PLACEMENT OF HARDWARE TASKS ON FPGA USING THE BEES ALGORITHM

Bassem Ouni, Ikbel Belaid, Fabrice Muller
University of Nice Sophia-Antipolis, LEAT CNRS, 250 rue Albert Einstein, bat 4, 06560 Valbonne, France

Maher Benjemaa
University of Sfax, National School of Engineers of Sfax, Research Unit ReDCAD, BP 1173-3038 Sfax, Tunisia

Keywords: Reconfigurable zones, Hardware tasks, FPGA, Placement and mapping, Optimization problem, Bees algorithm.

Abstract: The dynamic and partial reconfiguration in FPGA with heterogeneous resources is a challenge for the next years. It allows reconfiguring a specific hardware zone in FPGA while maintaining the activity of the remaining circuit's part. This paper introduces a new approach about how to solve the problem of placement of the hardware tasks on the recent reconfigurable technology using the honey Bees Algorithm. This approach aims at performing a good placement by maximizing the efficiency of the used resources and reducing the task's reconfiguration overheads. Experimental results show that the proposed method can perform a good placement of hardware tasks on the device by optimizing significantly the parameters of the cost function in terms of resources and execution time.

1 INTRODUCTION

The reconfigurable hardware technologies are characterized by many heterogeneous resources and multitasking. Hence, managing the hardware tasks and resources is strongly required. The placement of hardware tasks on FPGA consists of two steps: the first step is the partitioning, which manages the empty space in the technology and identifies the potential sites enabling execution of hardware tasks. The second step is the fitting, which selects the feasible placement solution.

In (Bazargan et al., 2000), two methods of placement are introduced: the first one called Keeping All Maximal Empty Rectangles (KAMER) searches all the Maximal Empty Rectangles (MER) after placing each task. Bazargan defines the MERs as the empty rectangles which are not contained in another empty rectangle and which are not necessarily disjoined. The second method denoted Keeping Non-overlapping Empty Rectangles keeps all the non-overlapping holes and is evoked after each split or merge operation.The work in (Ahmadinia et al., 2004) introduces a new method of on-line placement by managing the occupied space instead of free space due to the big size of the empty rectangles and the hardness of managing the free space.

The authors in (Marconi et al., 2008) extend the Bazargan's placement using an Intelligent Merging (IM) algorithm. IM combines three techniques of managing free resources: Merging only if Needed, Partially Merging and Direct Combine. IM accelerates Bazargan's method by three times and improves the placement quality by increasing the rate of accepted tasks.

A new approach of placement is introduced in (Handa and Vemuri, 2004) : the method of staircase. It handles the free space during the first step of the online placement. It tries to improve the KAMER method especially for tasks rejection.

Dissimilar to the other methods, we proposed in recent works an approach which takes into account the hardware tasks heterogeneity and it is applicable in the high complex applications. This method of off-line placement classifies the hardware tasks into many classes based on their resources; these classes are called *RZ* types (Reconfigurable Zones). The physical representations of the *RZ*s are *RPB*s (Reconfigurable Physical Blocs). These *RPB*s are 2D rectangular blocs representing the possible physical locations

Ouni B., Belaid I., Muller F. and Benjemaa M..
PLACEMENT OF HARDWARE TASKS ON FPGA USING THE BEES ALGORITHM.
DOI: 10.5220/0003361804980505
In *Proceedings of the 1st International Conference on Pervasive and Embedded Computing and Communication Systems* (PECCS-2011), pages 498-505
ISBN: 978-989-8425-48-5

of *RZ*s in the device. As depicted by Figure 1, *RZ*s and *RPB*s are modeled by blocs of heterogeneous resources called Reconfigurable Blocs (*RB*) to obtain models called *RB-models*.

Under the French research program FOSFOR project [1] (FOSFOR, 2010), we aim at seeking a good solution of the placement problem using the Bees Algorithm. This good solution tries to enhance the resources efficiency and the execution time and it is not necessary that it ensures the minimum of the cost function. The paper is structured as follows: in section 2 an overview of the problem of hardware tasks' placement on the device is presented. Inside section 3, the Bees Algorithm is introduced. Section 4 deals with our proposed method to solve this problem based on the Bees Algorithm. Then, section 5 shows the experimental results and finally in section 6 conclusions and future works are drawn.

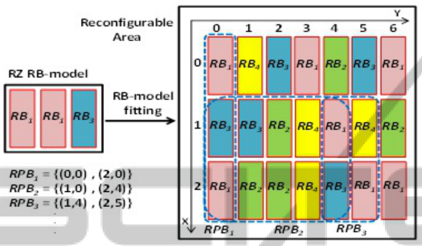

Figure 1: Example of *RPB*s of *RZ*.

2 THE PLACEMENT PROBLEM

The placement problem consists in placing the RZs on the possible RPBs and mapping the tasks on these RZs. This problem is a combinatory optimization problem under constraints. It is characterized by an explosive space of admissible solutions. The placement/mapping problem is defined by the couple (S, F), where S represents the set of the admissible solutions and F($S \Rightarrow R$, R is a set of reals) depicts the objective function of minimization. The resolution of the problem consists in searching the solution s^* included in S where $f(s^*) \leq f(s)$ for each s in S. As all the problems of optimization under constraints, the problem of placement/mapping is defined by the quadruplet (X,D,C,F), where $X = \{X1,X2\}$ and $D = \{D1,D2\}$. $X1$ contains the first set of variables which consists of *RPB* coordinates, their widths and their heights, $X2$ contains the occupation rates of each task on the placed *RZ*s, $D1$ and $D2$ represent respectively the finite domains of possible values of variables of $X1$ and $X2$. Hence, a potential solution for the problem consists in assigning each variable from $X1$ and $X2$ to a value from $D1$ and $D2$. C is a set of constraints which checks whether the combination of values is compatible with the variables. F is the minimization objective function which expresses the optimization criteria and enables the research of the optimal solution from the admissible ones. Consequently, we have associated to the placement/mapping problem the following non-linear mathematical model.

[1]FOSFOR (Flexible Operating System For Reconfigurable platform) is French program aiming at designing a real time operating system distributed on hardware and software execution units which offers required flexibility to application tasks through the run-time reconfiguration and homogeneous HW/SW OS services.

2.1 Constants

The constants are:
NT: number of tasks ; NZ: number of RZs.
NPB: number of RPBs ; NP: number of types of RBs.
$Device\_width$: width of the device.
$Device\_height$: height of the device.
$Device\_RB[lines][columns]$ of RBs: $RB-model$ of the device.
$RZ_j_RB[NP]$of RBs: $RB-model$ of the RZ_j.
$RPB_j_RB[NP]$of RBs: $RB-model$ of the RPB_j.
$RBType[NP]$: the NP types of RBs.
σ_k: cost of the correspondent RB_k.
C_i: the $WCET$ [2] of task T_i ; P_i: the period of task T_i.
$Config_{j,i}$: reconfiguration overhead of a task T_i in RZ_j.
$Threshold_i$: the lower threshold of occupation rate for task T_i in RZs.

2.2 Variables (X)

$X1 = \{(X_{jl},Y_{jl}),(X_{jr},Y_{jr}),W_j,H_j,1 \leq j \leq NZ\}$.
$X2 = \{\delta_{ji}, 1 \leq j \leq NZ, 1 \leq i \leq NT\}$.
Where:

(X_{jl},Y_{jl}): coordinates of the highest left vertex of the RPB for RZ_j.

(X_{jr},Y_{jr}): coordinates of the lowest right vertex of the RPB for RZ_j.

W_j: width of the RPB for RZ_j ; H_j: height of the RPB for RZ_j.

δ_{ji}:occupation rate of RZ_j by T_i.

[2]$WCET$(Worst-case execution time) is the maximum duration the task could take to run on a specific hardware platform.

2.3 Domains (D)

$D1 = \{d_{xl}, d_{yl}, d_{xr}, d_{yr}, dw, dh$ are domains of naturals: $d_{xl} = d_{xr} = dw = [0, Device\_width], d_{yl} = d_{yr} = dh = [0, Device\_height]\}$.

$D2 = \{d_{ji}$ is a domain of naturals, $1 \leq j \leq NZ$, $1 \leq i \leq NT$: $d_{ji} = \{0\}$ or $d_{ji} = \{100\}$ or $d_{ji} = [Threshold_i, 100 - Threshold_i]\}$.

2.4 Constraints (C)

2.4.1 Overlapping Constraint (CP1)

This analytic expression (1) should be respected to avoid the overlapping between the *RPB*s.

$$\forall p,q,p \neq q, et \forall RZ_p, RZ_q, X_{pl} + W_p < X_{ql} or$$
$$X_{ql} + W_q < X_{pl} or Y_{pl} + H_p < Y_{ql} or Y_{ql} + H_q < Y_{pl} \quad (1)$$

2.4.2 Heterogeneity Constraint (CP2)

This constraint (2) checks, before the placement of *RZ* on *RPB*, the existence of the type and the number of *RB*s required by *RZ* in this *RPB*.

$$\forall RZ_j, \sum_{\substack{Y_{jl}<n<Y_{jr} \\ X_{jl}<m<X_{jr}}} \sum_{device[m][n]=RBType[k]} 1 \quad (2)$$

2.4.3 Total Execution for Tasks Constraint (CM1)

This expression (3) verifies whether the totality of the task is well executed.

$$\forall T_i, \sum_{1 \leq j \leq NZ} \delta_{j,i} \leq 100\% \quad (3)$$

2.4.4 Overload of *RZ*s Constraint (CM2)

The following constraint (4) evades the *RZ* overload.

$$\forall RZ_j, \sum_{i \in \{Tasks\_idwithinRZ_j\}} \delta_{j,i} \times C_i/P_i$$
$$+ Config_{j,i}/P_i \leq Load\_Scheduling \quad (4)$$

Where $Load\_Scheduling$ represents the percentage which should not be exceeded to avoid RZ overload. It depends on the scheduling policy. In our case, $Load\_Scheduling = 100\%$ because the tasks are independent and the scheduling policy is EDF.

2.5 Minimization Objective Function

(F) is the cost function of the placement and mapping problem. We target to minimize this function (5).

$$F(X) = Coeffg1 \times EFF(X) +$$
$$Coeffg2 \times MAP(X) \quad (5)$$

EFF. This parameter evaluates the resources efficiency of the current placement of *RZ*s on the appropriate *RPB*s.

MAP. This parameter evaluates the optimality of the mapping of tasks on the *RZ*s. If the mapping of a task in *RZ* is impossible, the value of *MAP* returns -1. The two previous parameters are weighted in the objective function *F* by the coefficients *Coeffg1* and *Coeffg2*. Let's consider RPB_i the *RPB* where the RZ_i is placed within the current solution X. The following equations (6) and (7) represent these parameters.

$$EFF(X) = \sum_{1 \leq j \leq NZ} \sum_{1 \leq k \leq NP} \sigma_k$$
$$\times (RPB_j\_RB[k] - RZ_j\_RB[k]) \quad (6)$$

$$MAP(X) = \sum_{1 \leq m \leq 3} Coeffp_m \times MAP_m(X) \quad (7)$$

Where $Coeffp_m$ represents the coefficient of each parameter MAP_m.
Lets consider $Load\_RZ_i$ the load of RZ_i for each task placed in this *RZ*, it is depicted by this equation (8).

$$Load\_RZ_i = \sum_{p \in \{Tasks\_idwithinRZ_i\}} \delta_{i,p}$$
$$\times C_p/P_p + Config_{i,p}/P_p \quad (8)$$

As showed in the following expression (9), the first expression of MAP_1 evaluates whether the *RZ* is fully exploited. The second one checks whether the loads of placed *RZ*s are balanced, it means checking whether the loads of all *RZ*s are near to the needed average: $Average\_Load$.

$$MAP_1(X) = \sum_{1 \leq j \leq NZ} (100\% - Load\_RZ_j) +$$
$$((Load\_RZ_j - Average\_Load)^2)/NZ \quad (9)$$

Where $Average\_Load = \sum_{1 \leq j \leq NZ} Load\_RZ_j/NZ$
MAP_2 computes the sum of the reconfiguration times. It evaluates whether the tasks are much split on the *RZ*s in order to avoid overheads (reconfiguration and context switch). The reconfiguration overhead depends on *RZ*s and tasks. Thus, MAP_2 favorites mapping tasks with high occupation rates in the *RZ*s assuring the best efficiency of resources. MAP_2 is written

as follows (10):

$$MAP_2(X) = \sum_{1\le i\le NZ}\sum_{\substack{1\le p\le NT\\ \delta_{i,p}\neq 0}} Config_{i,p}\times npr_{i,p} \qquad (10)$$

Where npr_{ip} represents the maximal number of preemptions of the task T_p mapped on RZ_i.
MAP_2 improves the execution time of each task and the quality of services (QoS). It is also computed with the following assumption: the task T_i that is mapped in RZ_j could be preempted only if $\delta_{j,i}\%$ of this task is executed.
MAP_3 calculates the cost of the used *RZ*s during the placement of all the tasks on the device. The costs of the resources are determined taking into account the information in (Xilinx, 2009). The expression (11) depicts MAP_3.

$$MAP_3(X) = \sum_{1\le k\le NP}\sum_{\substack{load\_RZ_i\neq 0\\ 1\le i\le NZ}} \sigma_k \times Z_{i,k} \qquad (11)$$

Where $Z_{i,k}$ is the number of RB_k in RZ_i.
In recent works, we demonstrated that the placement/mapping problem is NP hard so that we will solve it using a heuristic method: the Bees algorithm.

3 THE BEES ALGORITHM

The Bees algorithm is inspired by the food foraging behavior of the honey bees (Pham et al., 2006; Pham et al., 2007b) and could be regarded as an intelligent optimization tool. This algorithm is used to solve the multi-objective problems and it is able to locate good solutions efficiently (Pham and Ghanbarzadeh, 2007; Pham et al., 2007a). The following code is the pseudo code of the Bees Algorithm:

```
1. Initialize the population with random solutions (Placing (n) bees randomly in the search space).
2. Evaluate the fitness of the visited sites (patches of flowers).
While (stopping criterion is not met)
    3. Select $(m)$ best sites having the highest fitness and also choose the (e) elite sites among these best patches.
    4. Recruit a number $(nsp)$ bees for $(m)$ selected sites and a number $(nep)$ bees for (e) elite sites and evaluate fitness $(wherensp < nep)$.
    5. Select the fittest bee from each patch.
    6. Assign the $(n-m)$ remaining bees to search randomly and evaluate their fitness.
End While.
```

We propose to adapt the Bees Algorithm with our context of placement and mapping of the tasks so that we can exploit the effectiveness of this algorithm to solve the placement problem.

4 THE PROPOSED METHOD

The objective of the proposed method is minimizing the cost function to obtain a good placement and mapping of hardware tasks on the device. The entries of this method are the *RZ*s, which are generated from the hardware tasks, and the *RPB*s. The outputs are hardware tasks placed on the technology while optimizing the various parameters of the objective function. In other words, the solution of the placement/mapping problem is the hardware tasks mapped on *RZ*s which are placed in *RPB*s. Accordingly, we consider a bee as a combination of *RZ*s which already contain the hardware tasks. This combination is depicted by a vector of RZs: RZV. We also consider a patch of flowers as a combination of *RPB*s which is represented by a vector of *RPB*s: $RPBV$. As illustrated by Figure 2, the set $\{P_k, k\in[1,nprm]\}$ represents the preemption points of the task T_i, $T_{i,k}$ is the $k-th$ execution section of this task and $nprm$ is the number of preemptions of T_i.

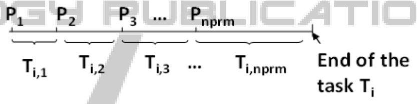

Figure 2: Preemption points and execution sections of T_i.

We repeat these next steps *Nitermapping* times.

4.1 Step 1: Generate Arbitrarily the Occupation Rates Matrix

The first step consists in calculating the matrix P which contains the occupation rates of the tasks in the RZs. Like the random choice of the scout bees from the colony to explore the flowers, this matrix is randomly generated so that the RZVs are chosen arbitrarily. Every line of this matrix should respect the constraint (CM1) and every column should also respect the constraint (CM2). The elements of this matrix $(P = p_{i,j}, 1\le i\le NT, 1\le j\le NZ)$ are vectors of the hardware tasks execution sections. Each execution section of a task is assigned randomly to a possible RZ containing enough resources to execute this task. It is not necessary that the execution sections of a task are assigned to all the possible RZs thus we can find an RZ where the mapping of some tasks is possible but does not contain any execution section. In this case, $p_{i,j}$ = (0). Also, if the mapping of a task T_i in RZ_j is impossible, $p_{i,j}$ = (0). Figure 3 shows the structure of this matrix.

Where k, p and q are naturals in $[1, nprm]$.

The next steps (from 2 to 7) are repeated *Nitr* times.

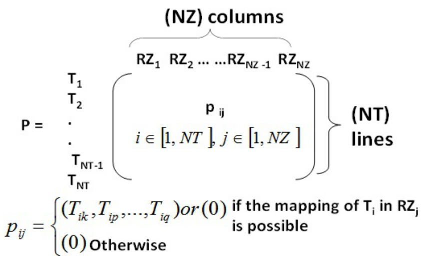
(NZ) columns
RZ1 RZ2 ... ...RZNZ -1 RZNZ
P =
T1 T2 . . TNT-1 TNT
p ij
i ∈ [1, NT], j ∈ [1, NZ]
(NT) lines
pij = (Tik, Tip, ..., Tiq) or (0) if the mapping of Ti in RZj is possible
(0) Otherwise

Figure 3: Occupation rates matrix.

4.2 Step 2: Random Construction of the Matrix of *RZ*s and *RPB*s

In this step, we target to prepare the *n RZV*s and *n RPBV*s. As showed in Figure 4, we generate the matrices *MRZ* and *MRPB* containing respectively *n* different *RZV*s and *n* different *RPBV*s. *MRPB* takes into account the overlapping constraint (CP1): A combination of *RPB*s depicted by one line of *MRPB* must not contain overlapped *RPB*s. The two matrices have the same dimension: *n* lines and *NZ* columns.
In Figure 4, *a*, *b*, *c*, *d* and *e* are naturals in $[1, NZ]$ and *p*, *q*, *r*, *s* and *t* are naturals in $[1, NPB]$.

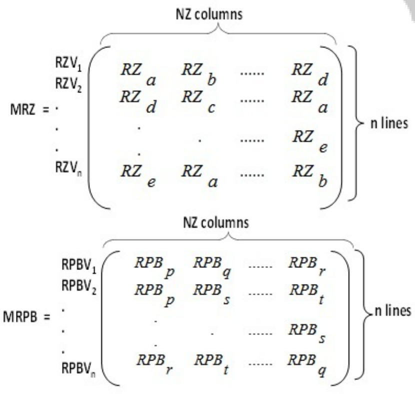

Figure 4: The matrices of the *RZ*s and *RPB*s.

4.3 Step 3: Place Randomly the *RZ*s on the *RPB*s

As depicted in Figure 5, this step, by analogy to the random bees' exploration of the flowers patches, involves placing randomly the *RZV*s on the *RPBV*s. During the placement, we must take into account the possibility of placement by checking that there are enough resources in *RPB* to execute the *RZ* and the fact that an *RPB* can contain only one *RZ*.

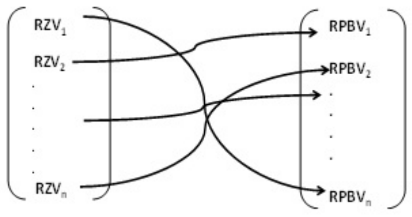

Figure 5: Random placement of the *RZ*s on the *RPB*s.

4.4 Step 4: Sort the RPBVs According to the Cost Function

This step begins by sorting *RPBV*s in an ascending order according to the cost function. Then, we select the *m* best *RPBV*s among which we choose the *e* elite *RPBV*s. Figure 6 illustrates the *RPBV*s sort.

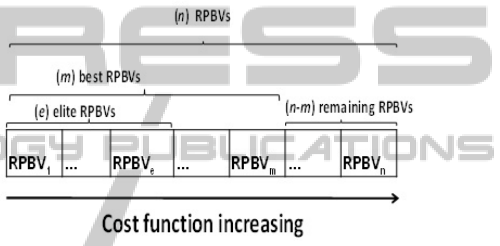

Figure 6: The *RPBV*s sort.

4.5 Step 5: Generate the Matrices of the Elite and Best RPBVs

Like in the colony of bees which assigns a big number *nep* of bees to the *e* elite patches, we place *nep RZV*s on each elite *RPBV* from the *e* elite *RPBV*s and then determine the *RZV* ensuring the minimum cost function for each *RPBV*. Similarly, on each best *RPBV* from the *(m-e)* best *RPBV*s, we place *nsp RZV*s ($nsp < nep$) and we also determine the *RZV* ensuring the minimum cost function for each *RPBV*. Figure 7 shows these placements.

According to these placements, we construct the matrices *EliteFitnessMatrix* and *SelectedFitnessMatrix* which contain respectively the cost functions of the elite and best RPBVs. Figure 8 and Figure 9 indicate these matrices.

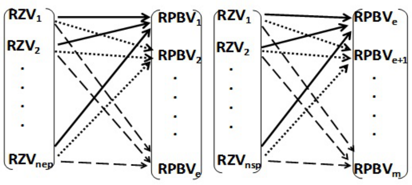

Figure 7: Placement of (nep) RZVs on the (e) elite RPBVs and (nsp) RZVs on the (m-e) best RPBVs.

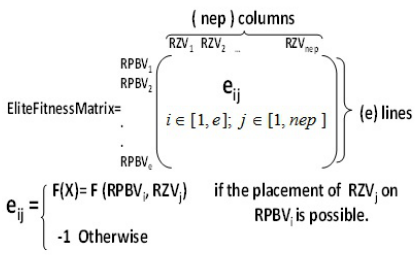
(nep) columns
RZV1 RZV2 ... RZVnep
RPBV1
RPBV2
RPBVe
EliteFitnessMatrix=
eij
i ∈ [1, e]; j ∈ [1, nep]
(e) lines
eij = F(X)= F (RPBVi, RZVj) if the placement of RZVj on RPBVi is possible.
-1 Otherwise

Figure 8: The matrix of the elite *RPBV*s.

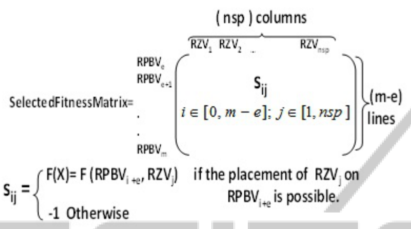
(nsp) columns
RZV1 RZV2 ... RZVnsp
RPBVe
RPBVe+1
RPBVm
SelectedFitnessMatrix=
sij
i ∈ [0, m − e]; j ∈ [1, nsp]
(m-e) lines
sij = F(X)= F (RPBVi+e, RZVj) if the placement of RZVj on RPBVi+e is possible.
-1 Otherwise

Figure 9: The matrix of the best *RPBV*s.

4.6 Step 6: Generate the Matrices of the Remaining RPBVs

By analogy to the Bees Algorithm which assigns the $(n-m)$ remaining bees to search randomly the nectar in the $(n-m)$ remaining patches, we place arbitrarily the $(n-m)$ remaining *RZV*s on the $(n-m)$ remaining *RPBV*s. Then, as showed in Figure 10, we generate the matrix *RemainingFitnessMatrix* which contains the cost functions resulting from these placements.

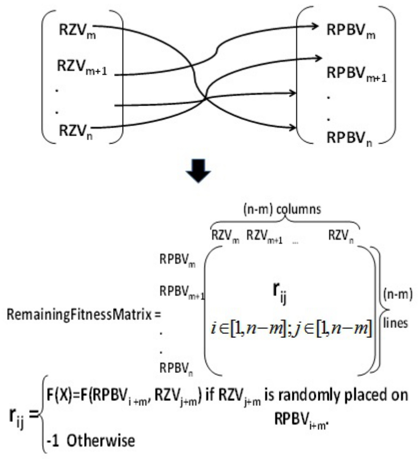

Figure 10: The matrix of the remaining *RPBV*s.

4.7 Step 7: Generate the Matrices of the Elite and Best RPBVs

After each iteration from the *Nitr* ones, we select the couple $(RZV_m, RPBV_m)$ ensuring the minimum of the cost function e_m. The value of e_m is exactly the minimum of the three matrices: *EliteFitnessMatrix*, *SelectedFitnessMatrix* and *RemainingFitnessMatrix*. RZV_m and $RPBV_m$ are respectively the correspondents RZV and RPBV.

We repeat the previous steps (from step 2 to step 7) *Nitr* times and we keep the results of each iteration (the minimum of the cost function e_m and the couple $(RZV_m, RPBV_m)$) in a three dimensional table: *TBEE*. Therefore, *TBEE* contains *Nitr* elements. Then, we choose the minimum of the cost function from this table e_{min} and the correspondent couple $(RZV_{min}, RPBV_{min})$. We also repeat the previous steps (from step 1 to step 7) *Nitrmapping* times and we keep, after each iteration, the results obtained next to the *Nitr* iterations: e_{min} and the correspondent couple $(RZV_{min}, RPBV_{min})$ in a three dimensional table: *TGLOBAL*. Thus, the size of *TGLOBAL* is *Nitrmapping*. Then, we choose the final solution which is the minimum of the cost function in *TGLOBAL*: e_{opt} and the correspondent couple $(RZV_{opt}, RPBV_{opt})$.

In our approach, the tasks with the same type are grouped in RZs. Thus, placing an RZ means placing a set of tasks at once unlike the off-line placement method in (Danne and Stuehmeier, 2005) which handles tasks independently from each other leading to the increase of the tasks' rejection. In addition, if we have a big number of tasks, the complexity of the problem in the proposed method will be small compared to *Danne* and *Stuehmeier* method as the number of RZs to place is fewer than that of tasks. Also, we take into consideration the preemptions of tasks and the overheads which is not the case in (Danne and Stuehmeier, 2005). Besides, we integrate the whole problem in a single heuristic while the method in (Danne and Stuehmeier, 2005) uses three heuristics to validate only its first step of placement.

5 EXPERIMENTAL RESULTS

To illustrate the proposed method, we have implemented an application composed of hardware tasks that are frequently used in the recent embedded systems performing video and audio applications. This application is characterized by hardware tasks of varied sizes and of heterogeneous resources. The hardware tasks are centralized around the microcontroller (T48) which configures these tasks and synchronizes the data flow. During the design of this application, we have synthesized the resources of each hardware task by the Xilinx tool ISE 11.3 and we have determined the configuration overheads of the obtained RZs by performing the partial reconfiguration flow by

Table 1: Hardware tasks features.

Modules	Instances	RBs	WCET (μs)	Period (μs)	Configuration overhead (μs)	Pre-emption points (μs)
MDCT	$\{T_1, T_2\}$	$\{2RB_1, 12RB_2, 3RB_3, 0RB_4\}$	40552	416666	1856	10000, 20000, 30000
AES	$\{T_3\}$	$\{4RB_1, 7RB_2, 1RB_3, 1RB_4\}$	51540	100000	2185	30000, 40000
DDS	$\{T_4, T_5\}$	$\{0RB_1, 1RB_2, 1RB_3, 1RB_4\}$	5000	12000	432	1000, 2000, 4000
T48	$\{T_6\}$	$\{5RB_1, 4RB_2, 0RB_3, 0RB_4\}$	20000	50000	605	5000, 10000, 15000
JPEG	$\{T_7\}$	$\{8RB_1, 12RB_2, 0RB_3, 2RB_4\}$	350000	416666	2421	200000, 300000

means of Xilinx tool Planahead 11.3 and by taking into account I/O routing.

Figure 11 indicates the hardware tasks composing the application and the microcontroller T48 which manages these tasks. The **AES** (Advanced Encryption Standard) task encrypts blocks of 20 Kbytes using a key of 256 bits. The **MDCT** module calculates the Modified Discrete Cosine Transform. **JPEG** task provides a hardware compression of 24 frames per second. The **DDS** (Direct Digital Synthesizer) module generates sinusoidal and programmable waves with adjustable frequency and phase. The characteristics of hardware tasks and their instances are presented in Table 1. These RBs are determined taking into account the reconfiguration granularity and the resources area in Virtex 5 technology (Xilinx, 2009). It contains 4 main types of resources *CLBL*, *CLBM*, *BRAM* and *DSP*. Therefore, RB_1 is 20 *CLBL*s, RB_2 is 20 *CLBM*s, RB_3 is 4 *BRAM*s and RB_4 is 8 *DSP*s. The preemption points are generated arbitrarily and based on the granularity and the *WCET* of the hardware tasks. For all the tasks, we consider that the first preemption point is equal to 0 μs.

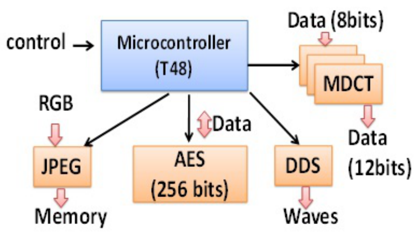

Figure 11: Hardware tasks of the application.

The parameters of the Bees Algorithm chosen to solve this problem are the following: $NiterMapping = 5$, $Nitr = 2$, $n = 6$, $m = 4$, $e = 2$, $nep = 4$ and $nsp = 2$.

Table 2 shows the RZs (Virtex 5 SX50T) and the number of possible *RPBs*. The good solution, resulting from the proposed method, is showed in Table 3. It indicates the different execution sections of the tasks mapped on the *RZ*s which are placed on the *RPB*s. This solution optimizes the parameters of the placement/mapping problem by minimizing the cost function (F). To illustrate the proposed method, we vary the parameter *NiterMapping* which represents the number of repetitions of the Bees Algorithm. We note that the more we repeat the Bees Algorithm, the more the cost function reduces.

Figure 12 depicts the floorplanning of *RPB*s on the technology *Virtex 5*.

The *RPB*s are so limited on the device so that we optimize the exploitation of resources and we avoid the resources waste. Thus, we preserve sufficient space for the static design. This optimization in use of resources minimizes the reconfiguration overhead of FPGA.

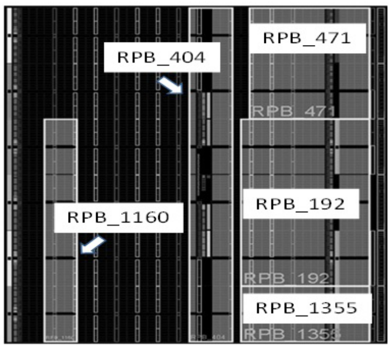

Figure 12: Floorplanning of *RPB*s on *Virtex 5 SX50T*.

Table 2: The number of RZs and possible RPBs.

RZs	RBs	Number of possible RPBs
RZ_0	$\{2RB_1, 12RB_2, 3RB_3, 0RB_4\}$	415
RZ_1	$\{4RB_1, 7RB_2, 1RB_3, 1RB_4\}$	534
RZ_2	$\{0RB_1, 1RB_2, 1RB_3, 1RB_4\}$	294
RZ_3	$\{5RB_1, 4RB_2, 0RB_3, 0RB_4\}$	617
RZ_4	$\{8RB_1, 12RB_2, 0RB_3, 2RB_4\}$	410

Table 3: Results of the Bees Algorithm.

RZ	RPB	Tasks	Occupation rate of RZ
RZ_4	RPB_{192}	$T_{7,1}, T_{7,2}, T_{7,3}$ $(100\% of T_7)$	$83,1776\%$
RZ_0	RPB_{404}	$T_{1,1}, T_{1,2}, T_{1,3}, T_{1,4}$ $(100\% of T_1)$ $T_{2,1}, T_{2,2}, T_{2,3}, T_{2,4}$ $(100\% of T_2)$	$19,306\%$
RZ_1	RPB_{471}	$T_{3,1}, T_{3,2}, T_{3,3}$ $(100\% of T_3)$ $T_{4,1}, T_{4,2}, T_{4,4}$ $(83\% of T_4)$	$85,8802\%$
RZ_3	RPB_{1355}	$T_{6,1}, T_{6,2}, T_{6,3}, T_{6,4}$ $(100\% of T_6)$	$40,0484\%$
RZ_2	RPB_{1160}	$T_{4,3}(17\% of T_4)$ $T_{5,1}, T_{5,2}, T_{5,3}, T_{5,4}$ $(100\% of T_5)$	$48,5133\%$

6 CONCLUSIONS

In this paper, we propose a heuristic method to solve the problem of hardware tasks placement on FPGA technology (Virtex 5) based on the Bees Algorithm. We tried to optimize the parameters of this constrained optimization problem so that we reduce the cost function. We also avoid the tasks rejection as we pack tasks on RZs and we employ the dynamic partial reconfiguration. Our experimental results show the efficiency of this method in terms of resources and reconfiguration overhead.

We plan in a future work to propose another method and to compare it with the Bees Algorithm. Also, we will exploit the results of this off-line placement to achieve an on-line scheduling of the tasks taking into account the inter-task communication and real time constraints.

ACKNOWLEDGEMENTS

The authors would like to thank the national agency of research in France and the world-ranking *Secured Communicating Solutions* (SCS) cluster that sponsor our research project FOSFOR. This work was supported by AIMMS technical support and Xilinx tools.

REFERENCES

Ahmadinia, A., Bobda, C., Bednara, M., and Teich, J. (2004). A new approach for on-line placement on reconfigurable devices. *International Parallel and Distributed Processing Symposium (IPDPS), Santa Fe, NM, U.S.A.*, page 134.

Bazargan, K., Kastner, R., and Sarrafzadeh, M. (2000). Fast template placement for reconfigurable computing systems. *IEEE Design and Test, Special Issue on Reconfigurable Computing*, pages 68–83.

Danne, K. and Stuehmeier, S. (2005). Off-line placement of tasks onto reconfigurable hardware considering geometrical task variants. *IFIP conference*.

FOSFOR (2010). Fosfor project, 2010. http://www.polytech.unice.fr/ fmuller/fosfor.

Handa, M. and Vemuri, R. (2004). An efficient algorithm for finding empty space for online fpga placement. *Design Automation Conference (DAC), San Diego, California, USA*, pages 960–965.

Marconi, T., Lu, Y., Bertels, K., and Gaydadjiev, G. (2008). Task placement algorithm for partial reconfigurable systems. *Design Automation Test Europe (DATE), Munich, Germany*, pages 1346–1351.

Pham, D. and Ghanbarzadeh, A. (2007). Multi-objective optimisation using the bees algorithm. *Proceedings of IPROMS, Cardiff, UK.*

Pham, D., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., and Zaidi, M. (2006). The bees algorithm - a novel tool for complex optimisation problems. *Proceedings of IPROMS, Cardiff, UK.*

Pham, D., Haj Darwish, A., Eldukhri, E., and Otri, S. (2007a). Using the bees algorithm to tune a fuzzy logic controller for a robot gymnast. *Proceedings of IPROMS, Cardiff, UK.*

Pham, D. T., Afify, A., and Koc, E. (2007b). Manufacturing cell formation using the bees algorithm. *IPROMS conference*.

Xilinx (2009). Xilinx, 2009. virtex-5 fpga configuration user guide. Technical report.