# Co-Synthesis of Dynamically Reconfigurable SOPCs Specified by Conditional Task Graphs

Radoslaw Czarnecki and Stanislaw Deniziak*

*Cracow University of Technology, Warszawska 24, 31-155 Cracow, Poland*

**Abstract:** In this work the co-synthesis method, that optimizes dynamically reconfigurable multiprocessor SOPC system architectures, is presented. The algorithm maximizes speed of a SOPC system, taking into consideration the space constraints of the FPGA. The algorithm starts with the initial solution, where all tasks are assigned to only one general purpose processor module. Next, it produces new solutions using iterative improvement methods. To the best of authors' knowledge, it is the first algorithm that takes into consideration mutually exclusive tasks in optimization of dynamically reconfigurable systems. Such tasks are specified using conditional task graphs. Partially reconfigurable FPGAs enable reuse of the same hardware resources for mutually exclusive tasks. In this way, the area occupied by an embedded system can be decreased and free space may be used for other purposes. It was shown that the presented approach can also increase the performance of a SOPC system.

**Keywords:** Co-synthesis, dynamic reconfiguration, FPGA, conditional task graph.

## INTRODUCTION

Today's FPGAs enable the integration of a complex system on one device, also called System On Programmable Chip (SOPC). Moreover, many reconfigurable architectures support a partial and dynamic reconfigurability [1]. Dynamic reconfiguration [2], also called a run-time reconfiguration, is the ability to change a hardware configuration during the execution of a processing task. This feature enables a larger part of the application to be accelerated in hardware. In partially reconfigurable FPGAs [1] only a part of the configuration can be modified. In this way, computations may overlap with reconfigurations, reducing the reconfiguration time overhead.

The aim of this work is to present an algorithm for the resource constrained co-synthesis of dynamically reconfigurable SOPCs. The algorithm maximizes speed of a system taking into consideration space constraints associated with the maximal admissible area of an FPGA. The target architecture of the system consists of general purpose processor cores and dedicated hardware modules, all are implemented in one FPGA device. The algorithm starts with the initial solution, where all tasks are assigned to one general purpose processor module. Next, it produces new solutions using iterative improvement methods. The reconfiguration time is taken into consideration in the task scheduling algorithm, in such a way, that impact of this time on the performance of the system is minimized.

In the most embedded systems a lot of their functionalities are mutually exclusive (Mutually Exclusive Tasks – MET). Specifications of data-dominated embedded systems, containing mutually exclusive tasks, are usually represented as conditional task graphs. Information about such tasks, combined with dynamic reconfiguration, enable better reuse the hardware resources of an FPGA. Mutually exclusive tasks may be assigned to the same hardware resource, then the area of the implementation can be significantly decreased, thus more tasks can be implemented in hardware. This often leads to faster systems. To the best of authors' knowledge, the method presented in this paper is the first algorithm for the co-synthesis of dynamically reconfigurable multiprocessor embedded systems that are specified using conditional task graphs.

The rest of the paper is organized as follows. The next sections review related work and introduce the concept of dynamically reconfigurable systems. Then our algorithm, called CESEDYRES, will be presented. Next, experimental results are given. Two examples containing METs will demonstrate the effectiveness of our method. The paper ends with conclusions.

## RELATED WORK

Two types of algorithms have been applied to the co-synthesis problem: optimal methods based on exhaustive exploration or integer linear programming (ILP) [3, 4] and heuristic ones. Optimal methods give optimal solutions, but they are limited only to small instances of the co-synthesis problem. Hence more essential are heuristic methods. The following types of heuristics have been applied to the co-synthesis: probabilistic algorithms (i.e. simulated annealing, genetic algorithms), constructive algorithms and iterative improvement algorithms. Due to their performance and effectiveness the most promising are: genetic algorithms [5-7] and iterative refinement algorithms [8, 9].

Chatha and Vemuri formulated a reconfiguration problem in the co-synthesis [10], but it was limited only to one

*Address correspondence to this author at the Department of Computer Engineering, Cracow University of Technology, Cracow, Poland;
E-mail: s.deniziak@computer.org

1874-1142/08 

CPU (software) – one FPGA (hardware) topology. Similar target architecture was used in [11, 12]. In these approaches a coarse grained hardware/software partitioning algorithm was used. The Nimble compiler [13] also assumes architecture consisting of one general purpose processor and a dynamically reconfigurable FPGA. More general model, consisting of many FPGAs and heterogeneous processors was used in CORDS [5]. All of these methods assume full-chip reconfiguration; they do not consider partially reprogrammable FPGAs. None of them considers SOPC systems, but all of these methods use multi-chip architectures.

In the evolutionary algorithm called SLOPES [14], partially reconfigurable Xilinx FPGAs are used, but the method does not consider the problem of placement of reconfigurable modules in the FPGA device. In [15] a multiprocessor SOPC architecture that can be implemented in partially reconfigurable FPGAs is considered, but the system is represented as a simple linear task graph. The method for the co-synthesis of dynamically reconfigurable systems is presented in [16]. The target architecture includes an embedded processor that dynamically changes functionality of the system, but this method is not applicable for multiprocessor systems. Another approach [3] considers advantages of partially reconfigurable FPGAs, but it is based on ILP method, thus it can be applied only for co-synthesis of small systems.

Other approaches concentrate on design environments for the implementation of dynamically reconfigurable systems. In [17] an IP-based design platform is presented. PaDReH [18] is a framework for design, implementation and validation of dynamically reconfigurable systems. A self-reconfigurable platform, where an FPGA dynamically reconfigure itself under the control of an embedded processor, is presented in [19]. Approaches concerning the realization of systolic arrays in FPGAs [20, 21] also consider similar problems.

None of the known co-synthesis methods targets at dynamically reconfigurable multiprocessor SOPCs. Most of them are dedicated to outdated FPGAs. Only [12, 22] impose strict linear placement constraints. Architectures generated without considering the physical location of tasks may be unrealizable in many modern FPGAs (e.g. Xilinx Virtex devices). Both methods are dedicated to HW/SW architectures containing only one CPU. The approach [12] is based on the Kernighan-Lin/Fiduccia-Matheyes (KLFM) heuristic that iteratively improves solutions by simple movements of tasks. In [22] the co-synthesis is based on the genetic approach and an improved list scheduling algorithm. The approach presented in this paper also specifies the co-synthesis of dynamically reconfigurable systems as a constrained placement problem.

The co-synthesis methods operate on internal model of a system that is based on specification in the form of communicating tasks. Task graph (TG) is the most popular representation of an embedded system [5, 6, 10, 12, 23]. In a few works conditional task graphs are also considered [24-27]. However, these methods are not intended for co-synthesis of dynamically reconfigurable systems. In some other works conditional scheduling algorithms are described [28, 29], but they are not dedicated to scheduling of tasks in DRSOPC systems. The method presented in this paper considers conditional task graphs (CTG) as a representation of an embedded system that can be dynamically reconfigured.

The main contributions of the approach presented in this work are the following:

- it considers multiprocessor SOPC architectures. To the best of authors' knowledge there is no known co-synthesis method dedicated to dynamically reconfigurable multiprocessor SOPC systems;
- it incorporates a physical layout of hardware modules, like in [12] and [22], but it takes into account the placement of processor cores, too;
- it operates on the conditional task graph. Although there are some co-synthesis methods using CTG, none of them can be applied to dynamically reconfigurable SOPC systems;
- the target system is self-reconfigurable. Most of the co-synthesis methods assume that run-time reconfiguration is performed using an external hardware. In the presented approach the target system architecture that includes reconfiguration controller is generated;
- heuristics used in the co-synthesis algorithm are capable of escaping from local minima. Experimental results showed that the presented algorithms usually find significantly better solutions than the first local maximum of performance.

## BASIC CONCEPTS AND DEFINITIONS

Let the conditional graph *CTG=(V,E)* represents mutual relationships between tasks of a system. The node $v_i \in V$ represents *i*-th task. The edge $e_{ij} \in E$ corresponds to the communication between tasks $v_i$ and $v_j$. The simple edge $e_{ij} \in E_s$ is an edge without assigned condition. The conditional edge $e_{ijk} \in E_c$ is an edge with assigned condition $c_k$. The following equations have to be fulfilled:

$$E_s \cap E_c = \varnothing \text{ and } E_s \cup E_c = E \quad (1)$$

Granularity of tasks is such that data transmission is the last activity of each task. Task $v_j$ may start, only when all its predecessors have finished their execution and when all data to this task have been transferred. The volume $d_{ij}$ of a data transfer is associated with the edge $e_{ij}$, if any data dependency occurs. Data transmission runs independently from the data processing. Fig. (**1**) shows a sample conditional task graph. Any number of conditional edges can come from one branch fork task $v_{fork}$, but all paths corresponding to the same condition have to join in the same join task ($v_{join}$). Conditions may be hierarchical. Two tasks $v_i$ and $v_j$ are mutually exclusive when exist $P_{k1}$, $P_{k2}$, $e_{klm} \in E_c$, and $e_{kpn} \in E_c$ such that: $c_m \wedge c_n = false$ and $e_{klm} \in P_{k1}$ and $e_{kpn} \in P_{k2}$ and $v_i \in P_{k1}$ and $v_j \in P_{k2}$, where $P_{k1}$ and $P_{k2}$ represent conditional paths starting from the node $v_k$.

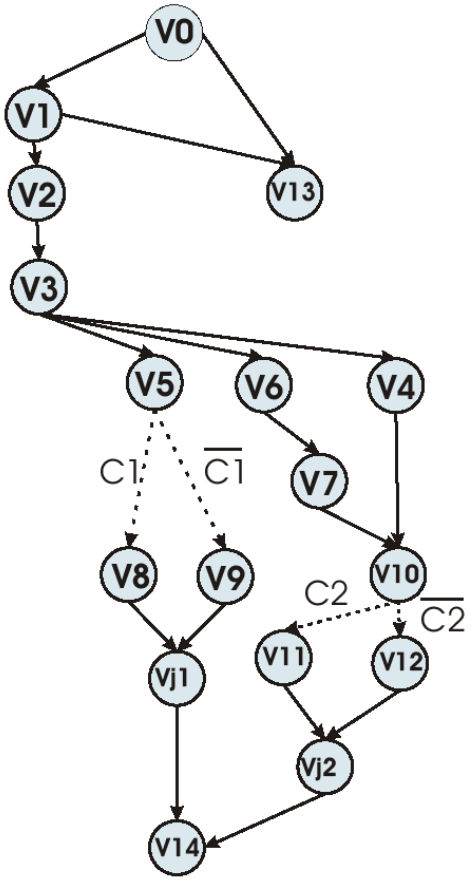


**Fig. (1).** Sample conditional task graph.

It is assumed that a library of software and hardware components that can be used for implementation of the system is given. In the library there are three types of resources: general purpose processors (G*PPs*), hardware virtual components (*VCs*) and communication links (*CLs)*.

Each task is characterized by the following parameters: $S_i(v_j)$ - the area (the number of FPGA logic cells) occupied by $VC_i$ executing task $v_j$, $C_i(v_j)$ - the amount of memory required to execute the task $v_j$ (only for software implementation), $t_i(v_j)$ – the time of execution of the task $v_j$ on the given processing element. Values of $S_i(v_j)$ and $t_i(v_j)$ are known for IP modules. For another tasks they can be computed using estimation methods [30].

Communication link is characterized by its area $Sc_i$ and its bandwidth $b_i$. The time of the transmission through $CL_k$, between tasks $v_i$ and $v_j$ is defined as follows:

$$t_k(v_i,v_j) = \begin{cases} \left\lceil \frac{d_{ij}}{b_k} \right\rceil & \text{- when tasks are allocated to different resources,} \\ 0 & \text{- when tasks are allocated to the same resource.} \end{cases} \quad (2)$$

Tables **1** and **2** show a sample resource library corresponding to the system specified by the graph from Fig. (**1**).

**Table 1. Library of Software and Hardware Components**

| | FPGA Area: 2000 CLBs, tr=0.86 µs / CLB | | | |
|---|---|---|---|---|
| | GPP (Su=203 CLBs) | | VC | |
| Task | $t_i(v_j)$ [µs] | $C_i(v_j)$ | $t_i(v_j)$ [µs] | $S_i(v_j)$ [CLB] |
| $v_0$ | 434 | 30 | 89 | 158 |
| $v_1$ | 817 | 27 | 94 | 227 |
| $v_2$ | 811 | 23 | 59 | 92 |
| $v_3$ | 747 | 7 | 57 | 80 |
| $v_4$ | 444 | 40 | 54 | 175 |
| $v_5$ | 1020 | 43 | 33 | 411 |
| $v_6$ | 755 | 34 | 1 | 319 |
| $v_7$ | 335 | 86 | 22 | 283 |
| $v_8$ | 250 | 57 | 5 | 482 |
| $v_9$ | 655 | 66 | 61 | 152 |
| $v_{10}$ | 382 | 32 | 94 | 224 |
| $v_{11}$ | 408 | 24 | 30 | 184 |
| $v_{12}$ | 945 | 29 | 21 | 109 |
| $v_{13}$ | 190 | 32 | 26 | 167 |
| $v_{14}$ | 881 | 8 | 40 | 181 |

**Table 2. Parameters of the Communication Link**

| CL | CC [CLB] | b | Availability |
|---|---|---|---|
| B1 | 10 | 9kB/µs | GPP, all VC (HW) |

It is assumed that the target architecture includes processor cores *GPPs* and dynamically reconfigurable sectors *RSs* that communicate with each other by communication links *CLs*. Hardware components (*VCs)* are placed in sectors *RSs*. One or more *VCs* can be assigned to one sector in the same time frame, thus *RS* can execute exactly one selected subset of tasks. Tasks assigned to one *RS* may run in parallel. After all tasks assigned to the same sector have finished their execution the sector is reconfigured to allocate new *VC* modules.

Areas of all available *RSs* are calculated as the sum of areas of some *VCs* from the library, next they are justified according to the requirements of a module based reconfiguration. The algorithm of *RSs* generation is explained in the next section. Reconfiguration time for each *RS* is calculated on the basis of the reconfiguration time of one logical cell ($t_r$), which is taken from the datasheet of a target FPGA.

It is assumed that the target architecture includes an additional embedded processor *GPPr*, which controls reconfiguration process.

There are two other parameters associated with *GPPs*: the maximum memory load ($CM_i$) is a maximum memory that can be used for an application, and the area occupied by the processor core ($Su_i$). There is also one parameter defined for $RS_i$ : the area of an FPGA occupied by the sector ($S_{RSi}$).

Let the architecture of a system be composed of $p$ processor cores $GPP_i$ ($i$=1,…, $p$), one processor that controls reconfiguration $GPP_r$ occupying the area $Su_r$, $r$ sectors $RS_i$ ($i$=$p$+1,…, p+$r$) and $c$ communication links $CL_j$ ($j$=1,…, $c$). The total area of the system is defined as follows:

$$S = \sum_{i=1}^{p} Su_i + \sum_{i=p+1}^{p+r} S_{RSi} + \sum_{j=1}^{c} Sc_j + Su_r \quad (3)$$

Execution time of all tasks in the system is defined as follows:

$$T=max(max(t_k(GPP_1),...,t_k(GPP_p)),max(t_k(VC_{p+1}),...,$$
$$t_k(VC_r)),max(t_k(CL_1),..., t_k(CL_c))) \quad (4)$$

where: $t_k(PE_j)$ is the finish time of a task scheduled as the last one on $PE_j$ (($VC_j$ or $GPP_j$) and $t_k(CL_j)$ is the finish time of a communication scheduled as the last one on $CL_j$. Therefore, performance of the SOPC system is the following:

$$\lambda=1/T \quad (5)$$

The goal of the co-synthesis is to find the fastest architecture of a self-reconfigurable SOPC system that meets the functional requirements given by *CTG* graph. The architecture has to guarantee feasible implementation (assuming module reconfiguration) and hasn't to exceed size of a target FPGA.

## INITIALIZATION

In the presented approach the hardware-software co-synthesis starts with an initialization. In this step all available sizes of *RS* are being calculated. Next, an initial solution is created.

The best sector size is the size that can contain as many as possible different groups of tasks. To achieve greater flexibility a few different sizes of *RS* should be available. Let $r$ be the number of *VC*s in the library, $l$ - the maximum number of sizes to be generated and $C_j^i$ - the $j$-th subset of any $i$ different *VC*s. The algorithm of computing $l$ available sizes of sectors looks as follows:

```
RSInit(){
i=1;
Sv={};
Ss={};
repeat {
    for each subset C_j^i do {
     s = Σ_{VC_k ∈ C_j^i} S(VC_k);
    if s ≤ max(S(VC_1),...,S(VC_r)) then Sv=Sv ∪ {s};
    }
     i++;
} until i>r;
for k=1 to l do {
    Find s_k which occurs most times (at least 2 times) in
    Sv;
if there is not such s_k then
Find s_k∈Sv such that values from the range <s_k-Δ;
s_k+Δ> occurs most times in Sv;
Remove all occurrences of s_k from the set Sv;
    }
for k=1 to l do {
Round s_k to the nearest available size of RS;
     Ss=Ss ∪ {s_k};
    }
}
```

In the first loop the algorithm computes sums of areas of all possible *VC* groups. Next (the second loop), it chooses sums which are most frequent or sums which values are the most similar (values are in a given range). In the last loop it rounds these sums to the nearest value of a reconfigurable block size required by the module reconfiguration and chooses them as available sector sizes.

The initial solution is the architecture where all tasks are assigned to one general purpose processor. Such solution leaves most space in FPGA available for *RSs*.

If an embedded system is represented by the conditional task graph, then all mutually exclusive tasks should be determined in the initialization step. For this purpose, the algorithm of a CTG labeling is used.

## CTG LABELING

Determining mutually exclusive tasks in the CTG is not straightforward, especially when conditions are hierarchical. The algorithm of the CTG labeling simplifies this problem. For each task, a few labels will be assigned. The label $c_k(v_j)$ equals *cond*, where *cond* is the identifier of a condition (all conditions are numbered). The label $level(v_j)$ corresponds to the level of $v_i$ in the hierarchy of conditional paths. The level is incremented after each conditional edge belonging to the same path in the CTG. The table of labels $v_j$->$fork[level(v_j)]$ contains task numbers that checks the condition corresponding to the level $level(v_j)$ in the hierarchy of conditional paths. The algorithm of the CTG labeling is the following:

```
labeling (v_j) {
if level(v_j)==0 then { c_k(v_j)=0; cond=0;}
for each successor v_i of v_j do {
    if e_ij is an edge without condition and c_k(v_j)=0, then {
        level(v_i)= level(v_j); c_k(v_i)=0;
    }
    else {
        if e_ij is a conditional edge then {
         level(v_i)= level(v_j)+1;
         cond++;
         c_k(v_i)=cond;
        }
      else if v_i ≠ v_join then{
                    c_k(v_i)= c_k(v_j);
         level(v_i)= level(v_j);
        }
        else if v_i==v_join then{
       level(v_i)= level(v_j)-1;
```

```
            ck(vi)=ck(vi->fork[level(vi)]);
              }
              for (m= level(vi); m ≥ 1; m--) do
              vi->fork[m]=vj->fork[m];
              }
        labeling(vi);
}
```

If execution of $v_i$ is not dependent on any condition then $level(v_j)$=0, and $c_k(v_j)$, $v_i$->*fork* are undefined. The edge without condition may be outside the conditional path (the first *if* statement in *for* loop) or may belong to the conditional path (the third *if* statement in *for* loop). If there are some tasks in the conditional path starting from $v_{fork}$ and ending in $v_{join}$, then each of them has the same labels $c_k$ and *level* as the immediate successor of the $v_{fork}$. If there is any conditional edge belonging to the conditional path then the level and the condition number are increased (the second *if* statement). If the task belonging to the conditional path is the $v_{join}$ task then $c_k$ and *level* are decreased. In the last *for* loop, the table of all numbers of fork tasks corresponding to $v_i$ task, belonging to the conditional path, is created.

## ARCHITECTURE REFINEMENT

The presented method of the hardware-software co-synthesis (COSEDYRES) is an iterative improvement procedure that creates new solutions (architectures of a system) through modifications of previous ones. Refinement process is controlled by the gain that describes quality of the improvement. The gain is the difference between two compared solutions. Quality of the solution is usually characterized by a few parameters (e.g. cost, performance). Two methods of the modification are used: adding one *GPP* or *RS,* and removing one *GPP* or *RS*. Both modifications can be made in one step of the algorithm. In this way, moving group of tasks, from one resource (processor or sector) to another, is possible in one step. Algorithm stops when there are no possibilities to obtain better solutions.

In the COSEDYRES, the physical constraints on placement of reconfigurable modules are taken into account [16]. Reconfigurable sectors are always strict linearly placed, i.e. each *RS* occupies a contiguous set of CLB columns. Such restrictions eliminate some possible optimal solutions, but physically unrealizable because of placement infeasibility.

The list scheduling method, where priorities are assigned to each task, is applied in the algorithm. For each task $v_i$, times of execution of tasks, on all paths starting from $v_i$ are computed. The longest time is taken as the priority of the task. Tasks with higher priorities are scheduled first. Task scheduling, allocation of *CL*s and communication scheduling are done simultaneously.

In each iteration step all possible modifications of the current solution are evaluated. The best solution, i.e. a solution giving the best gain, is chosen to the next step. To increase the probability of getting out of local maxima of performance, the possibility of optimization in the next steps is also taken into consideration, in the calculation of the gain.

Let the parameter α be defined as the available space in the FPGA:

$$\alpha = S_{max} - S_{cur} \tag{6}$$

where: $S_{max}$ is the area of the target FPGA device and $S_{cur}$ is the area of the current solution. The best solution is a solution with the highest λ. But from the other side, for solutions with the highest α, there is a greater probability of optimization in the next steps of the refinement. Thus, the gain $\Delta E$, that describes the quality of the improvement, is defined as follows:

$$\Delta E = \begin{cases} \Delta\alpha * \Delta\lambda & when\ \Delta\alpha > 0 \\ \Delta\lambda & when\ \Delta\alpha = 0\ and\ \Delta\lambda > 0 \\ -\Delta\lambda / \Delta\alpha & when\ \Delta\alpha < 0 \\ 0 & when\ \Delta\lambda \leq 0 \end{cases} \tag{7}$$

where: $\Delta\alpha = \alpha(A^{cur}) - \alpha(A^{prev})$ is the increase in the available area caused by the modification, $\Delta\lambda = \lambda(A^{cur}) - \lambda(A^{prev})$ is the increase in the speed, $\alpha(A^{cur})$ and $\lambda(A^{cur})$ are parameters of the current solution, $\alpha(A^{prev})$ and $\lambda(A^{prev})$ are parameters of the best solution, found in the previous step.

Let $PE_i$ be any processing element taken from the library of available resources ($GPP_i$ or $RS_i$). The draft of the co-synthesis algorithm for dynamically reconfigurable SOPC systems is the following:

```
COSYDYRES(){
  Generate available sizes of RS;
  Label CTG graph;
  Create an initial architecture A;
  Compute the area Scur of A;
  Compute the performance λcur of A;
  repeat
  Gain=0;
  for each PEi do {
  if (number of resources in A)≥2 then {
  for each PEj ∈ A do {
  A'=A - PEj;
  for each vk∈ PEj do {
        Find PEl∈A' that gives greatest value of δE
        after moving task vk to it;
  Assign vi to PEl
      }
  if ΔE>Gain then {
  Gain=ΔE;
      A^best=A';
      }
    }
  }
  A''=A' ∪ PEi;
  repeat
      Find task vk which gives greatest δE after moving it
      to PEi;
  if δE>0 then assign vk to PEi
  until there is no task that gives δE>0 after moving;
  A''=A''- PEs without tasks assigned;
```

```
    if ΔE>Gain then {
    Gain=ΔE;
          A^best=A'';
      }
    Place sectors in the FPGA;
    }
    if Gain>0 then A=A^best;
    until Gain == 0;
}
```

In the first part, the COSEDYRES algorithm tries to remove one resource (PE) from the architecture. All tasks from the removed PE are moved to other PEs (the inner *for* loop) according to the greatest value of a local gain $\delta E$ (which will be described later). To remove PE the number of resources in the architecture has to be greater then two. This step is repeated for each PE in the architecture then the solution giving the highest gain is stored as the best architecture. In the second part, the algorithm tries to allocate a new resource (this step is repeated for each resource from the library) and moves tasks from other PEs to the new one according to the highest gain (in the inner *repeat* loop). If the solution is better then the previous one then it is taken as the best architecture $A^{best}$.

The refinement is repeated until there is no better solution (the gain is not greater than 0). In the inner loops, where tasks are moved from one *PE* to another, the area of the SOPC system does not change. Hence, the new parameter: local gain $\delta E$, is used to rate such modifications. The performance of a system is not always increased after moving a task $v_k$, but may lead to higher speed of the system in further steps. When the execution time of $v_k$ in software is longer than the sum of the reconfiguration time and execution time of $v_k$ in hardware, then moving $v_k$ from software to hardware increases the probability of getting architecture with higher performance (more tasks are executed in hardware). If there are no possibilities to improve the solution then we assume that $\delta E$=-1. Finally, the local gain is defined as follows:

$$\delta E = \begin{cases} \Delta\lambda & when\ \Delta\lambda > 0; \\ (t_i(v_k) - t_{res(RSj)})/t_j(v_k) & when\ \Delta\lambda = 0\ and \\ & t_{res(PEj)} < t_i(v_k) \\ -1 & in\ other\ cases \end{cases} \quad (8)$$

where: $t_i(v_k)$ and $t_j(v_k)$ are times of execution of task $v_k$ by *PEs* from which and to which the task is moved, respectively; $t_{res(RSj)}$ is the reconfiguration time of the sector $RS_j$.

The last step in each pass of the refinement is a sector placement. Details are given in the next section. $A^{best}$ is the best architecture found in the current pass and it is taken as the initial architecture in the next pass.

If a few MET tasks $v_i,\ldots,v_{i+n}$ are assigned to the same *RS*, then the area of such *RS* has to be computed as: $max(S_{vc}(v_i),\ldots, S_{vc}(v_{i+n}))$, where $S_{vc}(v_i)$ is the area of *VC* implementing $v_i$. The algorithm prefers to assign MET tasks to the same *RS*. In this way the area of a system is decreased and then more tasks may be assigned to hardware, thus the system performance may be higher. When MET tasks are assigned to the same *GPP*, they can be scheduled in the same time frame; therefore the scheduling algorithm takes them into account, too.

## SECTOR PLACEMENT

It is assumed that the basic reconfigurable module is a frame, spanning the height of an FPGA (as in Xilinx FPGAs). Each *RS* consists of multiple adjacent frames. Hence, *RS* placement should be strict linear. Another constraint concerns the communication scheduling: the sector reconfiguration and any transmission through this sector are not allowed in the same time. Therefore, transmissions can not overlap in time with reconfiguration.

To find the best sector placement, all possible layouts of *RSs* in an FPGA are evaluated. For each *RSs* placement, task scheduling and communication scheduling, satisfying reconfiguration requirements, are performed. The fastest implementation is selected as the $A^{best}$. The draft of the placement algorithm is the following:

```
SectorPlacement(){
  Find all permutations of sectors in A^best;
      for each permutation A^p do {
       Schedule tasks, communications and
              reconfigurations;
              for each RS_k do{
                    if there are transmissions through RS_k
                    during the
               reconfiguration of the RS_k then{
             Modify the schedule by changing reconfigura-
             tions start times and/or start times of transmis-
             sions;
                                              }
              }
              Compute λ(A^p);
              if λ(A^p)> λ(A^best) then A^best=A^p;
      }
}
```

For each permutation of sectors, the algorithm checks if the reconfiguration time frame of $RS_k$ does not overlap with the time frame of the data transmission, between two communicating resources located on both sides of the $RS_k$. If there is such conflict, the reconfiguration or the transmission is delayed. Among all permutations, the one with the best performance is taken as the $A^{best}$ in the current step of the algorithm.

Fig. (**2**) illustrates sector placement in the architecture obtained for the system specified by the task graph from Fig. (**1**), using the algorithm described above. Fig. (**3**) shows the tasks scheduling and reconfigurations scheduling for this implementation. There are two dynamically reconfigurable sectors: *RS1* and *RS2*, two *GPPs* and a control processor

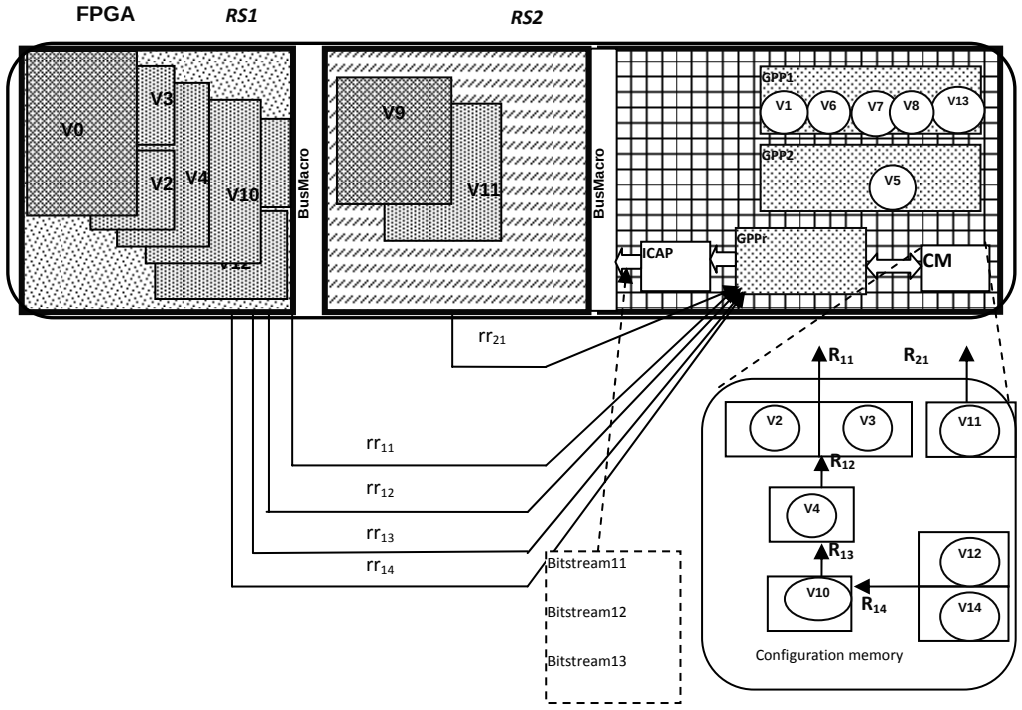


**Fig. (2).** Sector placement.

*GPPr*. All tasks assigned to *RS1* and *RS2* are implemented in hardware. Dynamic reconfiguration allows allocating more tasks than the sector size. Arrows show the order of reconfigurations. Such placement of sectors allows avoiding conflicts between reconfigurations and data transmissions. The only conflict that would appear is associated with the reconfiguration of the sector *RS2* and the transmission from *v7* to *v10*. To avoid it, the reconfiguration of this sector is delayed till the end of the data transmission from *v7* to *v10*.

## SELF-RECONFIGURABLE SYSTEMS

Architecture generated using COSEDYRES is supplemented with the *GPPr* processor that controls the reconfiguration of sectors. *GPPr* is a general purpose processor or a special IP module. Part of an FPGA is reserved for *GPPr*, before the algorithm starts. In this way the whole system is implemented in one FPGA and there are no other external modules to control reconfiguration process.

The method of implementation of self-reconfigurable systems is based on the Xilinx platform for dynamically self-reconfigurable architectures (SRP) [19]. One processor controls the reconfiguration of sectors. It can be an IP module like MicroBlaze soft core or a hardware core like the embedded PowerPC. Only one sector can be reconfigured at the given moment. The fastest 32-bits SelectMAP programming mode is used.

Bitstreams are generated by standard Xilinx tools according to Modular Design Flow [1]. One bitstream is generated for the initial system configuration, and partial reconfiguration bitstreams are generated for all configurations of each *RS* separately. Each reconfiguration starts after finishing preceding task or transmission. The procedure controlling reconfigurations that is executed by the *GPPr* looks as follows:

*Reconfigure(){*
*Load initial configuration $A_i$ to FPGA;*
***do{***

***wait*** *until $rr_i$=true;*

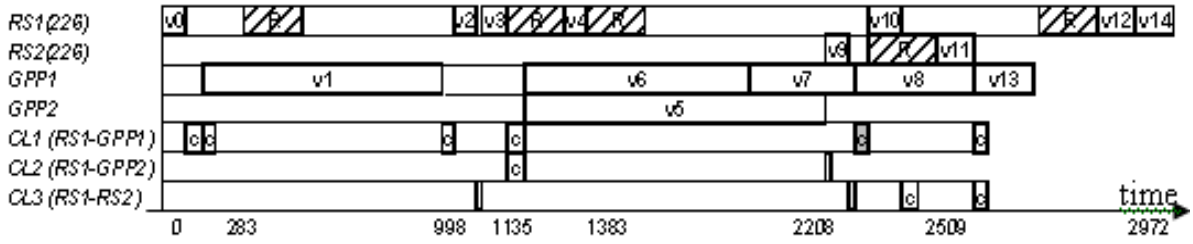


**Fig. (3).** Gantt charts of scheduled tasks and reconfigurations

**Table 3. Experimental Results for DRSOPC and SOPC Implementations**

| Nodes [N] | FPGA Area | DRSOPC | | | SOPC | | | Performance inc. [%] |
|---|---|---|---|---|---|---|---|---|
| | | Time | Area | Architecture | Time | Area | Architecture | |
| 10 | 1100 | **2509** | 933 | 2p,1s,5hw | **4078** | 927 | 2p, 3hw | 62% |
| 30 | 1500 | **4302** | 1498 | 3p,2s,18hw | **6734** | 1425 | 4p, 2hw | 57% |
| 50 | 2000 | **5926** | 1786 | 3p,3s,29hw | **8352** | 1825 | 6p, 3hw | 41% |
| 70 | 2500 | **6435** | 2480 | 5p,2s,30hw | **9605** | 2407 | 7p, 5hw | 49% |
| 90 | 3000 | **8508** | 2865 | 4p,4s,43hw | **9707** | 2658 | 8p, 6hw | 14% |
| 110 | 3500 | **8360** | 2708 | 5p,5s,50hw | **8859** | 3353 | 11p, 7hw | 6% |
| 130 | 4000 | **8815** | 2661 | 6p,3s,56hw | **9564** | 3995 | 12p, 13hw | 9% |
| 150 | 4500 | **10891** | 3332 | 7p,4s,60hw | **13037** | 4295 | 9p, 13hw | 20% |

*Get the next bitstream for $RS_i$ from a cache;*

*Reconfigure $RS_i$ with the new bitsream;*

*} **while** there are no waiting bitstreams in the cache;*

*}*

GPPr waits for any reconfigure request signal ($rr_i$), which is activated by PE finishing preceding task or transmission. Then, system is dynamically reconfigured by loading next bitstreams for the sector corresponding to the $rr_i$ signal. Sectors are reconfigured through special ICAP port (Internal Configuration Access Port) [19] (Fig. **2**).

## EXPERIMENTAL RESULTS

In this section some experimental results, showing the efficiency of COSEDYRES algorithm, will be presented. First, the benefits of the dynamic reconfigurability will be demonstrated. For this purpose, the same systems will be synthesized and implemented as DRSOPC and SOPC systems [31, 32]. Next, by comparing results of the co-synthesis of systems represented by *TG* and *CTG* graphs, the benefits of considering mutually exclusive tasks in the system optimization will be presented.

First, *TG* from Fig. (**1**) (but without considering MET tasks) and a library of available resources (Tables **1** and **2**) were considered. For the target FPGA consisting of 2000 CLBs and reconfiguration time of one CLB equal to 0.86 μs, the DRSOPC system with execution time equal to 1818 μs and the area equal to 1974 CLBs was received. The same system implemented as SOPC executes all task in 2463 μs and had the area equal to 1900 CLBs.

In order to estimate average benefits of the dynamic reconfiguration, some systems, specified by different random task graphs (from 10 to 150 nodes), where synthesized as SOPC and DRSOPC architectures. In the library of available resources were: one *GPP*, one *VCs* for each task and one communication link. The area of the target FPGA was adjusted to the number of tasks. Experimental results are given in Table **3**. The following columns of Table **3** contain: the size of the graph (number of nodes), the area of the FPGA, characteristics of architectures (for DRSOPC systems and SOPC systems). Architectures are characterized by: the total execution time of all tasks, the area and the number of *GPPs*, *RSs* and *VCs*. The last column shows speed-up of the system with dynamic reconfiguration in comparison with the system without dynamic reconfiguration (in percentage).

The experimental results show advantages of using partially reconfigurable FPGAs. Dynamically reconfigurable systems usually are significantly faster than these without dynamic reconfiguration. The same regions of the FPGA were used a few times for different functionalities, so more tasks were executed in hardware. The main bottleneck of dynamically reconfigurable systems is the time required for the reconfiguration, but proper scheduling of tasks lets minimize this problem.

Besides the quality of obtained results, the efficiency of the algorithm should be estimated by its computational complexity. Table **4** shows the estimation of the computational complexity based on experimental results. The following columns of Table **4** contain: the size of a graph, the total number of evaluated solutions, the number of passes, the computational complexity (CPU seconds to the number of nodes ratio), the number of solutions to the number of nodes

**Table 4. Computational Complexity of the COSEDYRES Algorithm**

| Nodes [N] | No. of Solutions l | No. of Passes k | $(CPU/n^3)*10^4$ | $l/n^2$ |
|---|---|---|---|---|
| 10 | 16 | 4 | 2.04 | 0.16 |
| 30 | 90 | 8 | 0.79 | 0.10 |
| 50 | 115 | 8 | 0.51 | 0.05 |
| 70 | 212 | 10 | 0.89 | 0.04 |
| 90 | 463 | 15 | 2.43 | 0.01 |
| 110 | 386 | 12 | 1.99 | 0.03 |
| 130 | 531 | 10 | 1.95 | 0.03 |
| 150 | 807 | 14 | 2.15 | 0.03 |

**Table 5. Comparison of results for COSEDYRES and COSEDYRES-CTG**

| Nodes [N] | FPGA Area | COSEDYRES | | | COSEDYRES-CTG | | | Number of MET | Performance Increase [%] |
|---|---|---|---|---|---|---|---|---|---|
| | | Time | Area | HW | Time | Area | HW | | |
| 10 | 1500 | **941** | 1498 | 4 | **804** | 1490 | 5 | 1 | 15 |
| 30 | 2000 | **4642** | 1863 | 13 | **3448** | 1986 | 15 | 2 | 26 |
| 50 | 2500 | **7054** | 2303 | 6 | **6531** | 2373 | 13 | 3 | 8 |
| 70 | 3000 | **9174** | 2935 | 21 | **8396** | 2964 | 35 | 4 | 9 |

ratio. On the basis of experiments for random task graphs, the computational complexity of the COSEDYRES algorithm can be estimated by: $O(n^3)$, where $n$ is the number of nodes, and the complexity depending on the number of solutions can be estimated by $O(n^2)$. The theoretical analysis indicated that the complexity of COSEDYRES can be estimated by $O(r^2n^2)$, where $r$ is the number of resources available in the library. In these examples there were two types of resources (one *GPP* and one *VC* for each task), so experimental results confirm the theoretical analysis. Moreover, in practice the computational complexity of the presented algorithm is even a little lower.

Finally, benefits of the optimization based on mutually exclusive tasks were estimated. The results of the synthesis of systems specified by random CTG graphs (COSEDYRES-CTG) were compared with results obtained for the same systems specified by TG graphs (COSEDYRES). The number of MET tasks in CTG graphs was adjusted to the number of nodes (such tasks were placed randomly in each CTG). The following columns of Table **5** presents: the size of the graph, the FPGA area, characteristics of architectures. Each system is characterized by: the execution time, the area and the number of hardware tasks in the architecture. The number of MET tasks in CTG is also given. The last column shows speed-up of the system that is represented by CTG in comparison with the system represented by TG (in percentage).

The experiments showed that considering mutual exclusive tasks, specified by CTG, enables to increase performance of a DRSOPC system using the COSEDYRES method. If MET tasks are assigned to the same reconfigurable sector, more tasks can be executed in hardware. METs are allocated only when the proper condition is satisfied (after reconfiguration of a sector). The performance increase depends on the number of MET tasks in CTG. If the number of METs in CTG is low then the performance increase may not be significant.

## EXAMPLE 1: USB HUB

The first example, presenting advantages of our method, is the co-synthesis of the USB 2.0 Hub. The most complex function in the Hub is the Transaction Translator (TT). Other functions should be implemented in hardware or have no hard constraints. Therefore, the co-synthesis will be only used to optimize a module implementing the TT. Specification of the TT may be composed of the following processes: HSH that implements communication between TT and a host (HSH may be decomposed into two processes: HSHR - receiving data from the host, and HSHT - transmitting data to the host), SSF that manages periodic transmissions from the

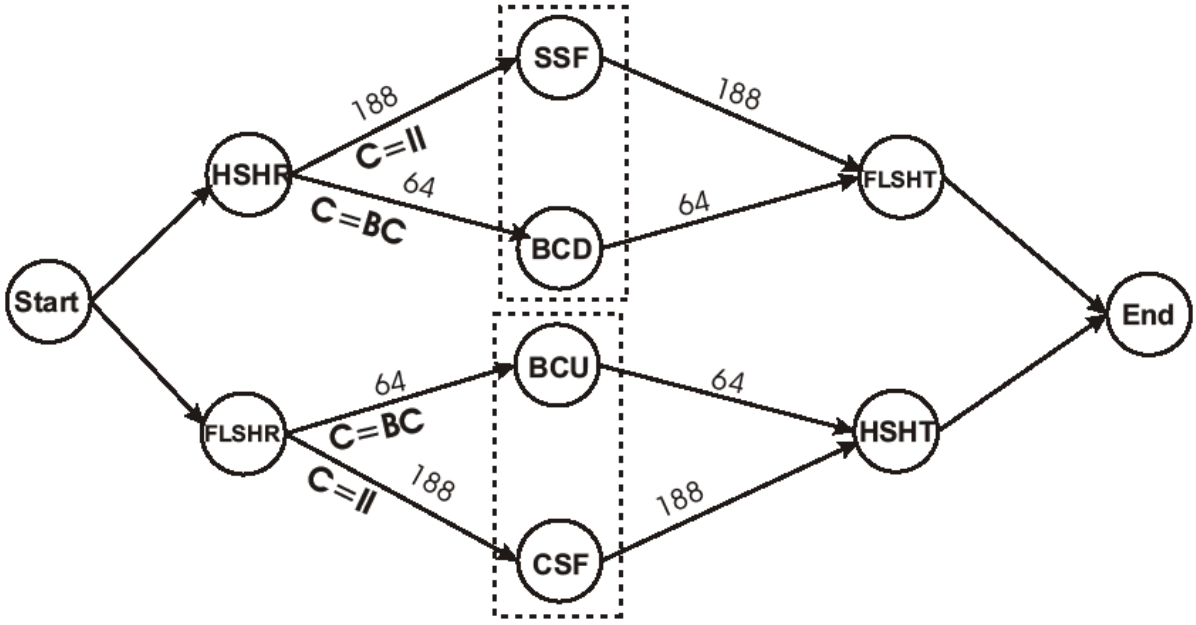


**Fig. (4).** Conditional task graph for TT.

host to devices, CSF – responsible for periodic transmissions from devices to the host, BC – manages block transmissions from devices to the host (decomposed into BCU and BCD). The CTG graph specifying the TT is presented in Fig. (**4**). Two pairs of tasks are mutually exclusive: (SSF, BCD) and (BCU, CSF). Information about it may allow performing better optimization.

Assume, that only one *GGP* module (with an area equal to 100 CLBs), and only one communication channel (with bandwidth equal to 80 MB/s), are available. The library of software and hardware components is given in Table **6**. The amount of data transmitted between tasks is showed in Fig. (**4**) as labels of edges. Time required for transmissions will be the following:

- for transmission of 188 B: 188/ 80 MB/s = 2350 ns,
- for transmission of 64 B: 64 B / 80 MB/s = 800 ns.

**Table 6. Resource Library for the TT**

| Task | SW | HW | |
|---|---|---|---|
| | t [µs] | t [µs] | S [CLB] |
| **HSHR** | 2400 | 20 | 400 |
| **HSHT** | 2400 | 20 | 400 |
| **SSF** | 1800 | 12 | 100 |
| **CSF** | 2000 | 12 | 135 |
| **BCD** | 1600 | 12 | 120 |
| **BCU** | 1600 | 12 | 120 |
| **FLSHR** | 1400 | 8 | 170 |
| **FLSHT** | 1400 | 8 | 170 |

It was assumed that the area of the TT is limited to **800 CLBs**. First, the TT was synthesized without taking into consideration the MET tasks. As a result, the architecture with the total execution time equal to **4408 µs** and the area equal to 741 CLBs was found. In this architecture 3 tasks were executed by *GPP* and others in hardware. Moreover, there was no possibility to increase the system performance using dynamic reconfiguration.

Next, the COSEDYRES method was used for the synthesis of the TT specified by the CTG graph from Fig. (**4**). The architecture consisting of two *GPP*s and two *RS*s were found. All MET tasks were assigned to the same *RS* sector. In this way, there is not necessary to allocate these tasks in the same time period, but they can be allocated, depending on actual value of the corresponding condition. Hence, the area occupied by these tasks was decreased twice. Task scheduling in this implementation is presented in Fig. (**5**). The total time of execution of all tasks in TT equals **2993 µs** and area equals 760 CLBs. This is the best solution for assumed space constraint. This example shows that taking into consideration MET tasks enables to get significantly faster architectures (about 33% in this case).

Next, the refinement process of the TT architecture will be demonstrated. Fig. (**6**) illustrates successive solutions chosen by the COSEDYRES algorithm during the refinement. Big points indicate solutions with the highest gain found in each step. It should be noticed that the algorithm is not greedy; it chooses not only solutions with the highest performance, sometimes chooses also slower ones but with larger available space in the FPGA. Such solutions give higher probability of achieving better solution in the next steps (like solution 8).

## EXAMPLE 2: EMBEDDED WEB SERVER

Today, many devices are controlled through Internet by HTTP protocol. Therefore, many embedded systems implement some network functionalities. The next example, demonstrating the advantages of the COSEDYRES method, will be the co-synthesis of the module implementing a simple web server. Function of this server may be specified by the

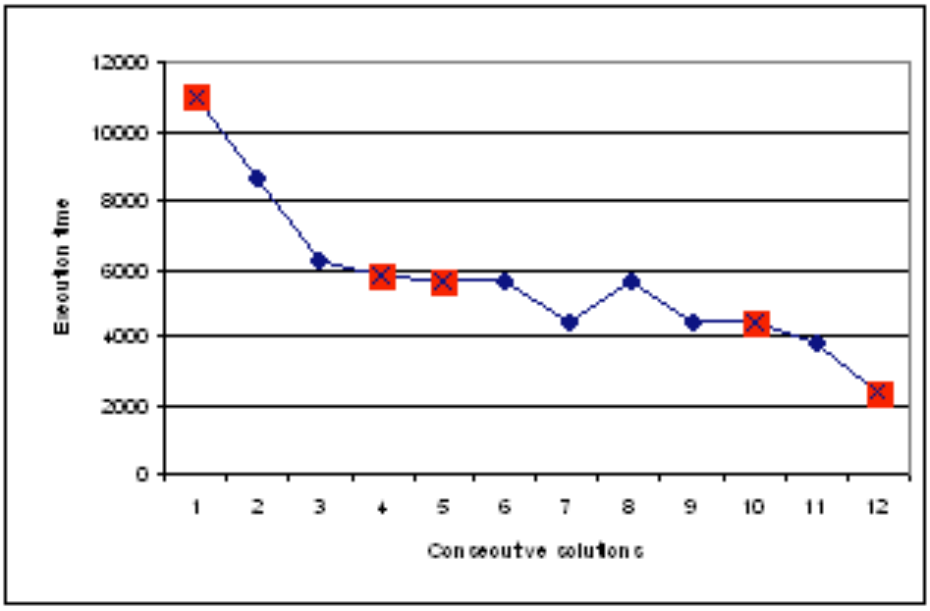


**Fig. (6).** Convergence chart for TT described by CTG.

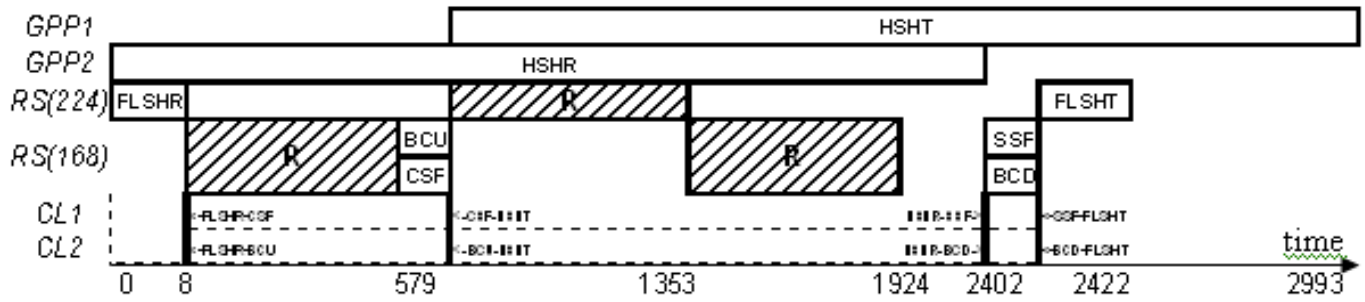


**Fig. (5).** Gantt chart for TT obtained as a result of COSEDYRES-CTG.

conditional task graph presented in Fig. (**7**). It consists of the following tasks:

- **GetReq**: process that waits on notifications on port 80. All requests are send to *ProcReq* and after emptying transceiver buffer the information is send to *Trans*.
- **ProcReq**: process that is activated after receiving information from *GetReq*. It reads data packages into buffer. After receiving a full HTTP request the information is send to *ProcGet* or *ProcPost*, depending on the request type. Since this process is the most complex, it is decomposed into *ProcReq1* and *ProcReq2*.
- **ProcGet**: processing the GET requests.
- **ProcPost**: processing the POST requests.
- **Trans**: sends consecutive parts of HTML files.
- **ManCon**: manages of connection status.

Tasks *Trans* and *ProcReq* (*ProcReq1, ProcReq2*) will never be executed in the same time, but depending on the *Dir* condition. If *Dir*=1 then after finishing *ProcReq*, task *ProcGet* or *ProcPost* will be executed, depending on the *Req* condition. Thus, in this CTG hierarchical conditions exist.

Assume that there is only one *GPP* module available with an area equal to 200 CLBs, and that the available communication channel transmits 30 B during 75 ns**.** Table **7** presents the library of hardware and software components. In the following experiments, the embedded web server was synthesized assuming the space constraint of the FPGA equals to **1000 CLBs**.

First, the system was implemented as a SOPC. The fastest SOPC architecture contains one *GPP* executing 3 tasks and four *VCs*. The Gantt chart for such SOPC system is given in Fig. (**8**). The characteristic of this system is the following: the total execution time **T=5150 µs** and the area S=968 CLBs.

Next, the COSEDYRES method was applied to obtain DRSOPC implementation, but the system was represented by the task graph (without conditional edges). In the initialization phase the following available sector sizes were generated: 56, 168, 280 and 336 CLBs. As the result of the cosynthesis, the architecture where all tasks were assigned to hardware was generated. It was possible due to dynamic reconfiguration. Scheduling of tasks for DRSOPC implementation, where system is represented by TG is illustrated in Fig. (**9**). For such system, characteristic is the following: **T=4026 µs**, S=996 CLBs. Increase in performance, in comparison to the SOPC implementation, equals 22%. Reconfigurations were partially performed in parallel with computations, thus the impact of the reconfiguration time on the decrease in the system performance was reduced.

Finally, the system specified by the conditional task graph was synthesized. A few pairs of tasks are mutually exclusive: {(*Trans*, *ProcReq1*), (*Trans*, *ProcReq2*), (*ProcGet*, *ProcPost*), (*Trans*, *ProcGet*), (*Trans*, *ProcPost*)}. COSEDYRES algorithm found an architecture with two reconfigurable sectors, with areas equal to 280 and 336 CLBs. Fig. (**10**) illustrates scheduling of tasks for DRSOPC system specified by the CTG graph. Tasks *ProcReq2* and *Trans* were assigned to the same *RS1* sector and scheduled parallel (they are executed depending on the *Dir* condition). Similarly tasks *ProcGet* and *ProcPost* were assigned to the same *RS2* and they are executed depending on the *Req* condition. Such tasks need not be allocated in the same time, but can be allocated by dynamic reconfiguration after the condition is evaluated. The characteristic of the architecture obtained using COSEDYRES method is: **T=3694 µs** and S=850 CLBs. Performance was increased by 8% in comparison with the previous solution and the area occupied by web server was decreased by 15%. Much better utilization of the space in the FPGA (smaller areas of sectors) caused that reconfiguration times of *RS*s were shorter, thus reducing the total execution time of the system.

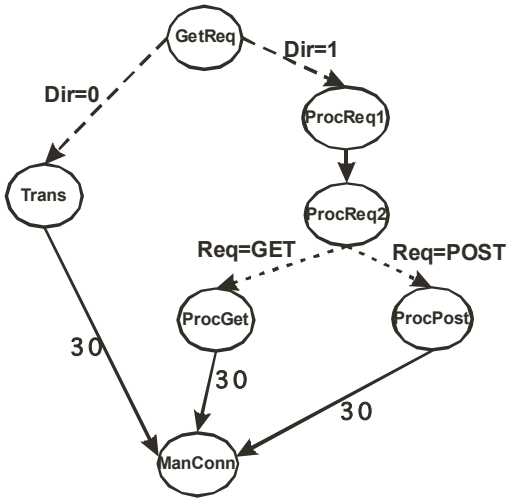


**Fig. (7).** CTG specification of the embedded web server.

Presented examples showed that the dynamic reconfiguration gives faster systems when reconfiguration tasks are properly scheduled and executed in parallel with computations. Moreover, considering mutually exclusive tasks in a system model brings additional possibilities of optimizations, in order to achieve faster dynamically reconfigurable systems (when MET tasks are allocated in the same sector).

**Table 7. Parameters of Tasks from the Embedded Web Server Specification**

| Task | SW | | HW | |
|---|---|---|---|---|
| | *t[μs]* | *S[B]* | *t[μs]* | *S[CLB]* |
| **GetReq** | 2000 | 600 | 400 | 250 |
| **ProcReq1** | 1200 | 1500 | 650 | 300 |
| **ProcReq2** | 2800 | 500 | 850 | 200 |
| **ProcGet** | 2500 | 1300 | 1000 | 300 |
| **ProcPost** | 1500 | 1200 | 300 | 50 |
| **Trans** | 500 | 400 | 150 | 150 |
| **ManConn** | 600 | 500 | 200 | 200 |

## CONCLUSIONS

In this paper the co-synthesis algorithm that targets at dynamically reconfigurable multiprocessor SOPC systems, with possibility of considering mutually exclusive tasks, was presented. The method is dedicated to the most popular partial reconfigurable Xilinx FPGAs. Reconfiguration of parts of the FPGA is controlled by the embedded processor. Hence, the whole system is placed in one FPGA. Dynamic reconfiguration enables different functionalities to be allocated in the same part of an FPGA. Due to implementation of more tasks in hardware, the overall performance is significantly higher, even if the reconfiguration may take some additional time. To the best of our knowledge, it is the first co-synthesis algorithm for multiprocessor SOPCs dealing with dynamically self-reconfigurable systems, and one of the first algorithms taking into consideration placement constraints for most popular modern FPGAs. This is also the first co-synthesis algorithm for dynamically reconfigurable SOPC systems that considers mutually exclusive tasks specified by the CTG graph. Such information let better utilize the space allocated to the designed system by assigning more task to hardware, thus speeding up the system. Moreover, the COSEDYRES algorithm has

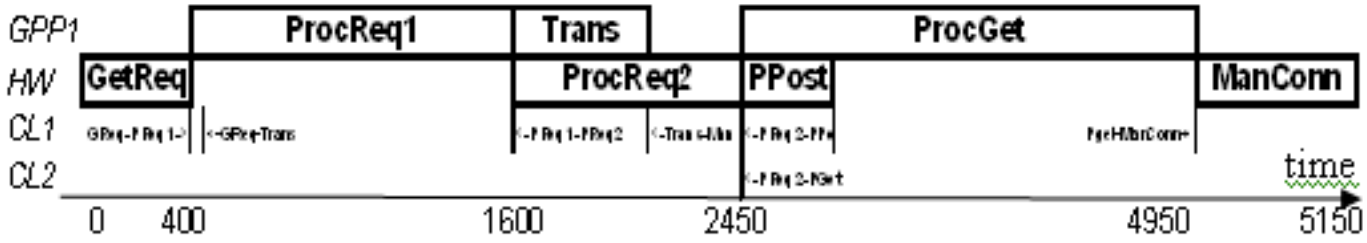


**Fig. (8).** Gantt chart of the web server implemented as SOPC.

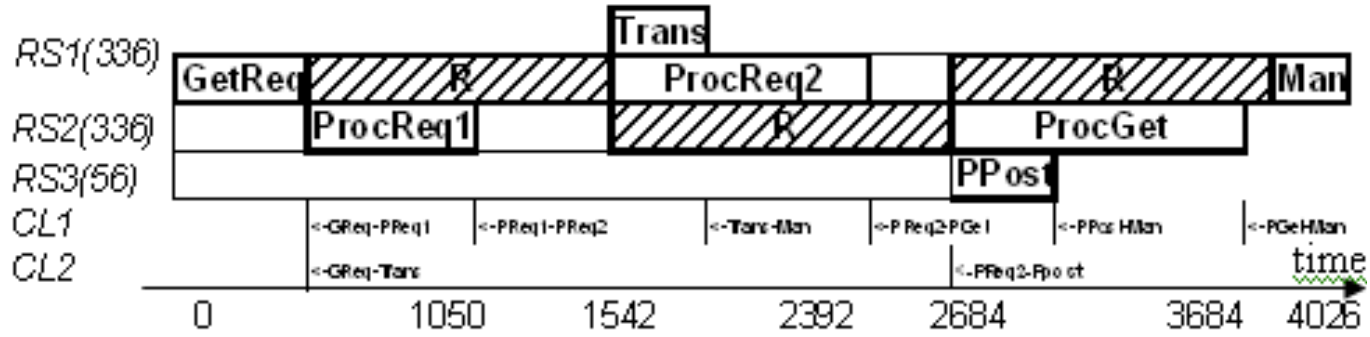


**Fig. (9).** Gantt chart of the web server represented by TG and implemented as DRSOPC.

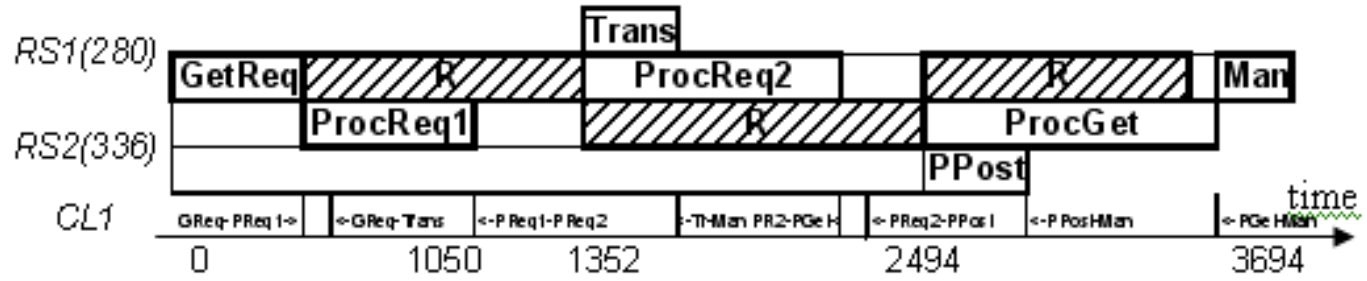


**Fig. (10).** Gantt chart of Web Server represented by CTG and implemented as DRSOPC.

a low computational complexity, and it has a capability of avoiding local maxima of the performance (that is the most common drawback of refinement methods).

## REFERENCES


[1] Xilinx Inc., "Two flows for partial reconfiguration: module based or difference based", *Xilinx Application Note XAPP290*, v.1.2, 2004.

[2] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software", *ACM Comput. Surv.*, vol. 34, no. 2, pp. 171-210, June 2002.

[3] S. Fekete, J. van der Veen, J. Angermeier, D. Göhringer, M. Majer, and J. Teich., "Scheduling and Communication-aware Mapping of HW-SW Modules for Dynamically and Partially Reconfigurable SoC Architectures", *Proc. of the Dynamically Reconfigurable Systems Workshop*, 2007.

[4] S. A. Khayam, S. A. Khan, and S. Sadiq, "A Generic Integer Programming Approach to Hardware/Software Codesign", *Proc. of IEEE International Multi Topic Conference IEEE INMIC 2001. Technology for the 21st Century*, pp.6-9, 2001.

[5] R. P. Dick and N. K. Jha, "CORDS: hardware-software co-synthesis of reconfigurable real-time distributed embedded systems", *Proc. ICCAD*, pp. 62-68, 1998.

[6] K. B. Chehida and M. Auguin, "HW/SW Partitioning Approach for Reconfigurable System Design", *Proc. CASES 2002*, pp. 247-251, 2002.

[7] A. Jhumka, S. Klaus, S. A. Huss, "A Dependability-Driven System-Level Design Approach for Embedded Systems", Proc. DATE'05, vol. 1, pp. 372-377, 2005.

[8] S. Deniziak, "Cost-Efficient Synthesis of Multiprocessor Heterogeneous Systems", *Control Cybern.*, vol. 33, no. 2, pp. 341-355, 2004.

[9] T.-Y. Yen and W. H. Wolf, "Sensitivity-Driven Co-Synthesis of Distributed Embedded Systems", *Proc. of International Symposium on System Synthesis*, pp. 4-9, 1995.

[10] K. S. Chatha, R. Vemuri, "Hardware-software codesign for dynamically reconfigurable architectures", *Proc. FPL*, pp. 175-184, 1999.

[11] S. Lee, S. Yoo, and K. Choi, "Reconfigurable SoC design with hierarchical FSM and synchronous dataflow model", *Proc. CODES*, pp. 199-204, 2002.

[12] S. Banerjee, E. Bozorgzadeh, and N. Dutt, "Physically-aware HW-SW partitioning for reconfigurable architectures with partial dynamic reconfiguration", *Proc. DAC*, pp. 335-340, 2005.

[13] Y. Li, T. Callahan, E. Darnell, R. Harr, U. Kurkure, and J. Stockwood, "Hardware-software co-design of embedded reconfigurable architectures", *Proc. DAC*, pp. 507-512, 2000.

[14] L. Shang, R. P. Dick, and N. K. Jha, "SLOPES: Hardware–Software Cosynthesis of Low-Power Real-Time Distributed Embedded Systems With Dynamically Reconfigurable FPGAs", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp. 508-526, 2007.

[15] J. Ou, S. B. Choi, V. K. Prasanna, „Energy-Efficient Hardware/Software Co-synthesis for a Class of Applications on Reconfigurable SoCs", *Int. J. Embedded Syst.,* vol. 1, no. 1/2, pp. 91-102, 2005.

[16] F. Ferrandi, M. D. Santabrogio, and D. Sciuto, "A Design Methodology for Dynamic Reconfiguration: The Coronte Architecture", *19th IEEE International Parallel and Distributed Processing Symposium* – Workshop 3, pp. 163-166, 2005.

[17] H. Kalte, D. Langen, E. Vonnahme, A. Brinkmann, and U. Ruckert, "Dynamically reconfigurable system-on-programmable-chip", *Proc. Euromicro PDP*, pp. 235-242, 2002.

[18] E. Carvalho, N. Calazans, E. Briao, and F. Moraes, "PaDReH – a framework for the design and implementation of dynamically and partially reconfigurable systems", *Proc. SBCCI*, pp.10-15, 2004.

[19] B. Blodget, P. J. Roxby, and E. Keller, "A Self-reconfiguring platform", *Proc. FPL*, pp.565-574, 2003.

[20] G. M. Megson, **"**Exploiting reconfigurability through high level synthesis.", *IEE Colloquium on Hardware-Software Cosynthesis for Reconfigurable Systems* (Digest No: 1996/036), pp. 5/1 - 5/4, 1996.

[21] I. M. Bland and G. M. Megson, "The Systolic Array Genetic Algorithm, An Example of Systolic Arrays as a Reconfigurable Design Methodology", *IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 260-261, 1998.

[22] B. Mei, P. Schaumont, and S. Vernalde, "A hardware-software partitioning and scheduling algorithm for dynamically reconfigurable embedded systems", *Proc. ProRisc Workshop on Ckts, Systems and Signal Processing*, 2000.

[23] Y. Qu, J.-P. Soininen, J. Nurmi, "A Parallel Configuration Model for Reducing the Run-time Reconfiguration Overhead", *Proc. DATE'06*, pp. 965-969, 2006.

[24] A. Daboli and P. Eles, "Scheduling Under Data and Control Dependencies for Heterogeneous Architectures", *Proc. of the International Conference on Computer Design*, pp. 602-608, 1998.

[25] Y. Xie and W. Wolf, "Allocation and Scheduling of Conditional Task Graph in Hardware/Software Co-synthesis", *Proc. DATE*, pp. 620-625, 2001.

[26] D. Wu, B. M. Al-Hashimi, and P. Eles, "Scheduling and mapping of conditional task graph for the synthesis of low power embedded systems", *IEE Proceedings Computers and Digital Techniques*, Vol. 150 Issue: 5 pp. 262-273, 2003.

[27] Y. Xie, L. Li, M. Kandemir, *et al*., "Reliability-aware co-synthesis for embedded systems", *J. VLSI Sig. Process. S.,* vol. 49, no. 1, pp. 87-99, 2007.

[28] S. Chakraborty, T. Erlebach, and L. Thiele, "On the complexity of scheduling conditional real-time code", *Algorithm. Data Structure.*, vol. 2125, pp. 38-49, 2001.

[29] W. Bossung, S. A. Huss, and S. Klaus, "High-level embedded system specifications based on process activation conditions", *J. VLSI Sig. Process. S.,* vol. 21, no. 3, pp. 277-291, 1999.

[30] J. Henkel, R. Ernst, "High-level estimation techniques for usage in hardware/software co-design", *Proc. Asia and South Pacific Automation Conference*, pp. 353-360, 1998.

[31] R. Czarnecki, S. Deniziak, and K. Sapiecha, "An iterative improvement co-synthesis algorithm for optimization of SOPC architecture with dynamically reconfigurable FPGAs", *Proc. EUROMICRO DSD*, pp. 443-446, 2003.

[32] R. Czarnecki and S. Deniziak, "Resource Constrained Co-synthesis of Self-reconfigurable SOPCs", *Proc. DDECS*, pp. 49-54, 2007.


Received: March 31, 2008 Revised: May 30, 2008 Accepted: June 2, 2008