An Enhanced Network Flow Algorithm for Temporal Partitioning into Reconfigurable Architectures

Abdo Azibi*, Ramzi Ayadi, Med Lassaad Kaddachi
Department of Electronics College of Technology at Alkharj, Saudi Arabia.

* Corresponding author. Tel.: +966504792206; email: aazibi@tvtc.org.sa
Manuscript submitted January 10, 2015; accepted March 8, 2015.
doi: 10.17706/jcp.10.3.176-183

Abstract: In this paper, we present a novel temporal partitioning methodology for dynamically reconfigurable computing systems to reduce the communication costs of the design. This aim can be reached by minimizing the transfer of data required between design partitions. Our algorithm use the network flow-based multi-way task partitioning algorithm to minimize communication costs for temporal partitioning.. The proposed methodology was tested on several examples on the Xilinx Virtex-II pro. The results show significant reduction in the communication cost compared with others famous approaches used in this field.

Key words: Temporal partitioning, dynamically reconfigurable architectures, communication costs, FPGA.

1. Introduction

Reconfigurable architectures (RA) constructed from one or more general purpose processors (GPP) and a set of reconfigurable processing units (RPU), then they combine high performance with flexibility and programmability. In general, reconfigurable architectures such as field programmable gate array (FPGA) are able to reconfigure rapidly to any custom desired form. Hence, there is growing interest in dynamically reconfigurable architectures (DRA) [1], [2] because logic utilization can be dramatically improved by time-sharing logic. Resources can be effectively reused, cutting hardware costs and improving performance. A program or application can be modeled by a data flow graph where the nodes represent computation tasks (integer addition, real multiplication, and integer comparison) and the edges the data dependency between nodes. The functionality of the nodes is specified in a conventional high level language like C++ or VHDL (Very high speed integrated circuit). In this context for a given application, the resources on the device are not allocated to only one operator but to a set of operators that must be placed at the same time and removed at the same time. With this, an application must be partitioned in sets of operators (nodes) called partition such as the size of each partition must be less or equal than a total area of device. The partitions will then be successively implemented at different time on the device. This process, called temporal partitioning, allows an application to be sequentially computed, by allowing a temporal sharing of resource among different sets of operators.

In the literature, many methods have developed by different authors to solve the temporal partitioning problem. In [3]–[5] authors used traditional scheduling methods, such as list scheduling. The idea behind the list scheduling approach is first to place all the nodes of a graph representing the problem to be solved in a list. A new partition (also called configuration) is built stepwise by removing nodes from the list and

allocating them to the partition until the size of the partition reached a given size limit (the size of the FPGA). A new partition is then created and the process is repeated until all the nodes from the list are placed in partitions. Others authors extended existing scheduling of high-level synthesis [6]–[8]. In [10] the authors used ILP algorithm. The ILP is a mathematical method for determining a way to achieve the best outcome, such as lowest latency.

The main problem of the ILP approach is its high execution time; therefore, the algorithm can only be applied to small examples. In [11] authors combined the force directed scheduling (FDS) algorithm and network flow algorithm to reduce the whole latency and the communication cost at the same time. In [12] author used leveling node method to determine the communication cost. However, for each end of stage the method is not a min cut just only a leveling cut. The network flow algorithm has been used to reduce the communication cost across temporal partitions. The first network flow algorithms has been used in [12], [13]. In this paper we present a new approach based on graph partitioning to solve efficiently the temporal partitioning problem for dynamically reconfigurable computing systems.

2. Temporal Partitioning Problem

In dynamically reconfigurable computing systems, the temporal partitioning problem can be formulated as a graph-based problem. A program or application can be represented by a data flow graph. A DFG is a directed acyclic graph (DAG) G=(V, E), where V is a set of nodes and E is a set of edges. For each node $T_i \in V$, there exists an area $A(T_i)$. Each node $T_i \in V$ represents a functional operation implementation and, correspondingly $A(T_i)$, represents the operation size. A directed edge $e_{i,j} \in E$ exists if the function represented by T_i depends on the function output represented by T_j, then a directed edge $e_{i,j} \in E$ represents the data dependency between nodes(T_i, T_j). We define the weight $W_{i,j}$ of $e_{i,j}$ as the amount of data transferred from T_i to T_j.

A temporal partitioning P of the graph G=(V, E), is its division into some disjoints partitions such as: P={$P_1, \ldots, P_k$}. A temporal partitioning is feasible in accordance to an available reconfigurable device H with area $A(H)$ and pins $T(H)$ (number of programmable input/outputs (I/Os) per device) if:

$$\forall P_i \in P \; we \; have \sum_{T_i \in P_i} A(T_i) \in A(H) \tag{1}$$

$$\forall P_k \in P, e_{ij} \; we \; have \; \frac{1}{2}\left(\sum_{(e_{ij} \cap P_k) \neq \emptyset \; and \; (e_{ij} - P_k)} W_{i,j} \right) \leq T(H) \tag{2}$$

Further, given a temporal partitioning P of the graph G=(V, E) into K disjoints temporal partitions P={P_1, ... ,P_k}. Based on Eq. (1)

$$\forall P_i \in P, A(P_i) \leq A_{RM}$$
$$A(H) = A_{RM} = (W_M * H_M) \tag{3}$$

A_{RM}, W_M and H_M represents the area, the width and the height of reconfigurable Module, respectively. Therefore, the area of the reconfigurable module should be grater or equal than $A(P_i)$.

3. Proposed Algorithm

The above aim can be reached by minimizing the transfer of data required between design partitions. Hence, our algorithm aims to find a way of graph partitioning that gives the optimal solution in term of communication cost between design partitions.

In this work, our main goal is how finding the way of partitioning the graph into K disjoint partitions such as the communication cost between design partitions has lowest value. This section shows how achieving a good solution to such graph partitioning problem. Given a temporal partitioning of G=(E, V) into K disjoint partitions P={$P_1, \ldots, P_k$}, the communication cost ,$CC(P_m)$, of partition Pm has been defined in [13] as follow:

$$CC(P_m) = \frac{1}{2}\left(\frac{\sum_{T_i \in P_m; T_j \in \overline{P}_m} W_{ij}}{|P_m|}\right) \quad (4)$$

This implies that:

$$TCC = \sum_{m=1}^{k} CC(P_m) = \frac{|V|}{2}\sum_{i=1}^{K}\left(\frac{\sum_{T_i \in P_m; T_j \in \overline{P}_m} W_{ij}}{|P_m||\overline{P}_m|}\right) \quad (5)$$

where TCC is the total communication cost. $|\overline{P}_m| = |P| - |P_m|$ is the number of nodes outside the partition P_m. Hence, we have the total number of nodes: $|\overline{P}_m| + |P_m| = |V| = n$

From the previous discussion, we formulate the temporal partitioning problem as follows.

Inputs: given a data flow graph G=(V, E)

Constraints:

1) $V = \bigcup_{i=1}^{k} T_i$. where K is a number of partitions
2) All dependency constraint relations are satisfied for all K partitions, let $Dep(T_i)$ denote the dependency constraint of a node T_i. For tow nodes T_i and T_j , we define $Dep(T_i) \leq Dep(T_j)$ if T_i must be scheduled no later than T_j.
3) $A(P_i) \leq A(H), 1 \leq i \leq k.$ Where $A(P_i)$ denoted the area of partition P_i and $A(H)$ denoted the total area of the device.

Objective: divided G into k partition such that the k and the total communication cost between all partitions are minimized with respecting all constraints.

In this paper we present a new temporal partitioning algorithm based on network flow algorithm which tries to achieve the minimum number of partitions with an interesting reduction of total communication costs between all partitions

3.1. Definitions

Given a data flow graph G=(V, E), we define:

1) A primary input (PI) node is a node without any predecessor.
2) A primary output (PO) node is a node without any successor.
3) The inputs adjacency matrix M_{in} as follow:
4) For tow nodes (T_i, T_j), $M_{in}(i,j) = W_{ij}$ if the function represented by T_j represent an input for the function represented by T_i.
5) The outputs adjacency matrix M_{out} as follow:
6) For tow nodes (T_i, T_j), $M_{out}(i,j) = W_{ij}$ if the function represented by T_j represent an output for the function represented by T_i
7) The degree matrix D as follow:

$$deg(T_i) = D(i,i) = \sum_{j=1}^{n}\big(M_{in}(i,j) + M_{out}(i,j)\big), D(i,j) = 0 \quad (6)$$

8) Given a temporal partitioning of G=(E, V) into K disjoint partitions P={P_1, ... ,P_k};
9) We define the cut size, $Cut(P_m)$, of partition P_m, as follows:

$$Cut(P_m) = (P_m, \overline{P}_m) = \sum_{T_i \in P_m; T_j \in \overline{P}_m} W_{ij} \quad (7)$$

This implies that: Total cut size:

$$TCut(P_m) = \sum_{m=1}^{k} Cut(P_m) = \sum_{m=1}^{k} \sum_{T_i \in P_m; T_j \in \overline{P}_m} W_{ij} \quad (8)$$

The use of the network flow approach to solve temporal partitioning problems was first proposed in [1], [2] is called FBB algorithm. The method is a recursive bipartition approach that successively partitions a set of remaining nodes in two partitions, one of which is a final partition, whereas a further partition step must be applied on the second one. Note that in the original network flow based algorithm [1], two nodes are randomly selected as the source nodes s and the sink nodes t so as to bipartition of the application into two parts P and $\overline{P}$ such that $s \in P$ and $t \in \overline{P}$, but our approach is slightly different. We modify step 1) of the FBB algorithm such that a network is constructed from G=(V, E) by introducing a new strategy for selecting nodes source s and sink t. Moreover, in steps 3) and 4), we consider precedence constraint effects when we do the partitioning. On the other hand, we choose the node with the smallest degree and collapse it to s or t such that the area constraint is satisfied as soon as possible. This is different from the original FBB algorithm in [1], which randomly selects one node.

3.2. Lemma 1

Given a temporal partitioning of G (V, E) into k disjoint partitions P={P_1, P_2,..., P_k}.

Based on Eq (5) and Eq(8) the problem of communication cost minimization can be expressed as: Minimize TTC subject to minimize $TCut(P_m)$ [13].

Then, Based on Lemma 1 : minimize TCC subject to minimize $\frac{|V|TCut(P_m)}{2|P_m||\overline{P}_m|}$ subject to minimize TCut(P_m).

3.3. Feasible Partitioning Algorithm

A direct extension of FBB to multi-way partitioning is to apply recursively the max-flow min-cut process to find one partition at a time that meets the area constraint until every node in G is assigned to a partition as shown in Fig. 1. A feasible partition P_i is a sub-partition of V that satisfies the area constraint, i.e. $A(P_i) \leq A(H)$. To partition the DAG into as a small number of partitions as possible, it is desirable to find one large feasible partition at a time. FP is an algorithm that repeatedly computes max-flow min-cut in to find a feasible partitions with as optimal communication cost as possible.

In step 1), a network is constructed from G=(V, E) by selecting two nodes source s and sink t.

In step 2), the maximum flow is pushed through the network. Incremental flow computation is employed, as only additional flow is added to saturate the edges from iteration to iteration in order to find a feasible min-cut.

In step 3), if $A(P_i) \leq A(H)$, then the nodes on one side of the min-cut are collapsed to the s seed node. This not only selects a feasible node under the area constraint but also satisfies the precedence constraint and optimizes the communication cost. According to lemma 1 and lemma 2, optimizing communication cost result by selecting the node with smallest degree. This process may not find a suitable node T_i of the $\overline{P}$ part, meaning that the($A(P_i) + A(T_i) > A(H)$) such that the sub-partition is the most optimal partition in terms of communication cost.

Minimize Communication cost Algorithm
Input: *A data Flow graph* ***G (V, E)***
Output: *temporal partitioning with optimal communication cost*
Step1: *Add a source s and a sink t with:*
Add edges between $\boldsymbol{s}$ *and the nodes with primary input*
Add edges between $\boldsymbol{t}$ *and the nodes primary output*
$FP \leftarrow \emptyset$;
Step2: *Compute max-flow from s to t. let* $\boldsymbol{P}$ *be the set of nodes reachable from s through the augmenting path, and* $\overline{\boldsymbol{P}} = \boldsymbol{V} - \boldsymbol{P}$, $\boldsymbol{Cut}(\boldsymbol{P}) = (\boldsymbol{P}, \overline{\boldsymbol{P}})$.
Record the cut size.
Step 3*: If* $\boldsymbol{A}(\boldsymbol{P}) \leq \boldsymbol{A}(\boldsymbol{H})$, *then begin*
For all nodes $\boldsymbol{T_i} \in \boldsymbol{adjacent}(\boldsymbol{P})$, *and all* $\boldsymbol{T_i} \in \overline{\boldsymbol{P}}$ *do begin*
If $(\boldsymbol{A}(\boldsymbol{P}) + \boldsymbol{A}(\boldsymbol{T_i})) < \boldsymbol{A}(\boldsymbol{H})$ *and precedence constraint is satisfied*
For all $\boldsymbol{T_j} \in \overline{\boldsymbol{P}} - \boldsymbol{T_i}$, *then begin*
Select the minimum degree of $\boldsymbol{T_i}$
Collapse $\boldsymbol{T_i}$ *in* $\boldsymbol{P}$ *to* $\boldsymbol{s}$
Collapse to $\boldsymbol{s}$ *a node* $\boldsymbol{T_i} \in \overline{\boldsymbol{P}}$ *incident on* $\boldsymbol{s}$.
End of if
Else
$FP \leftarrow P; V \leftarrow V - P$;
End of for-loop
End of if
Go to ***step2***.
Step4: *If* $\boldsymbol{A}(\boldsymbol{P}) > \boldsymbol{A}(\boldsymbol{H})$, *then begin*
For all nodes $\boldsymbol{T_i} \in \boldsymbol{adjacent}(\overline{\boldsymbol{P}})$, *and all* $\boldsymbol{T_i} \in P$ *do begin*
If $(\boldsymbol{A}(\boldsymbol{P}) + \boldsymbol{A}(\boldsymbol{T_i})) < \boldsymbol{A}(\boldsymbol{H})$ *and precedence constraint is satisfied*
For all $\boldsymbol{T_j} \in P - \boldsymbol{T_i}$, *then begin*
Select the minimum degree of $\boldsymbol{T_i}$
Collapse $\boldsymbol{T_i}$ *in* $\overline{\boldsymbol{P}}$ ***to t***
Collapse to $\boldsymbol{t}$ *a node* $\boldsymbol{T_i} \in P$ *incident on t.*
End of if
End of for-loop
End of if
Go to ***step2***.

Fig. 1. Feasible partitioning algorithm.

In step 4), the nodes on one side of the min-cut are collapsed to the t seed node. If a feasible partition is found, then $V = V - P$ and continues to find the next partition until every node is assigned a partition. According to step 2) of the FP algorithm, we can also obtain a set of min-cuts with different sizes.

Fig. 2 illustrates an example of using network flow for multi-way partitioning. In this case, the area constraint of a partition is 15, $A(H) = 15$. In each iteration, max-flow is computed and a min-cut is found. Then all the nodes on the smaller side of the min-cut are collapsed to form one seed node, so that more flows can be pushed through the network. In iteration 2, it is seen that if the node with minimum degree is collapsed to the sink t, sink t will have one out-edge which contradicts the precedence constraint as shown in Fig. 2(b). Hence, it must select another node to satisfy the precedence relation Fig. 2(c). Finally, this process goes on until a feasible partition is found and the min-cut size is 2 Fig. 2(d).

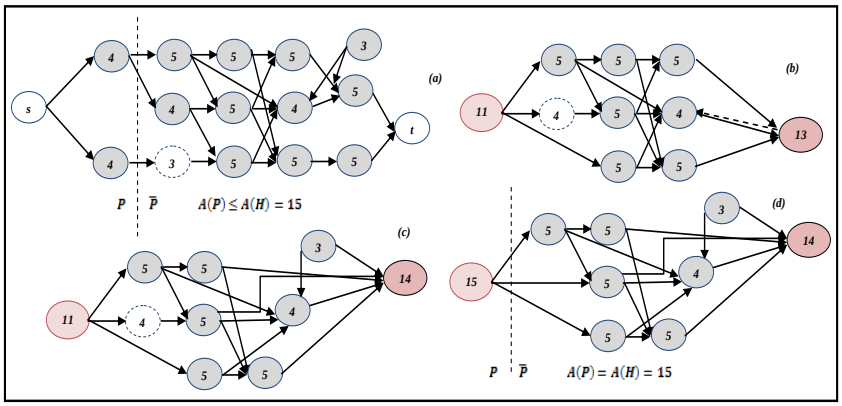

Fig. 2. Using the network flow Feasible partitioning algorithm: (a) iteration1, min-cut size is 3; (b) one out-edge of sink node *t*; (d) iteration 3, min-cut size is 3; a feasible partitioning is obtained and min-cut size is 3.

4. Experiment

Hardware architecture, XUP Virtex-II, on which this approach is to be mapped. The XUP Virtex-II Pro FPGA development system can be used at any virtually level of the engineering curricula, from introductory courses through advanced research projects. In our experiences, we used four approaches, list scheduling [12], initial network flow [13], improved network flow [2] and our proposed approach. In our experiences,

we evaluated the performance of each approach in term of total communication costs. The Fig. 3 shows the color layout descriptor "CLD" is a low-level visual descriptor that can be extracted from images or video frames. The process of the CLD extraction consists of four stages: Image partitioning, selection of a single representative color for each block, DCT transformation and nonlinear quantization and zigzag scanning.

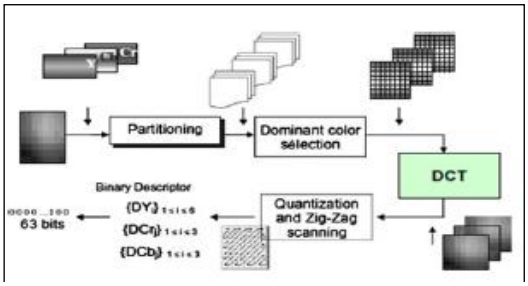

Fig. 3. Block diagram of the CLD extraction.

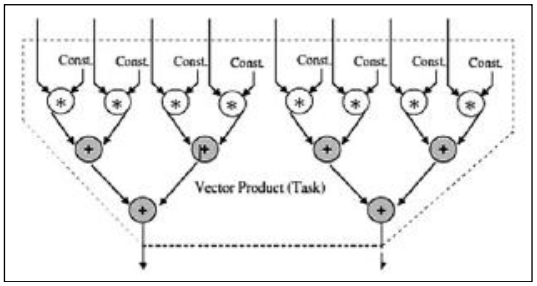

Fig. 4. Vector products.

Since DCT is the most computationally intensive part of the CLD algorithm, it has been chosen to be implemented in hardware, and the rest of subtasks (partitioning, color selection, quantization, zig-zag scanning and Huffman encoding) were chosen for software implementation. The model proposed by [5] is based on 16 vector products. Thus, the entire DCT is a collection of 16 tasks, where each task is a vector product as presented in Fig. 4. There are two kinds of tasks in the task graph. "T1" and "T2", whose structure is similar to vector product, but whose bit widths differ. Table 1 gives the characteristic of 4×4 DCT, 16 × 16 DCT task graphs. The Table 2 gives the different solutions provided by the list scheduling, the initial network flow technique, the enhanced network flow and the proposed approach. Results of Table 2 shows an average improvement of 14.3 %, 6.8 % and 28.1 % in term of total communications costs compared with initial network flow, improved network flow and list scheduling respectively. These results show the magnitude of benefit possible adopting our approach.

Therefore, our algorithm has a good trade-off between latency of the graph and reconfiguration overhead. Hence, our approach can be qualified to be a good temporal partitioning candidate. In fact, an optimal partitioning approach needs to balance computation required for each partition and reduce the reconfiguration overhead so that mapped applications can be executed faster on dynamically reconfigurable hardware.

Table 1. Benchmark Characteristics

DFGs	Nodes	Edges	Area (CLB)
DCT 4×4	224	256	8045
DCT 16×16	1929	2304	13919

5. Conclusion

Today's large and complex designs are now commonly implemented in FPGAs, however designer suffers principally from the time needed due to communication overhead, which is still relatively high. A high reconfiguration time may lead to impractical design mainly when designer focuses on minimizing the overall communication of the design partition. For that reason, we have developed in this paper a typical temporal partitioning algorithm to reduce the communication between design partitions. In fact, our algorithm finds the best schedule of nodes that minimizes the communication between design partitions of the graph. In addition, to show the effectiveness of our algorithm, the algorithm is experimented on benchmark circuits such as DCT task graphs. The studied evaluation cases show that the proposed

algorithm provides very significant results terms of communication cost and latency versus other well-known algorithms used in the temporal partitioning field.

Table 2. Design Results

	Propose algorithm	Initial network flow	improved network flow	list scheduling	Improvement versus initial network flow	Improvement versus improved network flow	Improvement versus list scheduling
Graph	DCT 4×4	DCT 4×4	DCT 4×4	DCT 4×4	DCT 4×4	DCT 4×4	DCT 4×4
TCcost	550	634	589	744	13.24%	6.21%	26.1%
Whole latency	7×CT	9×CT	9 × CT	9×CT	22%	22%	22%
Graph	DCT 16×16	DCT 16×16	DCT 16×16	DCT 16×16	DCT 16×16	DCT 16×16	DCT 16×16
TCcost	2035	2378	2193	3016	14.42%	7.2%	32.52%
Whole latency	11×CT	15×CT	15×CT	15×CT	26%	26%	26%

Acknowledgment

This work was supported by the Technical and Vocational Training Corporation: College of Technology at Alkharj, Saudi Arabia. We are grateful to our Dean Dean Mr Youssef Ahmad AL-JASSER for his motivation.

References

[1] Bobda, C. (2007). *Introduction to Reconfigurable Computing Architectures, Algorithms, and Applications*. Springer Publishers.

[2] Joao, M. P. C. (2003). On combining temporal partitioning and sharing of functional units in compilation for reconfigurable architectures. *IEEE Trans. Computers, 52(10),* 1362-1375.

[3] Jiang Y.-C., *et al*. (2007). Temporal partitioning data flow graphs for dynamically reconfigurable computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 15(12),* 1351-1361.

[4] Mtibaa, A., Ouni, B., & Abid, M. (2007). An efficient list scheduling algorithm for time placement problem. *Computers & Electrical Engineering, 33(4),* 285–298.

[5] Cardoso, J. M. P. (2003). On combining temporal partitioning and sharing of functional units in compilation for reconfigurable architectures. *IEEE Trans. Comput., 52(10),* 1362–1375.

[6] Trimberger, S. (1998). Scheduling designs into a time-multiplexed FPGA. *Proceedings of the ACM International Symposium on Gate Arrays* (pp. 153–160).

[7] Spillane, J., & Owen, H. (1998). Temporal partitioning for partially-reconfigurable field programmable gate. *Proceedings of Parallel and Distributed: Vol. 1388. Lecture Notes in Computer Science* (pp. 37-42).

[8] Ouni, B., Ayadi, R., & Mtibaa, A. (2011). Partitioning and scheduling technique for run time reconfigured systems. *Int. J. Comput. Aided Eng. Technol.*, *3(1)*, 77–91.

[9] Bouraoui, O., Abdellatif, M., & Bourennane, E.-B. (2009). Scheduling approach for run time reconfigured systems. *International Journal of Computer Sciences and Engineering Systems, 4(4),* pp. 36-44.

[10] Liu, H., & Wong, D. F. (1998). Network flow based circuit partitioning for time-multiplexed FPGAs. *Proceedings of Computer-Aided Design* (pp. 497–504).

[11] Liu, H., & Wong, D. F. (1998). Network flow based multi-way partitioning with area and pin constraints. *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems, 17(1),* 50-59.

[12] Bouraoui, O., Ramzi, A., & Abdellatif, M. (2011). Temporal partitioning of data flow graph for dynamiclly reconfigurable architecture. *Journal of System Architecture, 57(8),* 790-798.

[13] Ramzi, A., Bouraoui, O., & Abdellatif, M. (2014). Integrated temporal partitioning and partial

reconfiguration techniques for design latency improvement, Springer, *Evolving Systems, 5(2),* 133-141.

Azibi Abdo received his B.Sc. degree in applied electronics, from Hokkaido Institute of Technology, Japan in April, 2003. He got the master degree of engineering in electrical and electronic from Adelaide University, South Australia, Australia in Dec. 2010. His research interests include high level synthesis, methodologies development for reconfigurable architectures.

Ramzi Ayadi received his M.Sc. degree in electronics and microelectronics from the University of Monastir, Tunisia in 2007. He completed his thesis in 2012 from the National School of Engineering of Monastir, Tunisia. His research interests include high level synthesis, methodologies development for reconfigurable architectures.

Med Lassaad Kaddachi received his PhD degree in 2012 in electronic and also electrical engineering respectively from the University of Monastir, Tunisia, and the University of Henri Poincaré Nancy I, France. His research activity includes QoS management in real time embedded system and multimedia applications. He focuses mainly on the design and performances evaluation of hardware solutions for communication systems with multiple constraints.