV. Experimental Results

We have implemented the proposed merging algorithms and the datapath merging algorithm in [12], as well. The algorithm in [12] is based on the maximum weighted clique, while our algorithms use the bounded-execution time-weighted clique. All algorithms perform resource allocation, resource binding, and interconnection binding simultaneously with the aim of reducing the configuration time.

There are some computational intensive loops in each application in Mediabench suite which have the largest share of the execution time [15]. The entity of the loops makes them suitable for the execution by reconfigurable computer. To merge the DFGs correspond to the loops, initially benchmarks should be converted to intermediate representation. This way, each program was compiled using the GCC compiler and was profiled to determine which loops contributed the most to the program execution time. For each such loops, a DFG was generated from the loop body RTL code.

The configuration time of a bit-stream in a FPGA can be estimated as [(size of bit-stream) / (configuration clock Frequency)] [3]. After obtaining the bit-stream of the functional units and multiplexers from ISE 10.2, their configuration time was calculated. In our experiments, we used the configuration clock frequency for the FPGA which is 100 *Mbps* as in the case of Virtex5-xc5vlx.

TABLE I
The Loops Execution Time in Empeg2-decoder Application (*ms*) Where Loops Have 20000 Iterations

| **Datapath Merging Algorithms** | **Loops Execution Time (ms)** | | | |
|---|---|---|---|---|
| | **Loop1** | **Loop2** | **Loop3** | **Total** |
| Algorithm in [12] | 0.845 | 1.689 | 2.252 | 4.786 |
| High speed-1 algorithm | 0.633 | 1.209 | 1.280 | 3.122 |
| High speed-2 Algorithm | 0.645 | 1.243 | 1.631 | 3.519 |

For the first experiment we applied our algorithms and the algorithm proposed in [12] to three DFGs corresponding to the loops of MPEG2-decoder. To apply each algorithm, initially DFGs were scheduled using ASAP scheduling algorithm. Then, the datapath merging algorithms was applied to the DFGs to make the pipelined merged datapath. Afterward, the loops execution time via the merged datapath is calculated for each algorithm. Where the loops have 100 iterations, execution time of the loops for all datapath merging algorithms are the same. That means all algorithms make the merged datapath by using the maximum weighted clique in this situation. We repeated the experiment where the loops have 20000 iterations. Table II shows the execution time of the loops. The results indicated that whenever loops have 20000 iterations, High speed-2 algorithm has shorter execution time than the algorithm in [12] and High speed-1 algorithm achieved the shortest execution time among three algorithms.

We repeated the previous experiment for several applications from Mediabench suite where loops have 20000 iterations. We applied the datapath merging algorithms to three DFGs of the MPEG2-decoder, three DFGs of MPEG2-encoder, three DFGs of EPIC-decoder, three DFGs of EPIC-encoder and, two DFGs of G721 benchmark. After achieving the merged datapaths from applying the algorithms, for each application, the total increase in loops execution time compared to the loops execution time in original DFGs was calculated.

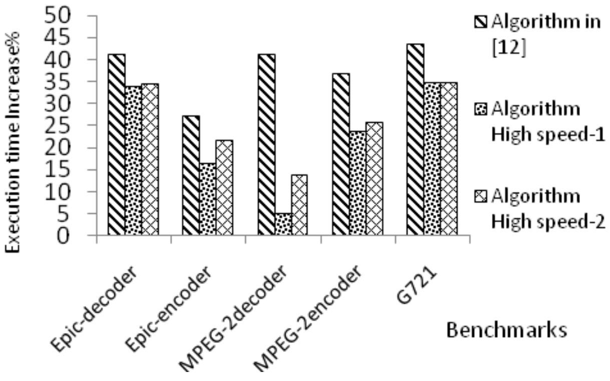

Fig. 4. Percentage increase in loops execution time in the proposed datapath merging algorithms and the algorithm in [12] compare to loops execution time in original DFGs

As illustrated in Fig. 4, the loops execution time in the proposed algorithms is lower than the algorithm in [12] where loops have 20000 iterations. This means that the proposed datapath merging algorithms performs better than maximum weighted clique algorithm. The minimum improvement in loops execution time, compare to the algorithm in [12], is 5% for High speed-2 algorithm in EPIC-encoder application. The highest improvement is 35% for High speed-1 algorithm in MPEG-2decoder. This shows that by increasing the number of functional units in DFGs, there might be multiplexers that increase the critical path delay of the merged datapath. the proposed algorithms prevent from adding such multiplexers to the critical path. Similar to High speed-1 algorithm, the hardware resulted from High speed-2 algorithm has execution time shorter than algorithm in [12]. Moreover except G721, High speed-2 algorithm reduces configuration time more than High speed-1 algorithm. This configuration time reduction advantage is that of it has longer execution time for the loops in comparison to High speed-1algorithm.

Fig. 5 shows the configuration time reduction percentage after applying the above-mentioned algorithms to the DFGs. It depicts that the maximum improvement in loops execution time, in MPEG-2decoder application, was gained at the cost of additional configuration time up to 5% in High speed-1 algorithm. The maximum improvement in loops execution time in High speed-2 algorithm was gained besides 1% more reduction in configuration time in comparison to the algorithm in [12]. It shows the last stage of High speed-2 algorithm have merged the functional units which are not on the critical path delay of the merged datapath. In addition, High speed-2 algorithm performs better than the algorithm in [12] and High speed-1 algorithm in reducing the configuration time for all applications except the application G721. It is because, there are just two DFGs in G721 and the previous algorithm could heuristically find the best solution for configuration time reduction. Overall, the proposed datapath merging algorithms are suitable for reducing the configuration time and creating high speed merged datapath.

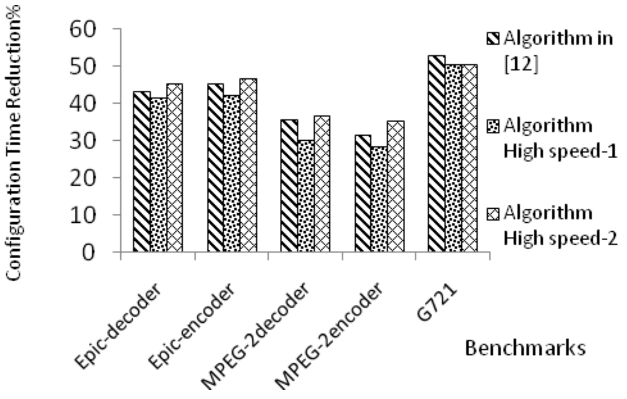

Fig. 5. Percentage reduction in configuration time for the proposed datapath merging algorithms and the algorithm in [12]

## V. Conclusion

This paper has presented two datapath merging algorithms for run-time reconfigurable systems. The ultimate purpose of these algorithms is making high speed merged datapath in addition to the reduction in configuration time for the computational intensive loops. This way, the similarity between DFGs was considered as a compatibility graph. Then the High speed merged datapath was made by finding the bounded-execution time-weighted clique from the compatibility graph in each step of datapath merging algorithm. We applied the proposed algorithms to merge the DFGs corresponding to the loops of Mediabench applications. The former algorithm reduced the loops execution time up to 35% in comparison to the previous datapath merging algorithm at the cost of additional configuration time up to 5%. The latter, reduced loops execution time up to 27% besides 1% reduction in configuration time. We conclude that the proposed datapath merging algorithms can efficiently lowered the loops execution time after creating merged datapath for reconfigurable systems.

## Acknowledgment

This research was supported by the Iran Telecommunication Research Center (ITRC) in the context of the project T/500/3462.

## Refrences

[1] Z. Li, "Configuration Management Techniques for Reconfigurable Computing", Ph.D. Thesis, Northwestern University, June, 2002.

[2] M. Rollmann and R Merker, "A Cost Model for Partial Dynamic Reconfiguration", *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation* (*SAMOS*), pp.182-186, Greece, July, 2008.

[3] P. J. Hwa, T. Mitra and W.F. Wong, "Configuration Bit-stream Compression for Dynamically Reconfigurable FPGAs", *IEEE/ACM International Conference on Computer Aided Design* (*ICCAD*), pp. 766–773, CA, USA, November, 2004.

[4] F. Farshadjam, M. Dehghan, M. Fathy and M. Ahmadi, " A New Compression Based Approach for Reconfiguration Overhead Reduction in Virtex-Based RTR Systems", *Elsevier journal on Computers & Electrical Engineering*, Vol. 32 , No. 4, pp 322–347, 2006.

[5] K. Compton and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software", *ACM Computing. Surveys*, Vol. 34, No.2, pp. 171-210, 2002.

[6] Y. D. Yankova, G.K. Kuzmanov, K.L.M. Bertels, G. N. Gaydadjiev, Y. Lu, S. Vassiliadis "DWARV: Delft Workbench Automated Reconfigurable VHDL Generator" *17th International Conference on Field Programmable Logic and Applications* (*FPL*), pp. 697-701, Amsterdam, The Netherlands, August, 2007.

[7] R. J. Meeuws, Y. D. Yankova, K.L.M. Bertels, G. N. Gaydadjiev, S. Vassiliadis," A Quantitative Prediction Model for Hardware/Software Partitioning", *17th International Conference on Field Programmable Logic and Applications* (*FPL*), pp.735-739, Amsterdam, The Netherlands, August, 2007.

[8] Ph. Coussy, A. Morawiec "High-Level Synthesis from Algorithm to Digital Circuit", *Springer*, 2008.

[9] L Chiou, S Bhunia, and K. Roy, "Synthesis of Application-Specific Highly Efficient Multi-mode Cores for Embedded Systems", *ACM Transaction on Embedded System Computing* (*TECS*), vol.4, no.1, pp. 168-188, 2005.

[10] N. Moreano, Ed. Borin, C. D. Souza, and G. Araujo, "Efficient Datapath Merging for Partially Reconfigurable Architectures", *IEEE Transactions on Computer-Aided Design of Integrated Circuit And Systems*, vol. 24, no. 7 ,pp. 969-980, July 2005.

[11] M. Fazlali, M.K. Fallah, M. Zolghadr, A. Zakerolhosseini, " A New Datapath Merging Method for Reconfigurable System", *5th International Workshop on Applied Reconfigurable Computing* (*ARC*) LNCS 5453, Karlsrohe Germany, pp.157-168 , March, 2009.

[12] M. Fazlali, A. Zakerolhosseini, M. Sabeghi, K. Bertels, and G. Gaydadjiev " Datapath Configuration Time Reduction for Run-time Reconfigurable Systems " *International Conference on Engineering of Reconfigurable Systems and Algorithms*, (*ERSA*), Las Vegas Nevada, USA, July 2009.

[13] M. Garey and D. S.Johnson, "Computers and Intractability-A Guide to the Theory of NP Completeness", San Francisco, CA, Freeman, (1979).

[14] P. R. J Ostergard, "A New Algorithm for the Maximum-Weight Clique Problem", *Nordic Journal of Computing*, vol. 8, no. 4, pp. 424-436, 2002.

[15] C. Lee, M Potkonjak, W. S. Mangione, "Mediabench: a Tool for Evaluating and Synthesizing Multimedia and Communication Systems", *13th Annual IEEE/ACM International Symposium on Microarchitecture* (*MICRO*), December, California the USA, pp.330-335, 1997.