Journal of Computer Science 2 (5): 422-430, 2006
ISSN 1549-3636
© 2006 Science Publications

An Iterative Method for Algorithms Implementation on a Limited Dynamically Reconfigurable Hardware

[1]Abdellatif. Mtibaa, [2,3]Abdessalem. Ben Abdelali, [2,3]Lotfi. Boussaid and [2]Elbey. Bourennane
[1]Laboratory EµE, Faculty of Sciences of Monastir, 5019, Monastir, Tunisia
[2]Laboratory LE2I, University of Burgundy, 21000 Dijon, France
[3]Laboratory C.E.S, National Engineering School of Sfax,(ENIS),B.P W. 3038, Sfax, Tunisia

Abstract: In this study we propose a framework and a combined temporal partitioning and design space exploration method for run time reconfigurable processors. Our objective is to help designers to implement an algorithm in limited FPGA area resources while respecting the execution time constraint. The algorithm to be implemented is represented by a task graph with different implementation alternatives (design points) for each task. We study the effect of hardware resources limitation in the choice of the algorithm implementation design point. The proposed method is based on an heuristic technique which consists on combining temporal partitioning and task design points selection to obtain solutions that satisfy the imposed constraints.

Key words: Reconfigurable hardware, run time reconfiguration, time partitioning, design points

INTRODUCTION

With today's deep sub-micron technology, the state-of-the-art FPGA have exceeded 10 million system gates, allowing for multimillion gates FPGAs operating at speeds surpassing 400 MHz. Many designs, which previously could only achieve speed and cost of density goals in ASICs, are converting to much more flexible and productive reprogrammable solutions. The major developments in FPGA logic density, speed, packaging etc. have made implementing a system of processor/s, IP blocks, and user logic in an FPGA (System On a Programmable Chip: SOPC) a possibility. This technology is currently being used for the acceleration of a wide variety of applications on a large number of systems. It has evolved so much that the real-time aspect is not the only objective of the designer[1]. It has allowed the association of the flexibility and the specificity. Several applications can be realized by specialized architectures by simply configuring the FPGAs each time the FPGA-based board is supplied.

With the advent of new device architectures and new software tools, the interest in Run-Time Reconfiguration (RTR) or dynamically reconfiguration logic has increased. This concept has introduced several advantages. It helps the designer to optimize his implementation by increasing the functional density of the FPGA coprocessor. It offers the possibility of sharing in time the available resources in the FPGA between the different tasks of an application. This can be accomplished by using either total or partial dynamic reconfiguration. This later allows the configuration of a part of an FPGA design to change while the circuit is running. The AT40K40 FPGA family of ATMEL allows the reconfiguration of any area of the component by modifying the SRAM configuration contents[1]. Actually, Xilinx offers this technology for his more recent families such as the Virtex-II Pro FPGAs[2-4].

In this study we propose a method for efficient management of a given FPGA area resources for a particular algorithm by exploiting the dynamic reconfiguration for possible use in SOPC. The proposed method consists in combining temporal partitioning techniques and design points selection of the different tasks constituting the algorithm. We aim to resolve the temporal partitioning problem for a given application while considering the characteristic of multi design point of each algorithm task. This can be very useful for possibly variable resources available on the FPGA when adding a new service or an update[5]. By using different alternatives for the algorithm tasks we increase the chance to meet the application constraints. In fact choosing the best design point for each task may not necessarily result in the best overall design. This depends on the architectural constraints and the dependency constraints among the tasks[6].

State of the ART in the dynamic reconfiguration domain: In literature a lot of interest was given to the dynamic reconfiguration and the opportunities given by the new FPGA technologies. Works in this field aim To reduce the difficulty in managing the dynamically reconfigured application and to provide a reliable implementation by developing a set of tools and associated methodologies addressing many issues related to the Dynamic Reconfiguration such as: Automatic partitioning of a conventional design, Specification of the dynamic constraints, Verification

Corresponding Author: Abdellatif. Mtibaa, Laboratory EµE, Faculty of Sciences of Monastir, 5019, Monastir, Tunisia
Tel. No. 00216 98 565026

of the dynamic implementation through dynamic simulations, Automatic generation of the configuration controller, etc.

In[7-9] the FSS (FPGA Support System) environment which is developed at Manchester University is represented. It facilitates the execution of hardware-based tasks on a dynamically reconfigurable FPGA. It supports the placement, execution and removal of blocks on the FPGA. A framework for the Design and Implementation of Dynamically and Partially Reconfigurable Systems "PaDReH" is proposed in[10,11]. It is presented with a design flow including partitioning, scheduling and validation. Papers[12,13] are related to the integrated design system called SPARCS (Synthesis and Partitioning for Adaptive Reconfigurable Computing Systems) which is developed in the ECECS Department at Cincinnati University. It aims to automatically partitioning and synthesizing designs for reconfigurable boards with multiple field-programmable devices (FPGA). The system contains a temporal partitioning tool, a spatial partitioning tool, and a high-level synthesis tool. In[14] a run-time reconfiguration system for FPGA computing resources is proposed; System behaviour and architecture are represented as a problem graph, and an architecture graph, respectively. The Model-Integrated Development Environment for Adaptive Computing (MIDE) project of Vanderbilt University[15] has as goal to develop high-level system design tools for implementing dynamically reconfigurable systems using adaptive computing technology. It is aimed at embedded systems of weapons like missile guiding systems. It uses DSP processors coupled to Virtex Xilinx FPGAs. The Berkeley Reconfigurable Architectures, Software, and Systems (BRASS) project of Berkeley University has proposed SCORE (Stream Computations for Reconfigurable Execution)[16], a computation model based on the organization of reconfigurable systems around the virtualization of three main hardware concepts: paged reconfigurable hardware, page communication through the use of streams, and storage. The Dynamically Reconfigurable Hardware Research at Bournemouth University[17,18] has proposed the DYNASTY tool, which is a generic CAD framework for research in the area of reconfigurable system design techniques and methodologies.

The Specific action « dynamically reconfigurable Architectures » of the CNRS (National Center of Scientific Research in France) Multi-field themes network on SOC provides three main research projects in dynamic reconfiguration: ARDOISE, DART and DNODE[19]. In difference to ARDOISE, DART and DNODE are note based on the use of the dynamically reconfigurable FPGA. ARDOISE propose a dynamically reconfigurable Architecture dedicated to embedded image and signal processing. In[1], an image-processing application, image rotation, that exploits the FPGAs dynamic reconfiguration method is presented. It shows that the choice of an implementation, static or dynamic reconfiguration, depends on the application nature. Paper[20] describes Implementation of JPEG2000 Arithmetic Decoder using Dynamic Reconfiguration. The target architecture is ARDOISE. Works about Dynamic Reconfiguration methods and applications presented in[21-23] are also related to the ARDOISE architecture.

Numerous algorithms for partitioning, scheduling and placement of tasks for reconfigurable computing devices are proposed in the literature. The paper[24] treats in addition to the partitioning, the scheduling of the tasks. The advantage of the approach presented in this study is the capability to model communication between nonadjacent on-chip configurations and multiple levels of logic. Other more recent works were interested in depth to the time placement problem[25-30]. The execution of a task on the reconfigurable device leads to the online placement problem, for which a method based on free rectangles management and heuristics fitting has been proposed in[28] and improved in[30,31]. In[25] the authors modeled the time placement as a three-dimensional problem. A task is represented by a cube in which the X and the Y coordinates represent the width and the height of the task in the given FPGA. The Z coordinates represents the time at which the task will be mapped in the FPGA.

In spite of the variety and the importance of the state of the art, this field remains very active. Many problems remain without efficient solutions or have yet to be solved and the academic community continues the suggestion of new techniques and methods to better exploit the dynamic reconfiguration in different applications and systems. The handled problems are due to new applications needs, environment and technology constraints.

Motivations and problem formulation: The actual systems for multimedia services have demanding applications that can be driven by portability, performance, cost, consumption and flexibility. A key challenge of mobile computing, for example, is that many attributes of the environment vary dynamically[32]. The key issue in the design of portable multimedia systems is to find a good balance between flexibility and high-processing power on one side, and area and energy-efficiency of the implementation on the other side. the rapid evolution of multimedia services and their quality necessitates the use of dedicated electronic systems that assure a big level of flexibility by giving the possibility for updates and new services addition while respecting the application constraints. Dynamically reconfigurable systems, usually based on dynamically reconfigurable FPGA, present a very interesting solution for such problems. In fact, with the development of new height performance FPGA families it becomes possible to achieve high performance in term of latency at relatively low cost in term of used hardware resources. Our research interest is related to SORC (system on reconfigurable chip) conception and Algorithm Architecture Adequation (AAA) for multimedia applications[33]. Resolving problems related to the us of dynamic reconfiguration in SOPC is one important issue in our work .

Critical treatments of a given application are usually executed by hardware modules. By considering the application constraints and the available hardware resources the designers have to choose between dynamic reconfiguration and static configuration (or ASIC implementation). In this study we consider that we have limited reconfigurable hardware resources that should be exploited for the implementation of a given algorithm. The resources limitation can be caused by global consideration related to the system and the application constraints or for the fact that the system already exists and no extensions are possible whereas adding a new service or an update or modification of existing service is required (Fig. 1:).

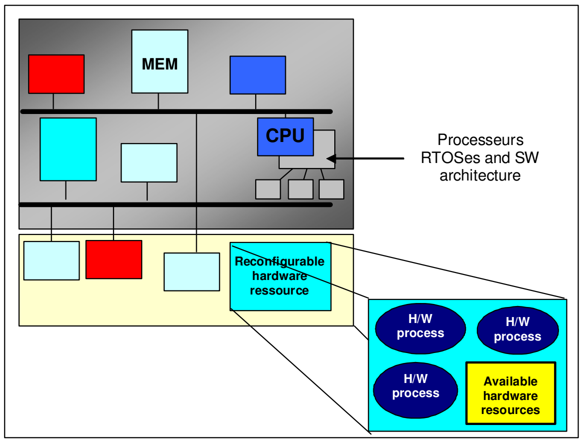

MEM
CPU
Processeurs RTOSes and SW architecture
Reconfigurable hardware ressource
H/W process
H/W process
H/W process
Available hardware resources

Fig. 1: A SOC with run time reconfigurable hardware resources

Our objective is then to be able to run our algorithm in this limited FPGA area resources while respecting the execution time constraint. The main idea consists in considering different implementation alternatives for the tasks set of the algorithm to be implemented and combining the temporal partitioning and tasks design points selection to obtain a solution satisfying the constraints (Fig. 2).

Different implementation alternatives are possible for the same task. They suggest different area-time tradeoff points. These different implementations are design points in the design space of a task. Choosing the best design point for each task may not necessarily result in the best overall design. When combining several design points we can satisfy different application constraints by using an adequate static selection method[5] to choose the convenient combination of tasks.

Exploring a very large design space can be too computationally expensive. To limit the number of candidate design points in our work we consider tasks with average or high granularity. Design space exploration of tasks does not concern the operators level.

The first step in our method is the Static Estimation (SE). This step can help the designer to choose between static and dynamic implementation. It also gives important information that can be used to take decisions in the partitioning algorithm progress. In the following paragraphs we represent the proposed method and the associated tool which integrate the different techniques used in this method and support the required input and output format.

Static estimation: The principal input of the proposed method is a tasks graph with different area and latency for each task. Figure 3 represent two possible methods to enter the task graph and parameters of the different tasks: graphical method and text based method. In this figure the number “2” written near the first task means that the correspondent task has for the moment two design points. Design points can also be introduced by using a text editor. Design points are composed of three parameters: area, time and consumption. In this work we interested only in the two first parameters.

The static estimation is done according to an iterative process based on a technique of unconstrained scheduling. It gives important information about static realization. In a first step the algorithm determines the As Soon As Possible (ASAP) and the As Late As Possible (ALAP) schedules of tasks. In a second step the mobility of each task is calculated. The mobility of a task (T_i) is calculated as follow:

$$\text{Mobility}\ (T_i) = \text{ALAP}\ (T_i) - \text{ASAP}\ (T_i)$$

Between the two scheduling limits ((ASAP) and (ALAP)) other schedules can exist. Their number (NM) is given by:

$$NM = \prod_{i=1}^{Nt} (mobility(T_i) + 1) \quad \text{; Nt: number of tasks}$$

The possible schedules extraction is based on an adequate dependency matrix representing the tasks graph. The algorithm excludes the invalid schedule which does not respect the dependency constraint by reasoning on the dependency matrix values and their location in the matrix.

The proposed framework shows the ASAP and ALAP scheduling results and the different possible schedules (Figure 4). It also allows calculation of the limit values of area (A_{min}, A_{max} : minimum area, maximum area) and latency (L_{min}, L_{max}: minimum latency, maximum latency) when using the best area design points or the best latency design points of each task .

Figure 4 represent an example of a task graph and the scheduling results for the design points presented in

Table 1: Example of design points for a task graph

T1 (ns,Clb)	T2 (ns,Clb)	T3 (ns,Clb)	T4 (ns,Clb)	T5 (ns,Clb)	T6 (ns,Clb)	T7 (ns,Clb)
(840,162)	750,128	860,276	875,174	752,220	820,196	650,185
(560,182)	500,138	700,320	625,235	620,325	560,356	525,235
(420,276)	375,180	480,400	375,336	465,385	435,396	385,325
(375,380)						

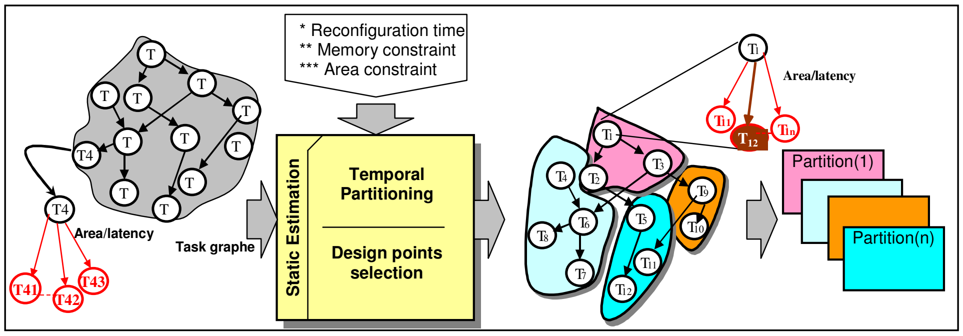
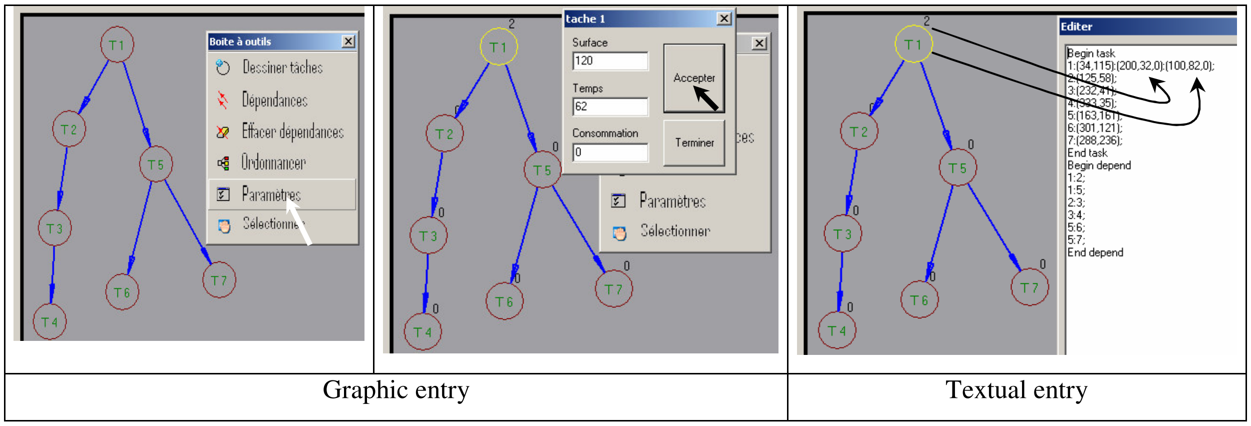


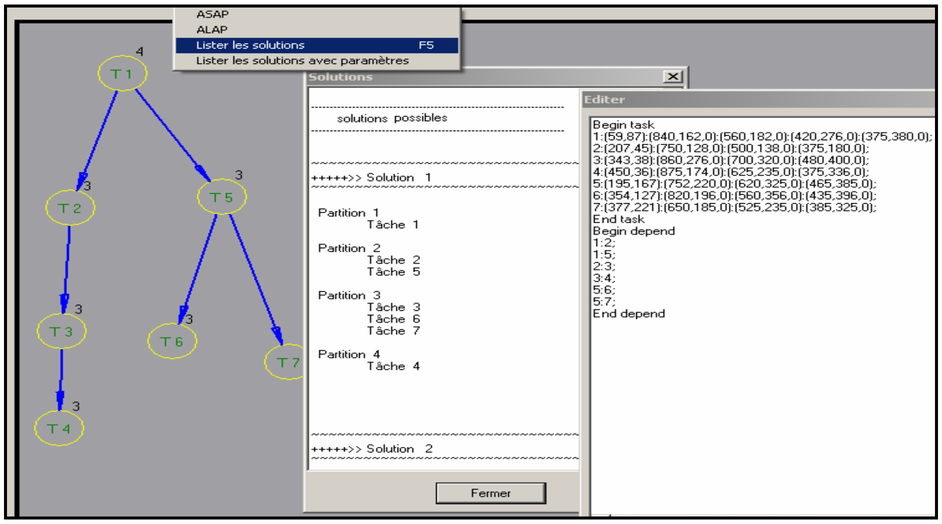
Fig. 2: Principle of the proposed method



Surface 120
Temps 62
Consommation 0

```
Begin task
1:(34,115):(200,32,0):(100,82,0);
2:(125,58);
3:(232,41);
4:(333,35);
5:(163,161);
6:(301,121);
7:(288,236);
End task
Begin depend
1:2;
1:5;
2:3;
3:4;
5:6;
5:7;
End depend
```

Fig. 3: User interface to enter the tasks graph



```
-----------------------------------------------
     solutions  possibles
-----------------------------------------------

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
+++++>> Solution  1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

 Partition  1
          Tâche  1

 Partition  2
          Tâche  2
          Tâche  5

 Partition  3
          Tâche  3
          Tâche  6
          Tâche  7

 Partition  4
          Tâche  4

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
+++++>> Solution  2
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
Begin task
1:(59,87):(840,162,0):(560,182,0):(420,276,0):(375,380,0);
2:(207,45):(750,128,0):(500,138,0):(375,180,0);
3:(343,38):(860,276,0):(700,320,0):(480,400,0);
4:(450,36):(875,174,0):(625,235,0):(375,336,0);
5:(195,167):(752,220,0):(620,325,0):(465,385,0);
6:(354,127):(820,196,0):(560,356,0):(435,396,0);
7:(377,221):(650,185,0):(525,235,0):(385,325,0);
End task
Begin depend
1:2;
1:5;
2:3;
3:4;
5:6;
5:7;
End depend
```

Fig. 4: A screen shot of the scheduling results for the example represented by Table 1

Table 1. The mobility of each task of the example is given by Table 2. The final scheduling result is represented by Table 3.

Table 2: Mobility of tasks for the example represented by Fig. 4

Task	T(1)	T(2)	T(3)	T(4)	T(5)	T(6)	T(7)
Mobility	0	0	0	0	1	1	1

The proposed partitioning method: The proposed method consists in modifying the partitioning algorithm presented in[34] to take in to account the different design point of tasks constituting the algorithm to be implemented. The input of the new partitioning algorithm is tasks graph with different area and latency for each task, area constraint, memory constraint and the reconfiguration time. The output of the algorithm is a set of time partitions. The tasks of each partition are executed when the partition is mapped on the reconfigurable device. We aim to obtain the optimal number of temporal segments and to place each task in the appropriate partition while satisfying area, memory and time constraints. The latency reduction is performed by selecting the appropriate design points in each algorithm iteration. The different steps of the proposed method are summarized by Fig. 5.

To obtain the optimal number of partitions the algorithm preserves the minimal number of partitions for which the area and memory constraints are satisfied (N_{min}) And verify the possibility to obtain a solution that satisfies the time constraint. To determine N_{min} the algorithm starts by calculating the theoretical minimal number of time partitions Nth_{min}, which is given by :

$$Nth_{min} = ceil\left(\left(\sum_{i=1}^{Nt}(A(T_i))_{min}\right)/A_c\right)$$

Nt :	number of tasks in the task graph
T_i :	task number I
A (T_i):	area of task T_i
(A (T_i))$_{min}$:	the minimum area of task T_i among the different design points of T_i
Ac:	area constraint (FPGA area)

ceil is a C++ function. Ceil (x) returns a value representing the smallest integer that is greater than or equal to x

If area and memory constraints are satisfied for at least one possible scheduling, N_{min} is equal to Nth_{min}, if not we increment the number of partitions.

The partitioning process for a specified number of partition (N) gives as a result the possible schedules that respect the memory and area constraints. The constraints test allows identification of the best solution that respect the time constraint. If no solution satisfies the time constraint a design points selection is performed to improve the latency of each partition. The constraints test will be applied for the obtained solutions. If, also, no satisfactory solution is obtained N will be incremented and we repeat the different steps

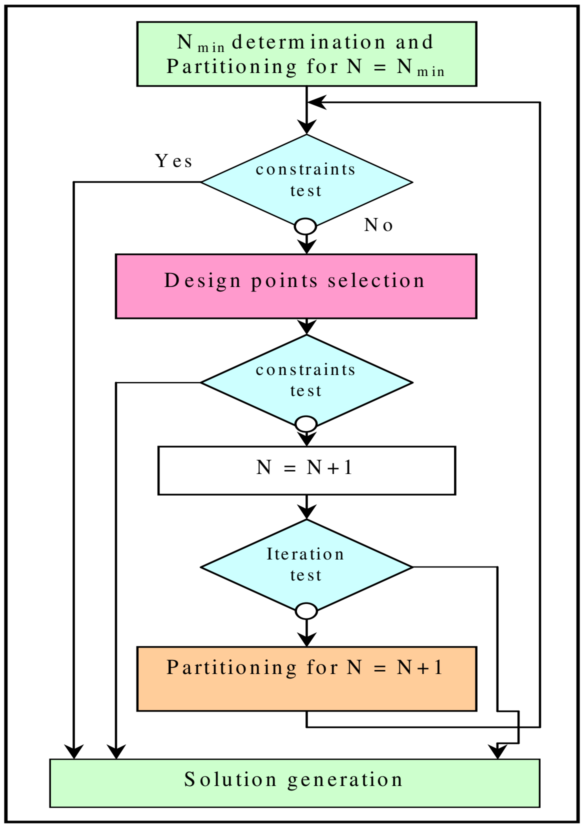


Fig. 5: Flow chart of the proposed method

for N = N+1. Before repeating the described steps a test called "iteration test" is done. Its objective is to verify if the actual number of partitioning (N) allows a feasible and improved solution to be obtained. As a simple condition to verify the feasibility is that N must be smaller than N_{tmax} which represent the theoretic maximal number of partitions. N_{tmax} is obtained by:

$$N_{t\max} = floor(C_t / R_T)$$

C_t:	time constraint
R_T:	reconfiguration time

Floor is a C++ function. Floor(x) returns a value representing the largest integer that is less than or equal to x

N_{tmax} represents a very coarse criterion. In our algorithm we consider a better criterion definition based on the time left for the execution of the different tasks for a given number of partitions (N). This execution time is defined as:

$$E_T = C_t - R_T \times N$$

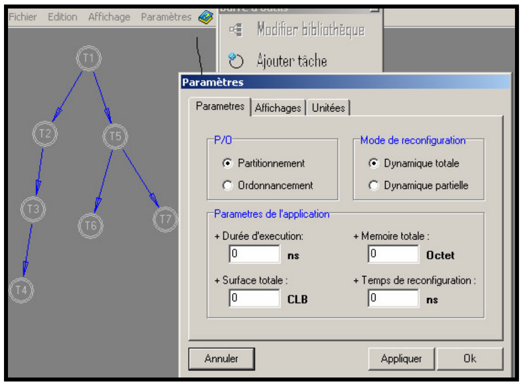
When the number of partitions increases using design points with a higher area becomes possible and a higher speed can be obtained, but the available execution time, E_T, decreases. Before studying solutions for the actual number of partitions we have to verify if E_T is sufficient to execute tasks. In this step the Static Estimation results are useful.

Table 3: Scheduling results for the example represented by Fig. 4 (S(i) : scheduling solution N°i; Ctr (i): control step N°i)

	Ctr 1	Ctr 2	Ctr 3	Ctr 4	Amin	L max	Amax	L min
S 1	1	2-5	3-6-7	4	1341	3327	2402	1695
S 2	1	2-5	3-7	4-6	1341	3327	2402	1755
S 3	1	2	3-5-6	4-7	1341	3325	2402	1615
S 4	1	2	3-5	4-6-7	1341	3325	2402	1665

Partitioning process: In the partitioning process for a given number of partitions (N) we always start with the minimum area design point for each task and we identify the possible schedules that respect the memory and area constraints. The obtained schedules must respect the dependency constraint: a task T_i on which another task T_j is dependent has to be placed either in the same partition as T_j or in an earlier one.

The reconfiguration time and the area, memory and time constraints are introduced by using the graphical interface represented by Fig. 6.



P/O
[✓ Partitionnement]
Ordonnancement

Mode de reconfiguration
[✓ Dynamique totale]
Dynamique partielle

Parametres de l'application
+ Durée d'execution: 0 ns
+ Memoire totale : 0 Octet
+ Surface totale : 0 CLB
+ Temps de reconfiguration : 0 ns

Fig. 6: Graphical interface to enter the user constraints

To respect the area constraint the sum of area costs of all the tasks mapped to a temporal partition must be less than the area constraint (Ac):

$$A\ (P_i) \leq Ac$$

A (P_i): area of partition P_i. It is calculated as following:

$$A(P_i) = \sum_j \Phi_{i,j} . A(T_j)$$

with: $$\begin{cases} \Phi_{i,j} = 1 & \text{if } T_j \in P_i \\ \Phi_{i,j} = 0 & \text{if } T_j \notin P_i \end{cases}$$

Data transfer between partitions take place due to dependent tasks belonging to different partitions (figure 7). This intermediate data needs to be stored between partitions and should be less than the memory constraint (Mc). For a given partition P_i we have to take into account only the available memory resources (Ma) which is calculated by: Ma = Mc - Mu

(M_u) represents memory resource used by previous partitions of P_i to communicate with subsequent partitions of P_i. Data used for this communication must still be saved until the execution of the concerned partitions. In Figure 7 the DATA (D2) will be used by the partition P_3. For the memory constraint of partition P_2 we have to subtract the necessary memory resources from D2.

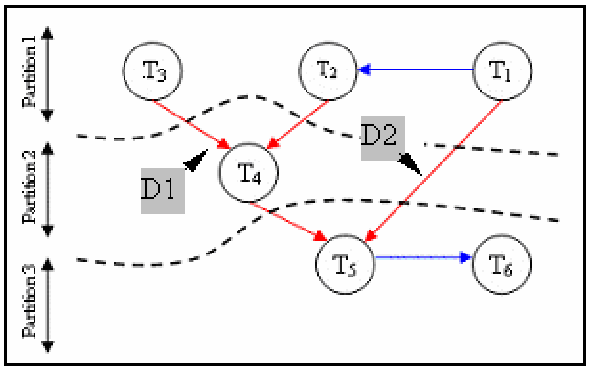


Fig. 7: Data transferring between partitions

The Partitioning problem resolution starts by extracting a dependency vector (DV) from the task graph as defined in[34]. In a second step, the algorithm builds the dependency matrix D. The matrix D has as dimension N(lines) x Nt(columns); with N represent the number of partition and Nt represent the number of tasks. From this matrix, the algorithm extracts the possible schedules of tasks while respecting the dependency constraints.

Design point selection: To be able to meet the latency constraint the design point's selection has as objective to improve the latency of each partition by exploring the different design points of each task constituting the considered partition. For a given partition the selection problem can be formulated as represented in figure 8. We aim to obtain the optimal latency while respecting the area and memory constraints.
Design points of each task is represented in the following form:

$$\left(A(T_1)^1, L(T_1)^1\right), \left(A(T_1)^2, L(T_1)^2\right), \ldots, \left(A(T_1)^n, L(T_1)^n\right),$$

$A(T_i)^j$:	area of the design point N° j of task i
$L(T_i)^j$:	latency of the design point N° j of task i

Area constraint is calculated by considering the sum of tasks area in the considered partition. The latency of a given solution depends on the execution order of tasks (sequential and parallel) and their dependency (Fig. 9). It will be the maximal latency among all the paths of the task graph mapped to the partition.

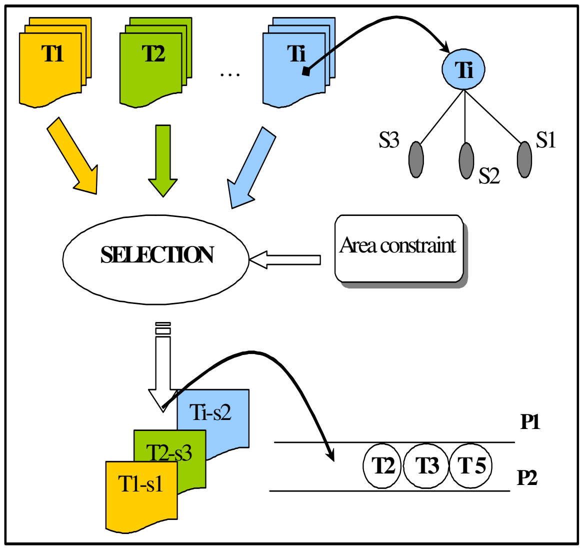


Fig. 8: Design points selection

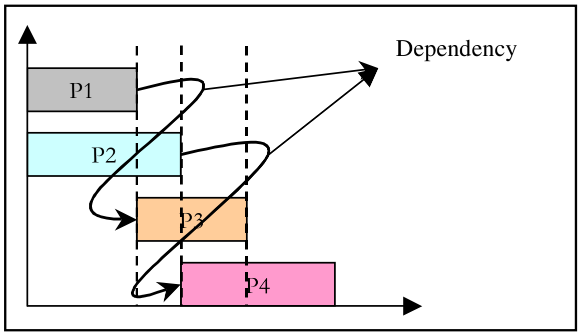


Fig. 9: An example of tasks dependency

The latency of partition P_i ($L(P_i)$) is given by:

$$Lat(P_i) = \underset{j}{Max}\{\Phi_{i,j} . LP(T_j)\}$$

with $LP(T_j) = \sum_{k \neq j} \Phi_{i,k} . \Phi_{j,k} . L(T_k) + L(T_j)$

Where

$\varnothing_{ij} = \varnothing_{ik} = 1$, if the task ($T_j$) respectively the task (T_k) belongs to the partition (P_i), else $\varnothing_{ij} = \varnothing_{ik} = 0$

$\varnothing_{jk} = 1$, if the task (T_k) depends on the task (T_j), else $\varnothing_{jk} = 0$.

For the example represented by figure 9, the latency is calculated as follows:

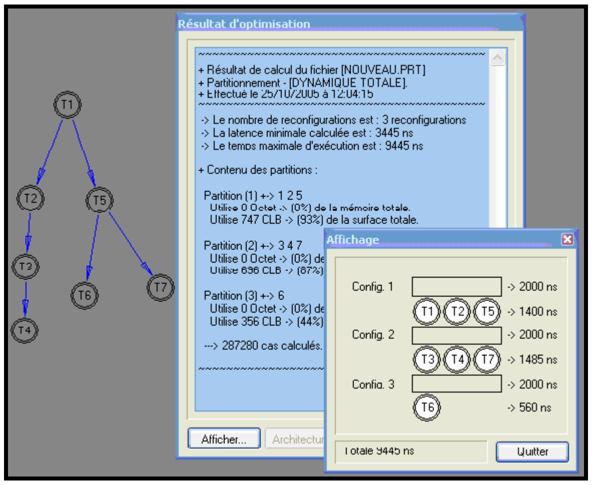
$$L(P_i) = Max[(L(P_1) + L(P_3)), (L(P_2) + L(P_4))]$$

We use the same principle to calculate the global latency for a given solution ($L(S_n)$) :

$$L(S_n) = \sum_{i=1}^{N} L(P_i)$$

THE PARTITION RESULT REPRESENTATION

The final result is a set of partition of tasks. Each task takes an appropriate design point so that the combinations of the different tasks according to the partition order correspond to a satisfying solution. The final result can be represented graphically as shown in Fig. 10.



```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
+ Résultat de calcul du fichier [NOUVEAU.PRT]
+ Partitionnement - [DYNAMIQUE TOTALE].
+ Effectué le 25/10/2005 à 12:04:15
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 -> Le nombre de reconfigurations est : 3 reconfigurations
 -> La latence minimale calculée est : 3445 ns
 -> Le temps maximale d'exécution est : 9445 ns

+ Contenu des partitions :

 Partition (1) +-> 1 2 5
   Utilise 0 Octet -> (0%) de la mémoire totale.
   Utilise 747 CLB -> (93%) de la surface totale.

 Partition (2) +-> 3 4 7
   Utilise 0 Octet -> (0%) de
   Utilise 696 CLB -> (87%)

 Partition (3) +-> 6
   Utilise 0 Octet -> (0%) de
   Utilise 356 CLB -> (44%)

 ---> 287280 cas calculés.

~~~~~~~~~~~~~~~~~~~~~~
```

```
Config. 1               -> 2000 ns
          T1  T2  T5    -> 1400 ns
Config. 2               -> 2000 ns
          T3  T4  T7    -> 1485 ns
Config. 3               -> 2000 ns
          T6            -> 560 ns

Totale 9445 ns
```

Fig. 10: Graphical representation of the final solution

CONCLUSION

In this study we have described a method and a framework that allows facilitating the implementation of a given algorithm in a limited run time reconfigurable FPGA area. The proposed method consists on combining temporal partitioning and a selection of task design points to obtain constraint satisfying solutions. This is based on the fact that choosing the best design point for each task may not necessarily result in the best overall design **as** this depends on the architectural constraints and the dependency constraints among the tasks. A dedicated framework was designed to implement the method with graphical interface to be more helpful for the user.

Our work currently concerns different aspects that aim to improve the effectiveness of the results and to concretize the use of DR and the proposed methods in several applications. In our work group we are interested in the multimedia field represented mainly by content based video indexing applications that can be a very important application domain for the RD. To improve the obtained results of the proposed methods for the DR it is necessary to take into account the internal structure of the new FPGA devices in terms of internal memory and resources and their dispositions. This depends on the FPGA family and characteristics that will be used as a supplementary input for our methods. It will represent additional conditions to be taken into account for result reliability improvement.

REFERENCES

1. Bourennane, E., C. Milan, M. Paindavoine and S. Bouchoux, 2002. Real time image rotation using dynamic reconfiguration. Real-Time Imaging J., 8: 277-289.
2. Butel, P., G. Habay and A. Rachet, 2004. Managing partial dynamic reconfiguration in Virtex-II Pro FPGAs. Xcell J., 50 : 32-37.
3. Xilinx, 2004. Two flows for partial reconfiguration: module based or difference based. Application. Note, Xilinx, Sept.
4. Bobda, C., B. Blodget, M. Huebner, A. Niyonkuru, A. Ahmadinia and M. Majer, 2004. Designing partial and dynamically reconfigurable applications on Xilinx Virtex-II FPGAs using HandelC. Technical Report 03-2004, University of Erlangen-Nuremberg, Department of CS 12.
5. Albouchi, A., A. Mtibaa, 2004. Component Selection for SOC. 16th Intl. Conf. Microelectronics: ICM'04, Dec. 6-8, Gammart, Tunisia, pp : 750-753.
6. Kaul, M. and R. Vemuri, 1999. Temporal partitioning combined with design space exploration for latency minimization of run-time reconfigured designs. design automation & test in Europe (DATE'99) Conf., p. 202, Munich, Germany, 9-12 Mar.
7. Edwards, M. and P. Green, 2003. Runtime support for dynamically reconfigurable computing systems. J. Systems Architecture, 49: 267–281.
8. Bubb, L., M. Edwards, P. Green, C. Pimlott, K. Rees, M. Stewart, A. Taylor, M. Vakondios and J. Yates, 2001. Run-time support environment for reconfigurable systems. Euromicro Symp. Digital Systems, Design (DSD2001), pp: 35-141, Warsaw, Poland, Sep. 04-06.
9. Green, P., M. Vakondios and M. Edwards, 2002. An evaluation of an FPGA run-time support system. 28th euromicro symposium on digital system design (DSD'02), pp: 299-307, Dortmund, Germany, Sep. 04-06.
10. Carvalho, E., N. Calazans, E. Brião and F. Moraes, 2004. PaDReH a framework for the design and implementation of dynamically and partially reconfigurable systems. Proc. SBCCI.
11. Carvalho, E., F. Möller, F. Moraes, N. Calazans, 2004. Design frameworks and configuration controllers for dynamic and partial reconfiguration. Technical Report Series, No. 042, Faculté d'informatique –Bresil, June.
12. Kaul, M., V. Srinivasan, S. Govindarajan, I. Ouaiss and R. Vemuri, 1998. Partitioning and synthesis for run-time reconfigurable computers using the SPARCS system. Military and Aerospace Applications of Programmable Devices and Technologies Conf. (MAPLD'98), NASA Goddard Space Flight Center Greenbelt, Maryland, Sep. 15-16.
13. Govindarajan, S. and R. Vemuri, 2000. Tightly integrated design space exploration with spatial and temporal partitioning in SPARCS. Proc. Roadmap to Reconfigurable Computing. 10th Intl. Workshop on Field-Programmable Logic and Applications (FPL 2000), Villach, Austria, Aug. 27-30.
14. Eisenring, M. and M. Platzner, 2002. A framework for run-time reconfigurable systems. J. Supercomputing, 21: 145-159.
15. Bapty, T., S. Neema, J. Scott, J. Sztipanovits and S. Asaad, 2000. Model-integrated tools for the design of dynamically reconfigurable systems. Technical Report #ISIS-99-01, ISIS, Vanderbilt University.
16. Caspi, E., A. Dehon and J. Wawrzynek, 2001. A streaming multithreaded model. Proc. 3rd Workshop on Media and Stream Processors, pp: 21–28, Austin, TX.
17. Vasilko, M., 2000. Design visualisation for dynamically reconfigurable systems. Proc. Roadmap to Reconfigurable Computing. 10th Intl. Workshop on Field-Programmable Logic and Applications FPL 2000, pp: 131–140, Villach, Austria, Aug. 27-30.
18. Vasilko, M. and D. Long, 1998. Design of a prototyping system for high-speed dynamically reconfigurable logic. Proc. 8th Annual Advanced PLD & FPGA Conference and Exhibition, pp: 208-219, Royal Ascot, Bracknell, England, May 12.
19. Torres, L., 2002. Architectures reconfigurables dynamiquement pour les systemes sur puce. Workshop du Réseau Thématique "System On Chip" du STIC_CNRS, Aussois, 23 Sep.
20. Sophie, B. and E. Bourennane, 2005. An application based on dynamic reconfiguration of FPGAs: JPEG2000 arithmetic decoder. Optical Engineering.
21. Abel, N., L. Kessal and D. Demigny, 2004. Design flexibility using FPGA dynamical reconfiguration. ICIP'2004.
22. Boudouani, N., 2004. Architectures reconfigurables dynamiquement: synthèse matérielle d'opérateurs de détection et d'estimation de mouvement temps réel. PHD thesis, university of Cergy-Pontoise - Mars.

23. Kessal, L., N. Abel et D. Demigny, 2005. Développement des IPs et ordonnancement des configurations dans une Architecture à Reconfiguration Dynamique. Journées Francophones sur l'Adéquation Algorithme Architecture JFAAA'05, Dijon, France, 18 Jan.
24. D. Chang and M. Marek-Sadowska, 1999. Partitioning sequential circuits on dynamically recontigurable FPGAS. IEEE Trans. Computers, 48: 565–578.
25. Bobda, C., 2003. Synthesis of dataflow graphs for reconfigurable systems using temporal partitioning and temporal placement. PhD Thesis. University Paderborn, Heinz Nixdorf Institute.
26. Ahmadinia, A., C. Bobda, D. Koch, M. Majer and J. Teich, 2004. Task scheduling for heterogeneous reconfigurable computers. Proc. 17th Symp. Integrated Circuits and Systems Design (SBCCI), pp: 22-27, Pernambuco, Brazil, Sep. 7-11.
27. Handa, M. and R. Vemuri, 2004. An efficient algorithm for finding empty space for online FPGA placement. Proc. 41st Ann. Conference on Design Automation, pp: 960–965, ACM Press.
28. Bazargan, K., R. Kastner and M. Sarrafzadeh, 2000. Fast template placement for reconfigurable computing systems. IEEE Design and Test-Special Issue on Reconfigurable Computing, pp: 68–83.
29. Steiger, C., H. Walder and M. Platzner, 2003. Heuristics for online scheduling real-time tasks to partially reconfigurable devices. Proc. 3rd Intl. Conf. Field Programmable Logic and Application (FPL'03), pp: 575–584.
30. Walder, H., C. Steiger and M. Platzner, 2003. Fast online task placement on FPGAs: Free space partitioning and 2d-hashing. Proc. 17th Intl. Parallel and Distributed Processing Symp. (IPDPS)/Reconfigurable Architectures Workshop (RAW), pp: 178.
31. Ahmadinia, A. and J. Teich, 2003. Speeding up online placement for XILINX FPGAs by reducing configuration overhead. Proc. IFIP Intl. Conf. VLSI-SOC, pp: 118–122, Darmstadt, Germany.
32. Gerard, J.M.S., J.M.P. Havinga, L.T. Smit, P.M. Heysters and M.A.J. Rosien, 2002,. Dynamic reconfiguration in mobile systems. In Field-Programmable Logic and Applications, pp: 171-181.
33. Ben Abdelali, A. and A. Mtibaa, 2005. Toward hardware implementation of the compact color descriptor. Adv. Engg. Software, 36: 475-486.
34. Ouni, B., A. Mtibaa, M. Abid, 2005. Synthesis and time partitioning for reconfigurable systems. Design Automation for Embedded Systems J., 9: 177-191.