This article has been published in the Dec. 2008 issue DAES Journal. Content is final as presented, with the exception of pagination.
The original publication is available at www.springerlink.com DOI 10.1007/s10617-008-9031-1
An International Journal Design Automation for Embedded Systems Vol. 12, No. 4, pp. 313-343, December 2008

CAD Tool for Hardware Software Co-synthesis of Heterogeneous Multiple Processor Embedded Architectures

Gul N. Khan and U. Ahmed
Electrical and Computer Engineering, Ryerson University
350 Victoria Street, ON M5B2K3 Canada
email: gnkhan@ee.ryerson.ca URL: http://www.ee.ryerson.ca/~gnkhan

Abstract: Hardware software co-synthesis process intends to determine an optimal architecture for an embedded application specified by a task graph or a specification language. In this paper, we present a co-synthesis approach targeting MPSoCs and distributed memory multiprocessor architectures for high performance embedded applications. Our co-synthesis approach produces pipelined multiprocessor architectures consisting of heterogeneous processing elements connected by a point-to-point communication structure. The co-synthesis process consists of four distinct phases; processing element selection for addition to the system, pipelined task allocation, scheduling and a regular interconnection topology mapping. Initially, an irregular topology is generated that is mapped to a regular architecture. Our co-synthesis methodology performs system partitioning and produces an irregular topology multiprocessor system. It also generates an optimal (or sub-optimal) regular topology architecture after considering some of the well-known regular topologies like mesh, hypercube, tree, etc. The co-synthesis method is demonstrated by exploring embedded architectures for MPEG encoder and artificially generated application task graphs representing complex embedded systems.

Keywords: Co-synthesis of Multiprocessor Architecture, Embedded Computer Design Tools, Hardware/Software Co-design, MPSoCs, Network on Chip.

1. INTRODUCTION

A number of application domains like multimedia, automotive, biomedical, networking and ambient intelligence technologies are computationally intensive and single processor based embedded systems can't provide all the functionality with desired timing and throughput. Such applications require multiprocessor systems or MPSoCs (Multiple Processor System on Chips). This research is motivated by the need to automate the design of high performance multiprocessor embedded systems. Automation of hardware-software (HW/SW) co-synthesis is driven by a number of factors including lower time-to-market, migration of software processes to hardware and vice versa and optimal generation of embedded architectures. HW/SW co-synthesis is an important component of co-design process and its flexibility can avoid early decisions. It allows designer to decide quite late in the design process about which technology CPU or dedicated hardware is to be employed. It also allows changing the implementation of processing modules at any stage of the design process and provides a number of benefits including short design-cycle and competitive systems. The main advantages of HW/SW co-synthesis are lower system design time and cost. Additionally, the handling of complex system design is made easier.

Single-processor embedded architecture consists of a CPU and one or more hardware modules. A number of initial research projects on HW/SW co-synthesis target single-processor architectures [1, 2]. A multiprocessor embedded architecture provides higher performance by distributing computation among CPUs and dedicated hardware modules. Several research groups have pursued multiprocessor embedded architectures [3]–[8]. SpecSyn employs a mixed (static/dynamic) performance estimation approach and it is difficult to capture dynamic changes of the execution time during design space exploration [3]. In POLIS, the target embedded architecture consists of CPUs combined with some DSPs and ASICs [4]. The timing analysis approach used in POLIS can capture much of the dynamic timing behavior as it uses a combination of high-level simulation and low-level estimation. However, it lacks precision in its analysis. Yen and Wolf analyze the interaction between different processes at the system level [5]. They start with an acyclic graph representing data dependencies among processes and calculate the worst-case execution time of the overall system by using partitioning and/or allocation information. Cai and Gajski have tackled heterogeneous multiprocessor architectures with multiple CPUs, coprocessors and other hardware modules communicating through multiple busses [6]. Other researchers have presented design flows for application-specific multiprocessor architectures using a point-to-point communication model [7, 8]. Most of these techniques targeting multiprocessor architectures lack generic aspects and can only tackle specific applications.

Various algorithms have been proposed for hardware/software co-synthesis but these algorithms are limited in the sense that they usually consider only shared bus architectures with a few general-purpose processors. Prakash and Parker presented their co-synthesis work targeting synthesis of application specific heterogeneous multiprocessor systems [9]. Their technique employs a formal mechanism by creating mathematical model for the constraints and objectives. The algorithm input is a data flow graph and it synthesized an arbitrary multiprocessor topology producing optimal results. However, their technique is slow and limited to only small applications due to mixed integer linear program based formulation. Heuristic approaches have also been utilized for co-synthesis of distributed memory architectures that build the solution step by step. COSYN is an example of constructive co-synthesis [10]. Most of the constructive approach based co-synthesis research has been concentrated on hierarchical systems [11]. An iterative co-synthesis approach has been employed to design generic system architectures by Wolf [12]. The algorithm targets irregular system topologies and can handle multiple CPUs and hardware (HW) modules. The algorithm separates selection and allocation processes, however, the addition of HW modules to the system may produce inefficient solutions. HW modules are almost always fully utilized and might never be removed from the system. Moreover, the algorithm does not pipeline the tasks and therefore cannot satisfy high throughput requirements.

Multiple processor task allocation and scheduling is an important component of co-synthesis and known to be NP-complete [13]. Therefore heuristics are employed to allocate and schedule tasks on multiple processors [14]. Higher throughput requirements demand pipelined systems [15]. Executing only the critical application sections in hardware cannot satisfy the increased throughput constraint and processing elements (PEs) need to be

arranged into concurrent execution stages. Bakshi and Gajski have presented a co-synthesis technique involving pipelining and scheduling of multiple PEs [16]. This technique is limited, as it cannot trade-off between hardware and software implementation for a given task. The hardware cost of adding a CPU is not considered and inter-task communication time is ignored. Another co-synthesis approach by Chatha and Vemuri involves pipelining [17]. Their algorithm generates architectures for data-flow oriented applications and it is centered around a branch and bound partitioner. It performs pipelining by heuristically selecting a data dependency and then creating a pipeline stage. Since task allocation and pipelining are performed without any interaction and without considering pipeline period, the algorithm can result in implementations with redundant pipeline stages. The method is limited to single CPU systems and executes with exponential time.

MPSoCs are composed of multiple processors for various applications including mobile terminals, set-top boxes, gaming and video processors. MPSoC is becoming a popular and prevalent design style to achieve low time-to-market, to simplify system verification and to provide flexibility in the design of complex and high performance platforms. HW/SW co-synthesis of MPSoC architecture is a major challenge and the average number of processors per MPSoC is growing [18]. For example, the computational load of ambient intelligence may require a large number of processors. MPSoC performance is determined by the ability of on-chip inter-PE communication network that can accommodate the communication needs of heterogeneous processing elements. It is claimed that Network-on-Chip (NoC) based systems are economically feasible for several product variants [19]. Betrozzi and others have proposed a synthesis tool 'Netchip' for designing domain specific on-chip network for MPSoCs [20]. It provides design support for regular topologies suited to interconnection structures of homogeneous as well as heterogeneous PEs. Netchip assumes that the system has been already partitioned, application tasks are mapped on to cores, and core graph is available to the topology mapping tool. Its main functions include topology mapping, topology selection and generation. SUNMAP performs the topology mapping and selection [21] while XpipeCompiler generates the topology [22]. However, our approach performs system partitioning as well as optimal system topology mapping and generation. HW/SW co-synthesis technique along with a regular topology selection methodology described in this paper, provides a tool for designing multiprocessor embedded architectures as well as MPSoCs. Our co-synthesis framework allows the comparison of regular and irregular topologies of the target system by co-synthesis and producing the application-specific irregular topology architecture. Then a sub-optimal regular topology architecture is generated, which is based on the irregular topology produced in the initial phase of cosynthesis.

2. COSYNTHESIS OF MULTIPROCESSOR EMBEDDED ARCHITECTURES

2.1. Overview

The co-synthesis approach described in this paper, targets multiprocessor embedded architectures consisting of heterogeneous processing elements (PEs) connected by a network structure. The main components of the

embedded multiprocessor architecture are PEs, local memories for program and data as well as inter-PE communication structure. The PEs can either be processors/DSPs (software-PEs) or dedicated hardware modules (hardware-PEs) that are linked to the interconnection network. We disassociate PEs from the interconnection network by decomposing the communication interface into two parts, one is specific to PE and the other is generic depending on the number and type of communication links and protocol. The interconnection structure being proposed is also suitable for MPSoCs [20]. In these systems, a master PE node is the initiator of communication while the slave node responds to the master. In addition to local memory, each PE node is considered as having a wormhole switch/router. The hardware area of the switch/router assumed to be part of the PE node. A software or hardware PE accesses the interconnection network using the network interface, which performs the data conversion into a format supported by the network.

We assume best effort, packet switched inter-PE communication systems, using wormhole switching. Static, look-up table based routing is assumed, with route tables embedded in the header flits of each packet (source routing). The PE node level switches have the sole role of transferring data through the network. Each switch/router is composed of fixed number of ports. Switches/routers are assumed to be capable of performing arbitration and data forwarding independently for every output port. Each output port can have a dedicated arbitration and forwarding unit. Round-robin based arbitration can be easily realized by a small amount of hardware. From the target architecture point of view, in addition to local memory, hardware area of switch and network interface is included in the PE node area. It is also assumed that the data buffering required for pipelining is provided by the local memory at each PE node.

The HW/SW co-synthesis approach described here requires the embedded application to be specified as an acyclic task graph encompassing the system requirements. It is also assumed that application task graph as well as the target system is of course grain nature where each PE node has a program/data memory and switch/router. The co-synthesis methodology also needs a library of PEs with profiling data in the form of worst-case execution time of each task for the available PEs and their hardware area. Overall system hardware area and execution time constraints are fed to the co-synthesis algorithm. The system execution time constraint is specified in terms of pipeline period. The co-synthesis tool can be fed with the system hardware area constraint only. The co-synthesis algorithm explores the design space following the hardware area constraint for a range of pipelined periods. The co-synthesis process begins with a single-CPU based all software solution of the application and adds PEs to the target system incrementally. Pipeline stages are created during the allocation and scheduling process. The overhead of task scheduling on software-PEs is ignored. The mathematical formulation described in this section can be amended to include the overhead of task scheduling on the same software-PE.

The co-synthesis methodology presented in this paper employs a two-phase system generation mechanism, where an irregular topology is generated first and then a regular architecture is determined. In the first phase, an application specific architecture is produced satisfying the system hardware area and pipelined period constraints. In the 2nd phase, directly connected PEs are mapped to a regular topology while meeting the HW

area and timing constraints as depicted in Fig. 1. The shaded part of Fig. 1 provides the details of 1st phase, where PEs are added to the target system one by one followed by pipelined task allocation and scheduling. Details of each step of this phase are provided in the next sub-sections.

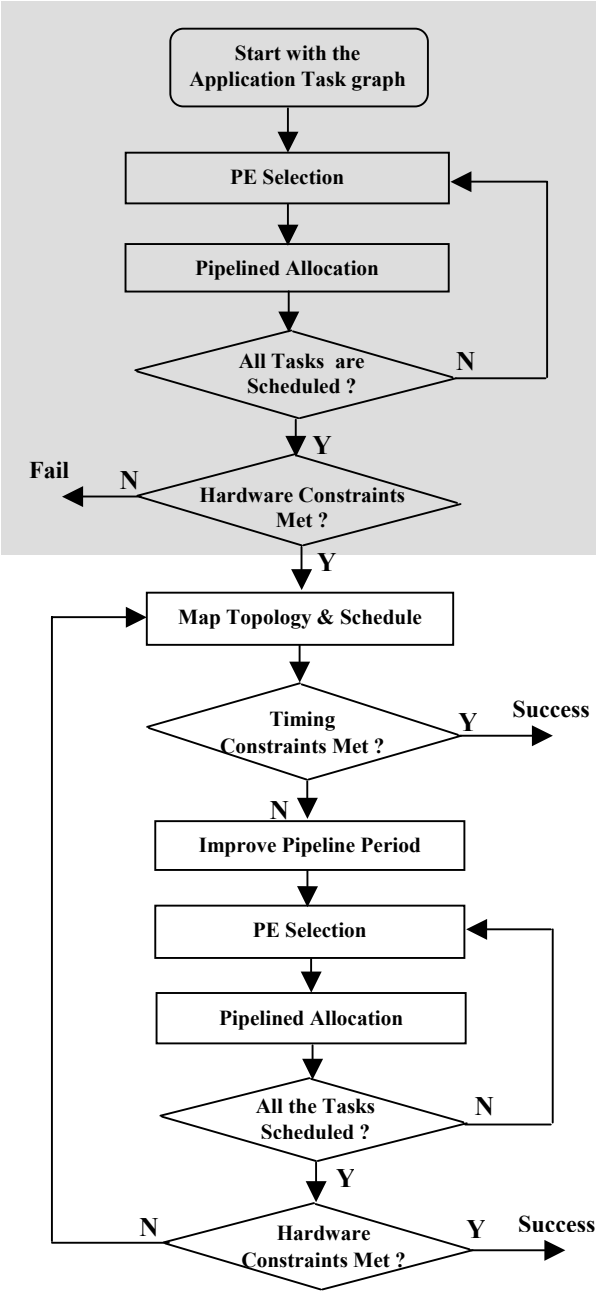


Fig. 1. System Topology Generation and Co-synthesis

2.2. Processing Element Selection for Addition to the System

Initially all the tasks are scheduled on a software PE to meet the timing constraints and if this is not possible, additional PEs are added to the system. This is an important and critical stage of our co-synthesis approach

where a software (SW) or hardware (HW) PE is selected for addition to the target system. A heuristic approach is employed to select the PE. The heuristic is based on the system performance gain and HW area of the PE being selected for addition to the system. Once an in-efficient PE is added to the system, it is only considered for removal when the target system could not meet the timing constraints even after adding as many PEs as the number of application tasks. The formulation of PE selection for adding to the system is mathematically rigorous and it chooses the heuristically best PE in terms of overall improvement in the system performance i.e. lower pipeline period and target system HW. The PE selection parameter consists of two factors namely the performance and HW area improvements. To keep this step simple, the effect of data communication delay on the performance when the tasks are allocated to different PEs is not considered. Following variables are defined in order to describe these factors.

- T_{SW} denotes the set of tasks that do not have a dedicated hardware resource (SW-PE) in the current system.
- T_{HW} denotes the set of tasks that have a dedicated hardware resource (HW-PE) in the current system.
- PE_{SW} and PE_{HW} denote the SW-PEs (CPUs) and HW-PEs respectively.
- $\mathrm{SYS}_{PE_{SW}}$ and $\mathrm{SYS}_{PE_{HW}}$ are the sets of software and hardware PEs in the current system.

The PE selection process begins with the calculation of cumulative software and hardware PEs execution times as given in equations (1) and (2). These equations only consider the SW and HW PEs in the current target system and calculate the cumulative software execution time factor (*CUM_SW_TIME*) that is the sum of execution times of all the tasks that are only executable on SW-PEs in the target system. Similarly, cumulative hardware execution time factor (*CUM_HW_TIME*) is the sum of the execution times of all tasks that are executable on HW-PEs in the current target system. These factors treat the tasks separately in terms of application tasks that are executable on HW-PEs and the tasks that are only executable by SW-PEs.

$$CUM\_SW\_TIME = \sum_{T_i} \sum_{PE_j} ExecTime(T_i, PE_j) \quad (1)$$

where $T_i \in T_{SW}$ and $PE_j \in \mathrm{SYS}_{PE_{SW}}$.
$ExecTime(T_i, PE_j)$ is the time of executing task T_i on a SW-PE, PE_j.

$$CUM\_HW\_TIME = \sum_{T_m} \sum_{PE_n} ExecTime(T_m, PE_n) \quad (2)$$

where $T_m \in T_{HW}$ and $PE_n \in \mathrm{SYS}_{PE_{HW}}$.
$ExecTime(T_m, PE_n)$ is the time of executing task T_m on a HW-PE, PE_n.

To explain the PE selection, task allocation and scheduling, consider a simple task graph with five tasks (T1, T2, … ,T5) shown in Fig. 2. The inter-task communication data is provided on the edges of the graph. There are two SW-PEs (PE1 and PE3) that can only execute all the tasks and one HW-PE (PE2) that can only execute task T3. Table 1 provides the library data in terms of PE hardware area and task execution times. The hardware area and pipeline time period constraints used for the example are 3200 and 30 units respectively. The co-synthesis process starts with one SW-PE solution and chooses PE1 due to its lower HW area. By using the PE selection

mechanism, another PE1 is selected for addition to the system. Consider the current target system with two SW-PEs (two instances of PE1) in the target system. At this stage, the system is not meeting the pipelined period constraint of 30 time units and another PE will be added to the target system.

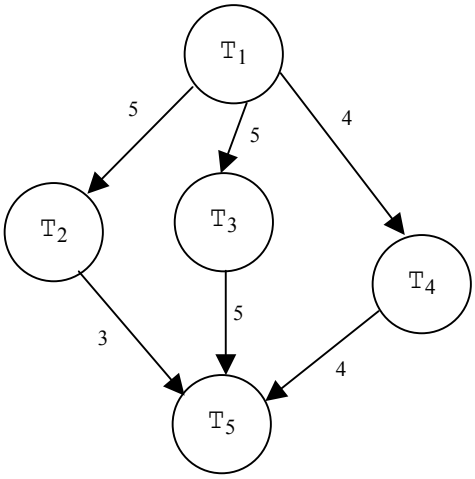


Fig. 2. Example Task graph

Table 1. PE Execution Time and Area Information

Tasks	PE1	PE2	PE3
T1	15	-	10
T2	22	-	20
T3	10	2	10
T4	10	-	5
T5	15	-	15

PE	Type	Area
PE1	SW	1000
PE2	HW	15
PE3	SW	1500

The cumulative software and hardware times of equations (1) and (2) are calculated as following.

CUM_SW_Time = 62 + 62 = 124 (There are 2 instances of PE1 in the current target system)

CUM_HW_Time = 0 (There is no HW-PE in the current system)

The process selection methodology also defines a hardware improvement factor $HW\_IMP(PE_{NEW})$ formulated by equation (3), which is the envisaged performance improvement obtained by adding a new hardware-PE (PE_{NEW}) to the system. This factor is used for calculating the system execution time factor (SYS_{CURR_TIME}) when a hardware PE, (PE_{NEW}) is considered for addition to the system as presented in equation (5).

$$HW\_IMP(PE_{NEW}) = \frac{\sum_{T_i}\sum_{PE_j} ExecTime(T_i, PE_j)}{SYS_{PE_{SW}}} - \sum_{T_i} ExecTime(T_i, PE_{NEW}) \quad (3)$$

where T_i is a set of tasks that can execute on PE_{NEW}.

$PE_j \in \text{SYS}_{PE_{SW}}$ and PE_j are software PEs that have been included to the system.

By using the hardware improvement factor and cumulative hardware and software execution time factors of equations (1) and (2), the system execution time factors before (SYS_{PREV_TIME}) and after (SYS_{CURR_TIME}) the addition of a new PE are estimated by using equations (4) and (5).

$$SYS_{PREV\_TIME} = \frac{CUM\_SW\_TIME}{(SYS_{PE_{SW}})^2} + CUM\_HW\_TIME \quad (4)$$

where $SYS_{PE_{SW}}$ represents software PEs in the current target system.

SYS_{PREV_TIME} is the time related performance parameter of the current target system when a new PE is not added to the target system where SYS_{CURR_TIME} is the time related performance parameter of the target system after the addition of a new PE to the target system. For HW-PE selection, SYS_{CURR_TIME} is calculated by subtracting the hardware PE improvement factor (HW_IMP) from the SYS_{PREV_TIME}. It is worth mentioning that SYS_{PREV_TIME} used in the 2nd part of equation (5) include the cumulative software and hardware execution time factors of the target system before adding PE_{NEW}.

$$SYS_{CURR\_TIME} = \begin{cases} \dfrac{CUM\_SW\_TIME + \sum_{T_i} ExecTime(T_i, PE_{NEW})}{(SYS_{PE_{SW}} + 1)^2} & \text{if } PE_{NEW} \in PE_{SW} \\ SYS_{PREV\_TIME} - HW\_IMP(PE_{NEW}) & \text{if } PE_{NEW} \in PE_{HW} \end{cases} \quad (5)$$

where T_i is a set of tasks that can be executed execute on PE_{NEW}.
$SYS_{PE_{SW}}$ are the number of SW PEs in the target system.

In the case of example task graph, the hardware improvement factor formulation presented in equation (3) is only calculated for PE2 and $HW\_IMP(PE2) = (10+10)/2 - 2 = 10 - 2 = 8$.

The system execution time of the target system (SYS_{PREV_TIME}) formulated by equation (4) while the system execution times after the addition of a new PE, (SYS_{CURR_TIME}) calculated by equation (5) are given below. SYS_{CURR_TIME} is calculated for all the available PEs, however, only one PE with the largest improvement is selected for addition to the target system.

$SYS_{PREV\_TIME} = 124/4 + 0 = 31$

SYS_{CURR_TIME} (PE1) $= (124 + 62)/9 = 20.67$

SYS_{CURR_TIME} (PE2) $= 31 - 8 = 23$

SYS_{CURR_TIME} (PE3) $= (124 + 50)/9 = 19.33$

To select a PE for addition to the system, the system performance improvement factor (PE_{PERF_IMPR}) has been formulated by using the improved system execution time (in terms of $SYS_{CURR\_TIME}/SYS_{PREV\_TIME}$) as presented in equation (6). The cost of adding a PE (PE_{NEW}) is also considered by employing the hardware (HW) area factor (PE_{AREA_FACTOR}) of equation (7) that takes into account the HW area associated with a particular PE.

$$PE_{PERF\_IMPR} = 1 - [SYS_{CURR\_TIME} / SYS_{PREV\_TIME}] \quad (6)$$

$$PE_{AREA\_FACTOR} = 1 - [Area(PE_{NEW}) / MAX\_AREA\ ] \quad (7)$$

where $Area(PE_{NEW})$ is the HW area of the new PE to be considered for addition to the system.
MAX_AREA is the HW area of an available PE with maximum hardware cost.

The PE selection parameter (PE_{SELECT}) is defined as a weighted sum of performance improvement factor (PE_{PERF_IMPR}) and HW area cost improvement factors (PE_{AREA_FACTOR}), which is given in equation (8). PE_{SELECT} for all the available PEs are considered to select the most suitable PE for addition to the system.

$$PE_{SELECT} = k \times PE_{PERF\_IMPR} + (1 - k) \times PE_{AREA\_FACTOR} \quad (8)$$

where k is a user defined area-performance trade-off factor and $k \in [0, 1]$.

For the example task graph of Fig. 2, PE selection parameters are calculated for each PE (PE1, PE2 and PE3) by using SYS_{PREV_TIME}, SYS_{CURR_TIME} and the HW area of these PEs as given below. In this particular case, we use an equal weighting for the area and performance improvement factors (i.e. $k = 0.5$). However, user has the option of utilizing a different weight.

PE_{SELECT}(PE1, PE2, PE3) = (0.33, 0.62, 0.19)

Our methodology chooses PE2 for addition to the system from the above values of PE selection parameters as PE2 has the maximum improvement parameter. The updated target system now contains three PEs (PE1, PE1 and PE2) whose HW is 2015 area units. When all the tasks are scheduled by following the pipelined task allocation and scheduling strategy described in the next sub-section, system pipeline period is still higher than the timing constraint of 30 time units. The same process of PE selection is repeated to add another PE to the system. Finally a four PE target system with three instances of PE1 and a PE2 is generated by this phase of the co-synthesis method that meets both the area constraint of 3200 and pipeline period constraint of 30 time units.

This phase of the co-synthesis, which adds a new PE to the target systems, terminates when all the tasks are scheduled successfully within the pipeline period constraint. Otherwise, it continues to add more PEs until the HW area constraint is violated. In this phase of co-synthesis, there is a chance that slow and inefficient PEs become part of the system. It may also result in a system that could not meet the required timing constraints. Another step is needed that enables the co-synthesis method to remove slow and inefficient software-PE that has become the system bottleneck. This situation arises when total number of tasks allocated to software PEs become equal to software PEs (i.e. $T_{SW} = SYS_{PE_{SW}}$). When it happens, a SW PE cannot be added to the system. One needs to replace the slow and inefficient SW-PEs with faster and efficient SW-PEs. An inefficient and a slow SW-PE is selected for removal by employing equation (9).

$$PE_{SLOW} = \underset{\forall PE \in SYS_{PE\,SW}}{MAX} \left[\sum_{T_i \in T_{SW}} ExecTime(T_i, PE_j) \right] \quad (9)$$

Equation (9) chooses the SW-PE (PE_{SLOW}) that takes maximum time to execute the software tasks of the application. If HW area constraint is being violated then PE HW area criterion can also be used to choose the

inefficient PE. Once such a PE is removed from the system, it is kept on a 'slow-PE' list whose PEs are not considered in the subsequent PE selection process.

2.3. Pipelined Task Allocation and Scheduling

PEs are selected and tasks are allocated to PEs iteratively, in the PE allocation and pipelining step. Pipeline stages are created during allocation, which results in systems that do not have any redundant pipeline stage. After adding a new PE, tasks are allocated along with pipelining of the task execution. Pipelined allocation is the process of assigning start time to each task satisfying the condition given in equation (10).

$$0 \leq StartTime(T_i) \leq [\ StartTime(T_i) + ExecTime(T_i, PE_j) + \text{MAX}(CommTime(PE_j, PE_k))\ ] \leq T_{PERIOD} \quad (10)$$

where $StartTime(T_i)$ is the time when task T_i starts execution.
$ExecTime(T_i, PE_j)$ is the execution time of task T_i on its allocated PE_j.
$CommTime(PE_j, PE_k)$ is the data transfer time from the predecessor PEs (PE_k) to PE_j.
PE_k are all the PEs where parent tasks of T_i are allocated.
T_{PERIOD} is system pipeline period constraint.

First step of pipelined task allocation is to assign a priority to each task. The priority criterion assigns high priority to those tasks, which are on the critical path (from the execution point of view) as formulated by equation (11).

$$Priority(T_i) = \text{MIN}(ExecTime(T_i)) + \text{MAX}(Priority(T_j)) \quad (11)$$

where $ExecTime(T_i)$ is the execution time of task T_i for all the PEs in the target system.
T_j task is a successor of task T_i.

Priority is assigned by starting at the tail of the task graph and setting the priority of each task as the sum of its minimum execution time (on any PE) and maximum priority of its successors. Similar priority measure has been extensively employed for multiprocessor scheduling as part of parallel computing research [13, 14, 15]. At this stage of task allocation and scheduling, a priority list of tasks is created. The prioritized list of tasks is sorted in terms of task priorities determined by equation (11). Pipeline allocation process initially finds start and finish time of each task for all the PEs. Start time for a task is defined in terms of the earliest start time (*EarliestStartTime*) and idle time of a PE (*PEIdleTime*). *PEIdleTime* is the time when a PE becomes available (idle) after executing all the tasks allocated to it. Start and finish times formulation for a task T_i when it is allocated to PE_j is represented by the following set of equations (12).

$$\left.\begin{aligned} StartTime(T_i, PE_j) &= \text{MAX}(\ EarliestStartTime(T_i), PEIdleTime(PE_j)\ ) \\ FinishTime(T_i, PE_j) &= StartTime(T_i, PE_j) + CommTime(T_i, PE_j) + ExecTime(T_i, PE_j) \\ EarliestStartTime(T_i) &= \text{MAX}(FinishTime(\ \text{PRED}(T_i)\ )\ ) \end{aligned}\right\} \quad (12)$$

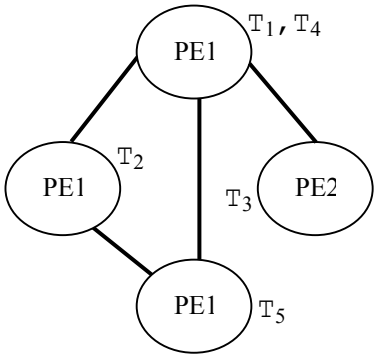
where $PRED(T_i)$ is the set of all the predecessors of task T_i.
$PEIdleTime(PE_j) = FinishTime$(Last task on PE_j).
$CommTime(T_i, PE_j)$ is the data transfer time to/from PE_j with respect to task T_i.

Task scheduling mechanism that is based on equations (11) and (12) will be discussed further in section 3.2 where a pseudo code for scheduling is also provided. The overhead of inter-task data communication is ignored

for those tasks when they are allocated to the same PE. The data communication time for a task T_i is only defined when T_i is allocated to PE_j. It is obtained by adding data transfer times of all the predecessors of T_i that are not allocated to PE_j as formulated in equation (13). Finish time of the highest priority ready task is calculated for all the PEs that has been added to the system. A task is allocated to the PE with the earliest finish time. When the earliest finish time violates the pipeline period constraint, a new pipeline stage is created and the task is added to the new pipeline stage.

$$CommTime(T_i, PE_j) = \sum_{\forall PRED(T_i) \notin PE_j} DataTransfer(PRED(T_i)) \quad (13)$$

Communicating PEs are directly connected for the irregular architecture generated by this co-synthesis phase. The system architecture generated for the example task graph of Fig. 2 is shown in Fig. 3(a). The architecture has four PEs containing three software-PEs and one hardware PE. Task allocation and scheduling of PEs for the example task graph is performed by following the pipelined allocation and scheduling process described above. The pipelined schedule of the target system, which has three instances of SW-PE (PE1) and one HW-PE (PE2) is also given below in Fig. 3(b). The execution is divided into three pipeline stages where tasks T1, T3 and T4 execute in 1st stage, T2 executes in the 2nd stage and T5 executes in the 3rd stage.



(a)

Task	Allocated to	Start Time	Finish Time	Pipeline Stage
T1	PE1 (a)	0	15	1
T2	PE1 (b)	0	27	2
T3	PE2	15	22	1
T4	PE1 (a)	15	25	1
T5	PE1 (c)	0	20	3

(b)

Fig. 3. (a) Irregular Topology with Allocation (b) Scheduling of the Example Task Graph

3. REGULAR INTERCONNECTION STRUCTURE

The multiprocessor system generated in previous section meets all the timing and HW area constraints, however the PEs are connected by an irregular topology. Irregular topology structures lack support of standard message passing systems, fault-tolerance, etc [23]. Regular interconnection structures have been proposed and employed for various MPSoC platforms [18]. Researchers have recently pursued scaleable and regular interconnection structures known as Network on Chips (NoCs), which have particular features that can enable the realization of complex MPSoCs [24]. Regular topologies like mesh, tree and hypercube support fault-tolerance and proven deadlock free inter-PE routing algorithms exist. If a communication link or a PE fails, data can be routed to other PEs through alternate paths. Moreover, off-the-shelf IPs are available for regular

topologies to expedite the system realization. A number of regular interconnection structures have been proposed for on-chip architectures of MPSoCs. For example, SoCBUS is a 2D mesh based interconnection structure that relies on a simple layout while its switch hardware is independent of interconnection network [25]. SPIN is another fat-tree based regular topology employed for on-chip interconnection structures of MPSoCs [26]. Octagon based regular topology has also been advocated for on-chip communication structures [27]. Hypercube topology embedded architectures has also been targeted recently for co-synthesis [28].

The final phase of our co-synthesis approach is the regular topology mapping and it is depicted in the 2nd half (un-shaded part) of Fig. 1. Topology selection follows another task scheduling phase. If all the tasks complete their execution within the pipeline period constraint, co-synthesis algorithm terminates successfully. Otherwise, the algorithm can proceed to reduce the pipeline period. Alternatively, other regular topologies can also be tried for successful topology mapping and scheduling within the pipeline period constraint. At this stage, our co-synthesis method selects an economical topology suitable for the embedded application after considering various regular topology structures.

3.1. Topology Generation and Addressing

Initially, a topology template of nodes is created and addresses are assigned to the nodes. All the regular topologies are defined using the same template. The topology template is presented by a graph (G_T) consisting of nodes (V_T) that are connected by edges (E_T) as presented in equation (14).

$$G_T = \{ V_T, E_T \} \tag{14}$$

where $V_T = \{v_0, v_1, ..., v_i\}$ is a set of nodes.

$E_T = \{e_i = (v_x, v_y) \mid v_x, v_y \in V_T\}$ is the set of edges.

Each node is assigned a unique address '*Addr(v_i)'* and the number of topology nodes are equal to the number of PEs in the system. A set of neighbours is defined for each node as given in equation (15).

$$N_{v_i} = \{ n \mid (v_i, n) \in E_T \} \tag{15}$$

The topology mapping process for 2D mesh and hypercube topologies is explained here. In the case of mesh, the template is generated by arranging the nodes in a grid. A topology mapping and scheduling algorithm is presented in a preliminary method presented earlier [29]. Number of rows and columns in a mesh are determined by finding the total number of PEs in the system (*SYS_PE*), which is determined by adding software and hardware PEs of the system given in equation (16).

$$SYS\_PE = |SYS_{PE_{SW}}| + |SYS_{PE_{HW}}| \tag{16}$$

Number of rows and columns of a 2D mesh are determined by employing the equation set (17).

$$\left.\begin{aligned} MAX\_ROWS &= \left\lceil \sqrt{SYS\_PE} \right\rceil \\ MAX\_COLS &= \left\lceil \left(SYS\_PE \div MAX\_ROWS \right) \right\rceil \\ ADDR\_BITS &= \left\lceil \log_2 (MAX\_ROWS) \right\rceil \end{aligned}\right\} \quad (17)$$

Row address bits (*ADDR_BITS*) are determined by the equation set (17) and column address bits are also determined in the same way. Nodes are added row-wise, while addresses are assigned to each node by concatenating row and column addresses. For example, node 6 (0110) of Fig. 4 has the address bits that correspond to 2nd row and 3rd column i.e. (01_10). After address assignment, the neighbors are added if they exist. The pseudo code provided below illustrates this process for 2D mesh topology.

```
node = 0 ;
FOR row = 0 to MAX_ROWS - 1
FOR col = 0 to MAX_COLS - 1 {  // Assign address to
   // node by concatenating addresses row and column
   Addr(v_node) = (row << ADDR_BITS) | col ;
   | N_Vnode | = 0 ;  // set neighbour count to '0'
   IF (  Exist(UPPER_NEIGHBOUR)  )
      AddNeighbour(UPPER);  // add upper neighbour
   IF (  Exist(LOWER_NEIGHBOUR)  )
      AddNeighbour(LOWER);  // add lower neighbour
   IF (  Exist(LEFT_NEIGHBOUR)  )
      AddNeighbour(LEFT);      // add left neighbour
   IF (  Exist(RIGHT_NEIGHBOUR)  )
      AddNeighbour(RIGHT);    // add right neighbour
   node++;
   IF (  node == SYS_PE  )
      RETURN;  // return when all nodes have been added
} // FOR
```

Fig. 4 depicts an eight-node 2D-mesh topology generated after application of the above pseudo code. The number of hops needed to transfer data from one node to another node is the sum of absolute differences of corresponding row and column addresses. For instance, data transfer from node '0' (00_00) to node '9' (10_01) would require three hops as differences in row and column addresses are two and one respectively.

The topology generation process for hypercube topology is also presented for completeness. Addresses for hypercube range from 0 to *SYS_PE*. After assigning addresses, the neighbors for a node are added while the degree of hypercube identifies the number of neighbors for each node where $DEGREE = \lceil \log_2(SYS\_PE) \rceil$. The adjacent nodes in hypercube have only 1-bit difference between their addresses. This feature is employed to assign neighbors for each node as given in the following pseudo code.

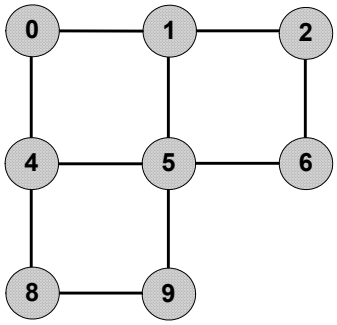


Fig. 4. Eight-Node 2D-Mesh Topology with Address Assignment

```
FOR node = 0 to SYS_PE -1 {
  Addr(v_node) = node;
  | N_Vnode | = 0 ;  // reset neighbor count to '0'
} // FOR
 //Set neighbors for each node
FOR i = 0 to SYS_PE -1
 FOR j = 0 to SYS_PE -1 {
  IF (AddrBitDiff(v_i, v_j, DEGREE) == 1)
  // add v_j as v_i's neighbor
 AddNeighbor(v_i, v_j);
 } // FOR
```

Fig. 5 shows an 8-node hypercube where the above procedure is used to assign addresses to its node. The number of address bit difference determines the inter-node path length in terms of number of hops among the nodes. For example topology node 1 (001) and 6 (110) has a bit difference of three indicates a three-hop length path between nodes 1 and 6. A similar methodology and specific topology properties have been used to generate the templates for tree and other regular topologies.

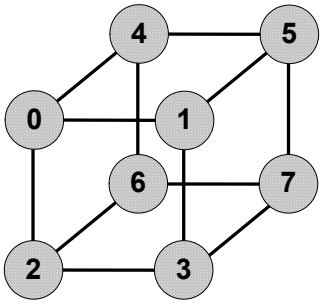


Fig. 5. Eight-Node Hypercube with Address Assignment

At this stage of mapping, a blank topology template has been created, however, PEs are not allocated to the topology nodes. PEs are mapped to the topology nodes such that the total communication delays among all the system tasks are minimal. It maximizes data traffic among neighbor (directly connected) nodes. A topology

node is allocated if a PE has been assigned to it. Consequently allocated neighbours of a node v_i are the adjacent nodes that have been assigned a PE and the neighbours are denoted as $N_{v_i}^A \subseteq N_{v_i}$. Neighbor node traffic is formulated in equation (18).

$$NeighborTraffic(PE_i, v_j) = \sum_{\forall v_a \in N_{vj}^A} CommTime(PE_i, PE(v_a)) \quad (18)$$

where $PE(v_a)$ is the PE_j that is mapped to the topology node v_a.
$CommTime(PE_i, PE_j)$ is the time taken to transfer application data between PE_i and PE_j.

During the allocation of PEs to the topology nodes, unallocated neighbors of the current node are stored in a FIFO list and PEs are assigned first to the neighbours of the allocated nodes. The pseudo code given below illustrates the allocation of PEs to the topology nodes irrespective of the topology (mesh, hypercube or tree). The start node is assumed to be node 0 of the topology, however it can be selected differently depending on the position of the node in the irregular topology architecture.

```
PE0 → V0 ;  //Assign first PE to start node
PE0 = ALLOCATED;  //Add all neighbours of v0 to FIF0
AddtoFIFO(NV0);
        //Processing all un-allocated nodes
WHILE ( Vu = GetfromFIFO() ) {   // Calculate neighbour
        // traffic for current  node  and all unallocated  PEs
  FOR ALL UNALLOCATED PEs: PEi
  FOR j = 0 to | NVu |
       Calculate-NeighbourTraffic(PEi, Vj) ;
       // Get PE with maximum neighbour traffic
  PEA = GetMaxTrafficPE();
  PEA = Vu ;    //Assign PEA to Vu
  PEA = ALLOCATED;
      //Add unallocated neighbours of Vu to FIFO
 AddtoFIFO (NVu)
} //Continue till FIFO is empty
```

An efficient topology is selected by calculating the overhead for each topology with respect to the irregular topology. The topology overhead is defined as the extra time spent on transferring data between PEs due to indirect and multi-hop communication paths of a regular topology. It is determined by calculating the number of hops between all the communicating PE pairs and the amount of data to be transferred via these hops as formulated by equation (19). Number of hops, *HOPS* is determined from the addresses of source and destination nodes. For a 2D-mesh the number of hops needed is the sum of address bit differences of rows and columns of the source and destination nodes. In the case of hypercube, the number of hops between any two nodes is their address bit differences.

$$Overhead_{TOP} = \sum_{i=0}^{(SYS_{PE}-1)} \sum_{j=0}^{N_{PEi}} [HOPS_{TOP}(PE_i, PE_j) - 1] \times CommTime(PE_i, PE_j) \quad (19)$$

where $HOPS_{TOP}(PE_i, PE_j)$ is the path length (hops) between PE_i and PE_j

CommTime(PE_i, PE_j) provides the time required to transfer data between PE_i and PE_j

3.2. Task Scheduling

The application tasks are rescheduled by following list scheduling after the PEs are mapped to a regular topology. It is the same technique for pipelined task allocation and scheduling that has been described in section 2. Pipeline stages are not modified and each task is scheduled in the same pipeline stage and allocated to the same PE where it was scheduled in the first phase of co-synthesis. Scheduling is performed by selecting the highest priority task whose predecessors are already scheduled. Earliest start time and PE idle time are determined for the selected task by following the equation set (12). Data communication time is also calculated by taking into account the location of the PE node in the topology for the current task and PEs of its predecessor tasks. Finish time of the task is calculated by using its execution time and the communication delays. If the task completes its execution within the pipeline period, it is removed from the ready task list. If the task cannot complete within the pipeline period, more PEs are added to the system to reduce the pipeline period as described next. A pseudo code given below illustrates the scheduling process employed at this stage of co-synthesis.

```
READY_TASK = Task0; // Initiailze the ready task list
// Continue while there is a task to be scheduled
WHILE ( TaskR = GetReadyTask() ) {
    PE = GetTaskPE(TaskR); // Get PE where TaskR is allocated
    // Get time when all parents of TaskR finish execution, it is '0' for a new pipeline stage.
    Eearliest_Start_Time = GetEarliestStartTime(TaskR);
    // Time when PE finishes executing tasks allocated before 'TaskR'
    PE_IDLE_TIME = GetPEIdleTime(PE);
    // Time to transfer the data from parent tasks including
    // the delay due to extra hops in the regular topology
    COMM_TIME = GetDataXfrTime (PE, TaskR);
    // Time taken by 'TaskR' to complete its execution on 'PE'
    EXEC_TIME = GetTaskExecTime(PE, TaskR);
    // Time when 'TaskR' finishes its execution on 'PE'
    FINISH_TIME =  GetTaskFinishTime( TaskR, MAX(Eearliest_Start_Time,
            PE_IDLE_TIME), COMM_TIME, EXEC_TIME );
    IF (FINISH_TIME > TPERIOD) RETURN ; //Add a new PE or Pipeline Stage
    UpdateReadyTasks();
}  // while
```

3.3. Pipeline Period Reduction

Communication delays introduced by the topology mapping can terminate the scheduling phase unsuccessfully. In that case, additional PEs are added to the system so that tasks can complete their execution within the pipeline period constraint. Pipeline period can be reduced by the maximum violation time; however other tasks may not violate the pipeline period with the same amount and this may redundantly add more PEs in the system. Moreover, all pipeline stages do not have the same time period as tasks in some pipeline stages complete earlier than the pipeline period. Therefore reducing the period by maximum violation time can result in

expensive system. We employ an iterative period reduction mechanism where period is reduced by a smaller amount every time the scheduling phase fails. The reduced pipeline period is determined by employing the topology overhead ($Overhead_{TOP}$). The reduced pipeline period can also be related to the number of missing communication links in the regular topology as compared to the irregular topology. For constants K1 and K2, the pipeline time period can be estimated by using equation (20). The pipeline period can also be reduced by a constant factor of 10% to 50% instead of employing the equation given below.

$$T_{Period}\,(Reduced) = T_{Period} - (\mathrm{K1} \times Overhead_{TOP})\,(\mathrm{K2} \times Links_{M}) \quad (20)$$

where $Overhead_{TOP}$ is associated with the selected topology.

$Links_M$ is the number of missing communication links for the regular topology.

The application task graph is re-cosynthesized by employing the reduced pipeline period to generate a new irregular topology by following the first phase of the cosynthesis methodology. The new irregular topology is mapped to the regular interconnection structure meeting the pipelined period constraints as described in this section. When all the tasks are scheduled within the timing constraint, the algorithm terminates successfully otherwise pipeline period is further reduced and the same process is repeated. In case, any of the regular topologies does not satisfy the hardware area and pipeline period constraints, the irregular architecture generated in the first phase can also be considered for system implementation.

4. Assumptions, Limitations and Complexity of Co-syntheis Methodology

The inter-PE communication structure, we are assuming in our co-synthesis methodology has also been advocated for Network-on-Chip [19]. This communication structure is also suitable for MPSoCs, where communication occurs between master and slave interface of the PE nodes. A PE node contains a simple wormhole switch/router with equal number of input and output ports and its hardware area is assumed to be part of the node. The PE node also contains a local memory addressing the buffering requirements for pipelining and communication. Software and hardware PEs access the communication network using a Network Interface (NI), which perform packetization and de-packetization. It is assumed that a PE will make use of two network interfaces, the master and slave.

We have assumed packet switched communication with static wormhole routing where route tables are embedded in the header flits of each packet. These features ensure that the inter-PE communication system components are simple, and switches/router will require small hardware area. Wormhole switching ensures that the buffering requirement of each switch is small, where HW space is at a premium [30]. In addition, no virtual channel support is assumed, and deadlock-free operation is ensured through the topology and route selection. The switches/routers have the sole role of transferring data through the network. Each switch is composed of a fixed number of ports, where each port consists of an input port and corresponding output port. The point-to-point links between PEs are composed of two unidirectional channels, which together make up a link. Switches are assumed to be capable of performing arbitration and data forwarding independently for every output port. It

means that each output port has its own dedicated arbitration and forwarding unit. The arbitration method impacts the perceived performance of each PE node in the system. However, we are not addressing these features in our co-synthesis framework and it will be considered in future system generation tools.

The irregular topology generated by the first phase of co-synthesis is unique in the sense that the PEs requiring any application data transfer are directly connected. In this way, there is no communication-link conflicts for the 1st phase of co-synthesis as there are dedicated links between communicating PEs. Moreover, inter-task communication delays don't have any significant effect. These delays play a significant role in the 2nd phase of co-synthesis when a regular topology is mapped and there may be indirect connection (multiple hop) between some communicating PEs.

The co-synthesis approach describes in the previous sections employs a greedy heuristic to explore the design space of target system architecture. The co-synthesis begins with a single SW-PE system and adds PEs to the system iteratively to meet the system timing and HW area constraints. The order and type of PE selection for addition to the target system can affect the overall system architecture. PE selection strategy is critical for the generation of an efficient and economical system. An unbefitting PE addition to the system can produce an expensive system that may not meet the timing and HW area constraints. The PE selection mechanism employed by the co-synthesis methodology is rigorous as it selects the heuristically best available PE in terms of performance and cost. However, there is a remote possibility that this process can lead to unfeasible solution. To cater for this rare situation, a remedy is incorporated that can remove any slow and inefficient SW-PE from the target system as discussed in section 2.2. However, when to remove a slow PE from the target system is a significant point that has not been considered. For the presented co-synthesis approach, slow and inefficient SW-PEs are removed when number of PEs in the target system become equal to the number of application tasks and no further PE can be added to the system. There is a possibility that if inefficient PEs are removed earlier, a proficient architecture can be produced. A user-defined parameter, k weighs the cost and performance factors for the target system. This parameter is employed in the PE selection step of co-synthesis and user can vary and tune this weighing factor, k in equation (8) to produce feasible solution for an application task graph.

Another important heuristic employed during regular topology mapping is the pipeline period reduction in the 2nd phase of co-synthesis. The pipeline period reduction is not required for most of the real and artificially generated applications as presented in the experimental results. The need for pipeline period reduction arises when the communication delay among the PEs hosting the communicating tasks results in a task finish time that exceed the pipeline period constraint. When this condition occurs, the co-synthesis approach can opt for a number of alternatives. Two of the options that has been adopted in our co-synthesis approach is to reduce the pipeline period and re-cosynthesize the application or look for another regular topology that meets the pipeline period constraint. Another option that has not been employed in our co-synthesis methodology is to re-map the regular topology in such a way that the pipeline time period constraint is not violated. Even one can opt for the

last option and employ a complex topology mapping mechanism in the first place, however, it will not guarantee to meet the pipeline period constraints.

The complexity of the co-synthesis methodology can be evaluated by considering the individual steps of its both phases. The main parameters to be considered are the number of tasks (n) of the application task graph, number of different types of PEs (ma) in the PE library and number of PEs (mb) in the current target system. The co-synthesis starts with one SW-PE in the target system and PE selection step adds PEs to the target system one by one. The target system can have at the most 'n' number of PEs where $mb \leq n$. The PE selection parameter (PE_{SELECT}) is calculated for 'ma' different types of PEs and its complexity will be in the order of O(ma). The PE selection computation for each PE considered for addition to the system can be roughly estimated from equations (1) to (5), which is in the order of O($n \times mb$). The co-synthesis methodology separates the PE selection step from task allocation and scheduling. Task allocation and scheduling involves the priority assignment for 'n' tasks and its order of computation is at the most O($n \times mb$). Similarly, the start and finish time calculations for all the tasks can be approximated in the order of O($n \times mp$). Where 'mp' is the average number of PEs in the system that execute the predecessor tasks for each application task. In this way, the complexity of the first phase of co-synthesis for an 'n' node application task graph, which involves most of the computation can be approximated by equation (21).

$$\text{Complexity for the } 1^{st} \text{ phase of co-synthesis} = O(ma \times [2(n \times mb) + (n \times mp)]) \quad (21)$$

The 2nd phase of co-synthesis has three main steps including regular topology template generation, mapping of irregular architecture to regular topology and rescheduling to meet the pipelined period constraint. The complexity of topology template generation depends on the number of PEs in the system, which are equal to 'mb'. In the case of topology mapping, neighbor traffic is calculated for each PE in the target system for its allocation to neighbor nodes of the topology and its order of computation can be estimated as O($mb \times mb$). It may be adjusted for different regular topologies as these topologies have different number of neighbors. The rescheduling in the 2nd phase is not as complex as of 1st phase because task allocation performed in the 1st phase is not disturbed. The additional delay of communication due to multiple hops for the regular topology is added to task finish time. Therefore, for an 'n' task application task graph, the order of computation for rescheduling can be approximated as O($n \times mp$). As before, mp is the average number of PEs in the system where predecessor tasks are allocated for each application task (n). It is also worth mentioning that 'mp' is much smaller than the total number of PEs (mb) in the target system. Pipeline period reduction option in the 2nd phase of co-synthesis is an expensive option and it has been rarely used for the applications presented next. Our algorithm first looks for alternative regular topologies that do not require any pipeline period reduction.

5. EXPERIMENTAL RESULTS

5.1. MPEG Encoder System Generation

MPEG encoder application is used to test and verify our co-synthesis methodology. MPEG video encoding standard employs block-based motion compensation to reduce temporal redundancies and DCT (discrete cosine transform) based compression for spatial redundancies [31]. It produces three types of coded frames known as I-frames (intra-coded frames), P-frames (predictively-coded frames) and B-frames (bi-directional predictively coded frames). MPEG encoder is specified as a data-flow task graph shown in Fig. 6. It is a coarse-grain task graph consisting of 21 nodes with each node representing a block of computation. Numbers at the edges indicate the amount of data transfer among tasks in bytes for each frame of the video sequence. The video sequence is assumed to be in RGB format where 'Initialize' task performs initialization and maintains a state machine to determine the coding type (I, P or B) required for the current frame. 'YC_bC_r Conversion' task converts the current image block and reference images (B and P frames) from RGB to YC_bC_r format. 'Sub-Sample' task performs sub-sampling of colour difference components (C_b and C_r) for the block of current image. 'Split Fwd Ref Image' and 'Split Bwd Ref Image' tasks split the forward and backward reference images into four overlapping regions to perform motion vector search over these regions concurrently. FS1-FS4 and BS1-BS4 tasks perform the motion vector search over forward and backward reference images respectively. 'Fwd Motion Vector' and 'Bwd Motion Vector' tasks select the best forward and backward motion vectors respectively. 'Interpolate' task interpolates the forward and backward motion vectors for bi-directionally coded frames. DCT task calculates discrete cosine transform for an image block or motion-predication error block. 'Quantize' task performs quantization and 'DCAC Coding' task codes the dc and ac components of quantized discrete cosine transformed block. 'Entropy Encode' task performs Huffman encoding and 'Finalize' task packs the Huffman-coded symbols into a compressed bit stream.

The co-synthesis approach assumes that execution time of task for each PE is available. For HW-PEs, this information can be obtained manually or by using other techniques [32]. Software execution time for these tasks is calculated by executing each task for two different variants of Nios-II CPU, one standard Nios-II PE and the other with dedicated hardware multiplier. Motion vector search, DCT, quantization and DC/AC Coding tasks are also implemented as dedicated hardware modules. Each of these tasks has been implemented in Verilog and synthesized for Altera's Stratix FPGA. Execution time is acquired in terms of clock cycles required to complete the task. The hardware area cost associated with each PE (SW as well as HW PEs) is the number of logic elements (LEs) needed to implement it on an Altera Stratix FPGA. Table 2a lists the hardware area of PEs in terms of LEs, while worst execution times for all the tasks for each PE are listed in Table 2b. The execution times are listed in terms of clock cycles where SW-PEs (Nios-II CPU) as well as HW-PEs execute at 50MHz clock. These values of HW area in LEs and execution time in clock cycles are the actual and real values obtained by implementing all the tasks both in either in software, hardware or in both.

MPEG encoder task graph along with task execution times and HW area (for each PE) is fed to the co-synthesis algorithm. A range of constraints is also used to obtain a number of target systems. The co-synthesis algorithm produces a regular system topology consisting of heterogeneous PEs and a pipelined schedule for the tasks. Main parameters of the output consist of HW area (cost) of the system, number of PEs, system pipeline period and number of pipeline stages created by the co-synthesis methodology. The algorithm is executed for time period constraints varying from $6.5 \times 10^6 \Rightarrow 172 \times 10^6$ clock cycles. The constraints for HW area ranges from 4000 to 38000 logic elements. The lower hardware area constraint can be estimated by assuming that the task graph can be implemented on one SW PE, and in this particular case it may be a Nios-II CPU having 3662 logic elements.

TABLE 2 (a) PEs Library Info for MPEG Encoder

Name	Description	Type	Cost (area)
NIOS_MUL	Nios CPU with Multiplier	SW	4065
NIOS	Nios CPU	SW	3662
FWD/BWD MVS _ENG	Motion Vector Search Core	HW	615
DCT_ENG	DCT Core	HW	1008
QUANT_ENG	Quantization Core	HW	712
DC/AC_ENG	DC AC Coding Core	HW	453

TABLE 2 (b) Task Execution Times for MPEG Encoder

Task	Execution Time (Cycles)		
	Software NIOS_MUL	Software NIOS	Hardware
Initialize	1194178	1245890	-
YCbCr-Conversion	6141804	11261292	-
SubSample	122539	199147	-
Split Fwd Ref Image	1338842	1338842	-
Split Bwd Ref Image	1338842	1338842	-
FS1-FS4	19983314	20025208	131524
BS1-BS4	19983314	20025208	131524
Fwd Motion Vector	52161	90945	-
Bwd Motion Vector	52161	90945	-
Interpolate	149770	304906	-
DCT	377600	775136	936
Quantize	145537	260677	468
DCAC Coding	620181	620181	1214
Entropy Encoding	164918	242688	-
Finalize	256904	334876	-

The target system results produced by the co-synthesis algorithm are given in Table 3. Additional PEs are integrated into the target-system as the time constraint becomes tighter. The pipeline stages also increase as the system pipeline time-period decreases. The target system that meets the tightest constraint (6.5×10^6 cycles) consists of seven SW-PEs and eleven HW-PEs and tasks execute in five pipeline stages. On the other extreme, system corresponding to the slowest timing requirement is around 26 times slower and it consists of one software-PE (NIOS_MUL). Another important characteristic of the resulting systems is the interconnection of PEs in a regular topology. Table 4 provides the characteristic of the system for some representative test cases. It

also illustrates the overhead involved in arranging the PEs to a particular topology as well as the number of missing communication links in each topology as compared to the irregular topology. There is no extra PE needed to map the system to the regular topology. Fig. 7 illustrates the design space exploration corresponding to various test cases.

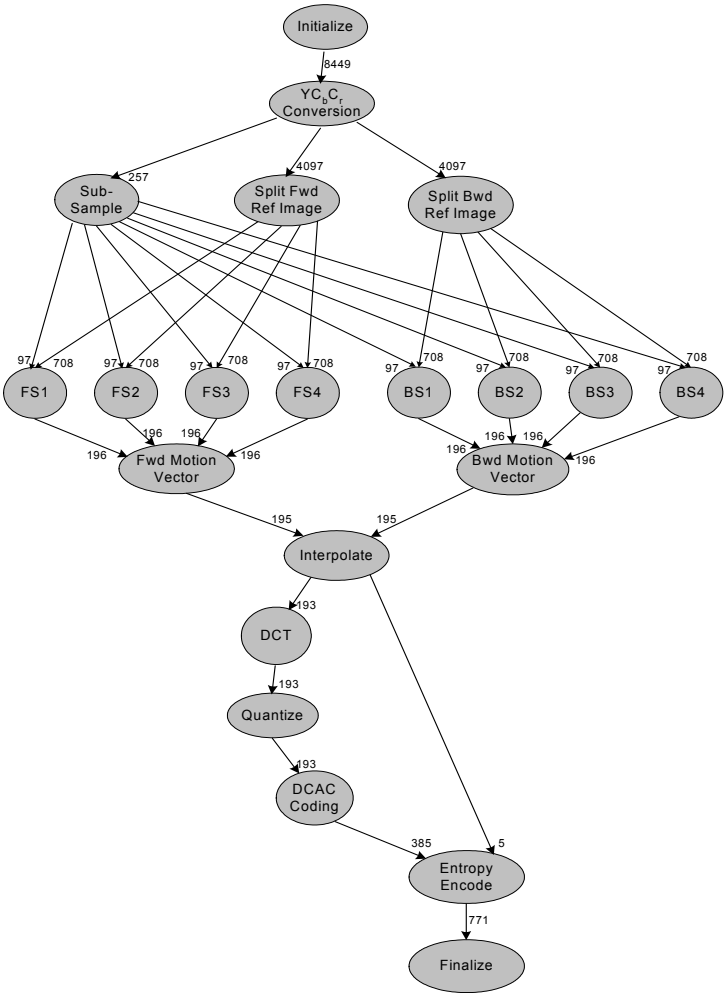


Fig. 6. MPEG Encoder Task Graph

It can be observed from the target system results that as the system timing constraints become tighter, overhead and the number of missing links ($Links_M$) increase. For MPEG encoder, no extra PE was required and our co-synthesis methodology produces various topologies for different constraints. Mesh topology has the lowest overhead for test case 1, and target system consists of 18 PEs and five pipeline stages. Fig. 8 depicts an irregular interconnection structure of PEs before they are mapped onto a regular topology. The regular topology and corresponding schedule map for each task is depicted in Fig. 9. It provides the scheduling of all the tasks along with the details of task allocation. The vertical axis of Fig. 9b lists the MPEG encoder tasks along with their allocated PEs in the form of *task_name(PE_name)*. Figures 10 and 11 provide similar details for the test case 4, where PEs are arranged in a hypercube topology. Similarly, Figures 12 and 13 provide details for test case 5, where a quad-tree topology is selected due to lower overhead.

TABLE 3 Target Systems Generated for MPEG-Encoder

Test Case	Constraints		System Results					
	Time x10^6	Hardware Area x10^3	Time x10^6	Hardware Area x10^3	PEs	Pipeline Stages	SW PEs	HW PEs
1	7	38	6.42	36.5	18	5	7	11
2	8	17	7.46	16.3	11	3	3	8
3	9	17	7.59	16.3	11	3	3	8
4	10	17	8.81	16.3	11	3	3	8
5	25	15	21.5	14.7	9	4	3	6
6	40	15	40	14.6	7	4	3	4
7	45	13	41.5	12.8	6	4	3	3
8	55	13	53.9	12.2	5	3	3	2
9	61	12	60.4	11.6	4	4	3	1
10	63	12	61.4	11.4	3	3	3	0
11	95	8.0	90.5	7.73	2	2	2	0
12	172	4.2	171.8	4.07	1	1	1	0

TABLE 4 Topology Information for MPEG-Encoder

Test Case	Time x10^6 Constraints	Tree Topology		Mesh Topology		Hypercube		Selected Topology
		$Overhead_{TOP}$	$Links_M$	$Overhead_{TOP}$	$Links_M$	$Overhead_{TOP}$	$Links_M$	
1	7	24051	27	13037	18	14696	24	MESH
2	8	6201	15	7978	19	5394	15	H-CUBE
3	9	6201	15	7978	19	5394	15	H-CUBE
4	10	5946	16	9681	16	4017	13	H-CUBE
5	25	2034	9	7458	14	2647	9	TREE
6	40	2329	10	4746	12	2942	10	TREE
7	45	936	5	3163	7	1549	6	TREE
8	55	1880	6	2710	4	2295	5	TREE
9	61	2490	4	2905	3	2905	3	TREE
10	63	999	2	999	2	999	2	Any

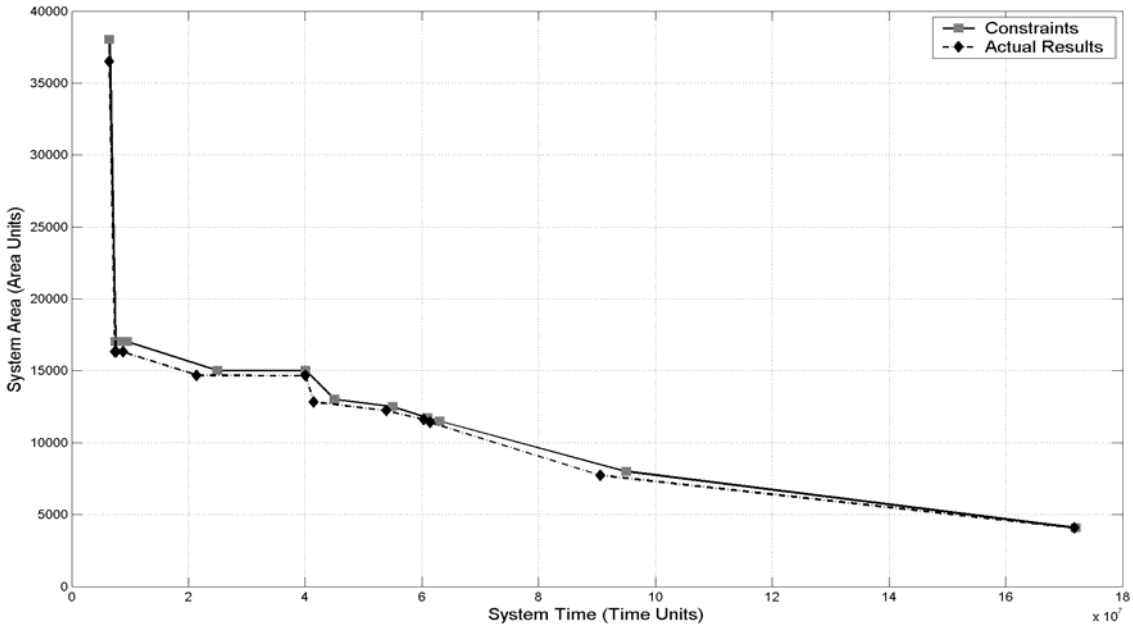


Fig. 7. Design Space Exploration for MPEG Encoder

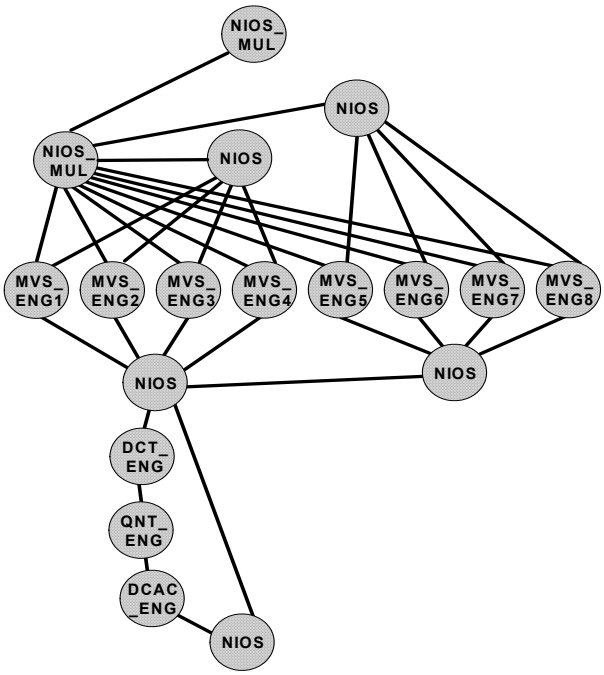
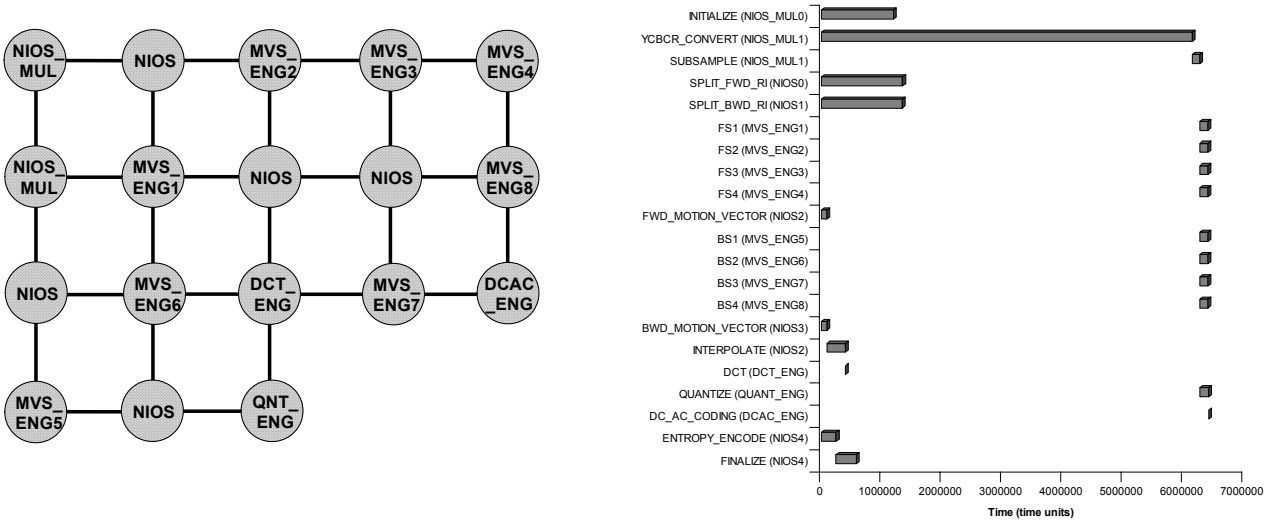


Fig. 8. Irregular PE Topology (Test case 1)



(a) (b)

Fig. 9. (a) MPEG Encoder PEs Arranged as Mesh (b) PE Schedule for the Mesh

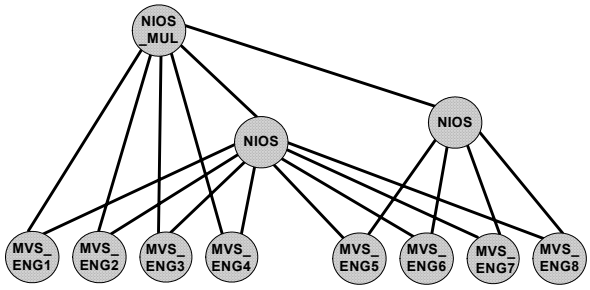
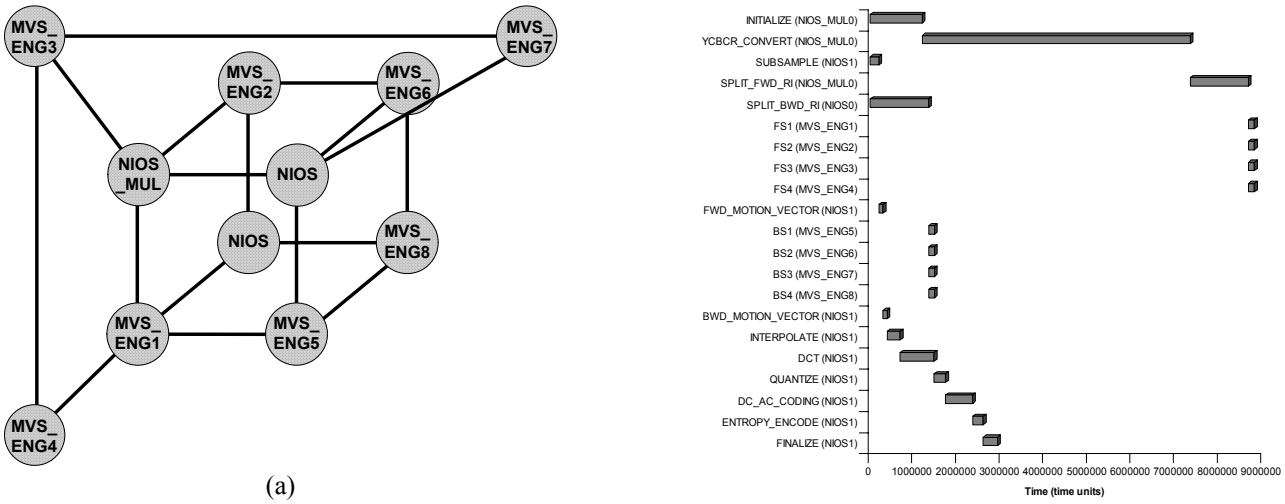


Fig. 10. Irregular PE Topology (Test case 4)



(a) (b)

Fig. 11 (a) MPEG Encoder PEs arranged as Hypercube (b) PE Schedule for the Hypercube

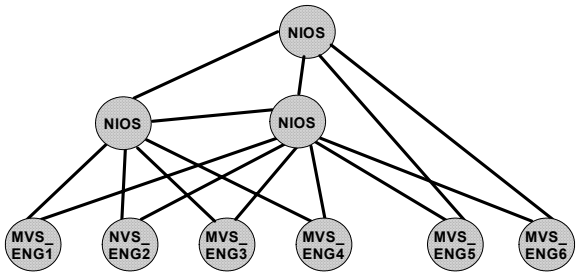
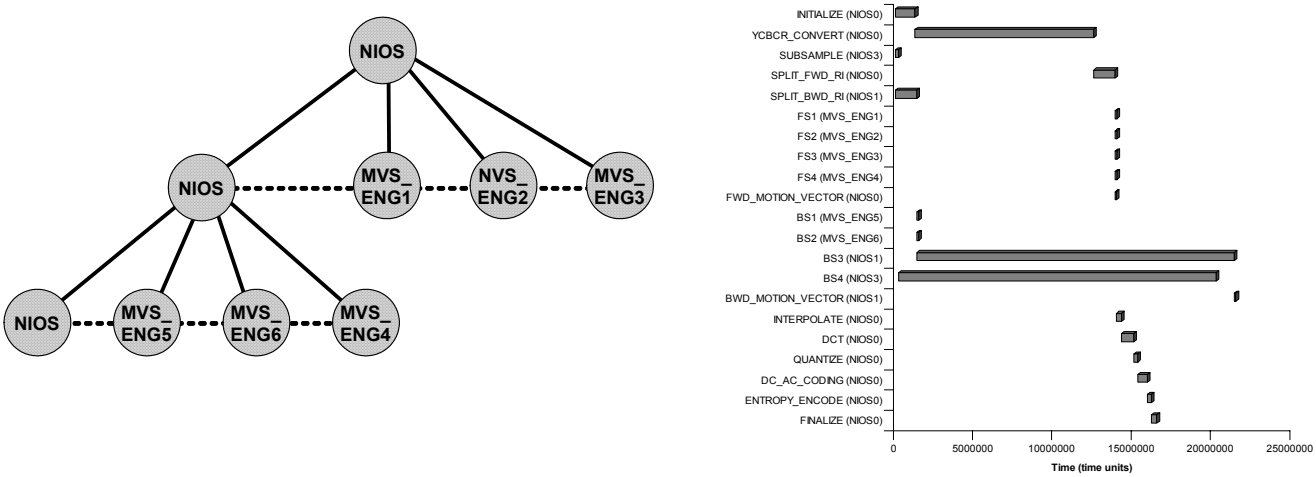


Fig. 12. Irregular PE Topology (Test case 5)



(a) (b)

Fig. 13 (a) MPEG Encoder PEs Arranged as Quad-Tree (b) PE Schedule for the Quad-Tree

We have conducted some additional experiments to compare the performance of our co-synthesis approach with the iterative co-synthesis technique proposed by Wolf [12]. Wolf's co-synthesis method is implemented to design the same MPEG encoder system. The hardware area constraints are relaxed for the Wolf's approach by keeping the same timing constraints. The MPEG encoder system generated by Wolf's approach is compared

with the system generated by our methodology and the results are presented in Table 5 for both approaches. Our co-synthesis approach generates regular topology based efficient systems for tighter timing constraints due to pipelined task allocation and scheduling. Moreover, it produces regular topology architectures in contrast to irregular structures generated by Wolf's approach. Generally, irregular topology architecture for the same target system has lower hardware area as compared to regular structures generated by our methodology. In spite of regular topology systems generated, our method outperforms the Wolf's approach for large scale multiple PE systems (> 8 PEs). Regular topology structures can also use a number of off-the-shelf IPs for efficient and fault-tolerant inter-PE communication structures. Our co-synthesis method also outperforms for large scale embedded systems consisting of 50⇒400 tasks in practical co-synthesis execution times as presented in the following section.

TABLE 5 Comparative Systems Generated by Wolf's Co-synthesis Approach

No.	**Constraints**		**Wolf's Co-synthesis Approach**		**Our Co-synthesis Approach**	
	Time x10^6	**HW Area x10^3**	**PEs**	**HW Area x10^3**	**PEs**	**HW Area x10^3**
1	7	42	19	41.6	18	36.5
2	8	20	12	19.96	11	16.3
3	10	17	11	16.3	11	16.3
4	25	15	10	14.7	9	14.7
5	40	15	7	14.6	7	14.6
6	45	13	6	12.8	6	12.8
7	55	13	6	12.8	5	12.2
8	61	12	4	11.6	4	11.6
9	63	12	3	11.4	3	11.4

5.2. Co-synthesis of Artificially Generated Task Graphs

A number of experiments are also conducted by performing co-synthesis on random task graphs that provide further insight about the regular topology overhead as well as the benefits of pipelined task allocation. These graphs are generated by randomly varying the number of predecessors and successors for each task, depth of the task graph and amount of data transferred among tasks. An in-house tool is used for generating these task graphs. Successors and predecessors for a task are varied from 2 to 20. Number of PEs available for the system are also randomly selected and both hardware and software PEs are considered. Task execution time depends on the type of PE. Large-scale random task graphs are generated to conduct the experiments. Results for five different types of graphs created having 50⇒400 nodes are presented here. Multiple graphs for each type are used and each task graph is then tested for a range of HW area and timing constraints. Timing constraints are varied from 25000 to 70000 time units for different task graphs (graph 'a' to 'e') of 50 nodes each. Area constraints range from 1800 to 16500 area units. Various regular topologies are considered for different cases and it can be concluded from our experiments that overhead for regular topologies increases with smaller pipeline period.

Table 6 lists the target systems' pipeline period time, hardware area with the number of PEs and pipeline stages for 50 node task graphs while Table 7 provides the topology information. It can be observed from the results that for tighter timing constraints extra PEs are required as number of missing links and inter-PE communication increase. It is also observed that number of missing links increase for tighter constraints and with the larger number of tasks in the graph. For tighter constraints, extra PEs are required to cater for possible delays introduced by the regular topology mapping.

TABLE 6 Time and Area Results for 50 Node Task Graphs

Task Graph	Constraints		Results					
	Time	Area	System Time	System Area	PEs	Pipeline Stages	SW PEs	HW PEs
a	25000	8500	22574	8492	8	8	6	2
a	40000	5300	30884	5268	4	4	4	0
a	50000	2700	46840	2634	2	2	2	0
a	70000	1400	66795	1317	1	1	1	0
b	25000	14200	24851	14131	8	8	6	2
b	30000	9600	27336	9529	6	7	4	2
b	40000	7000	35579	6903	3	3	3	0
b	50000	5000	40417	4602	2	2	2	0
c	25000	16500	23594	16456	8	7	6	2
c	30000	11000	29092	10742	5	5	4	1
c	35000	8100	33442	8031	3	3	3	0
c	40000	5500	39606	5354	2	2	2	0
d	25000	11500	24144	11390	10	6	6	4
d	30000	9600	28298	9512	8	5	5	3
d	40000	5300	38595	5232	3	3	3	0
d	45000	3500	43813	3488	2	2	2	0
e	30000	11000	26350	10075	5	6	5	0
e	40000	8100	34844	8060	4	4	4	0
e	45000	6100	42934	6045	3	3	3	0
e	50000	4100	49468	4030	2	2	2	0

TABLE 7 Topology Information for 50 Node Task Graphs

Graph	Time Constraint	Tree Topology		Mesh Topology		Hypercube Topology		Selected Topology	Extra PEs
		$Overhead_{TOP}$	$Links_M$	$Overhead_{TOP}$	$Links_M$	$Overhead_{TOP}$	$Links_M$		
a	25000	30573	30	29057	26	17419	24	HYPERCUBE	2
a	40000	15117	6	7475	4	7475	4	HYPERCUBE	1
b	25000	21260	27	23021	22	17601	23	HYPERCUBE	1
b	30000	9834	14	10866	13	9452	11	HYPERCUBE	0
b	40000	6755	2	6755	2	6755	2	TREE	0
c	25000	28498	29	25701	26	17140	24	HYPERCUBE	3
c	30000	13322	9	13210	7	13227	8	MESH	2
c	35000	7341	2	7341	2	7341	2	TREE	0
d	25000	27041	37	31235	33	22872	30	HYPERCUBE	2
d	30000	21888	22	24441	18	20920	18	HYPERCUBE	1
d	40000	8756	2	8756	2	8756	2	TREE	0
e	30000	12597	12	17438	10	13475	10	TREE	0
e	40000	9450	6	7626	4	7626	4	HYPERCUBE	1
e	45000	5958	2	5958	2	5958	2	TREE	0

5.3. Co-synthesis Algorithm Execution Time

Time taken by the co-synthesis approach for a multiprocessor embedded architecture is an important parameter for its effectiveness. To study and investigate this effect, execution time of the algorithm for each test case (a, b, c, d and e) is recorded with varying number of tasks (50⇒400 nodes). The co-synthesis algorithm is of iterative nature and its execution time does not solely depend on the number of tasks. Tighter hardware and timing constraints require a large number of iterations. Fig. 14 presents the minimum, average and maximum execution times for a wide range of task graphs. Our co-synthesis approach is able to provide final output with in 2.5 seconds for the largest task graphs (400 tasks). Worst-case execution time to co-synthesize the application task graphs with 100 nodes is lower than 500 milliseconds when executed on a Pentium-IV based workstation. This clearly shows that the cosynthesis methodology is practical and applicable to various types of applications. For example Chatha's co-synthesis algorithm took 30 minutes for a 30-node task graph when executed on a comparable working station [17]. We have also executed the Wolf's co-synthesis technique [12] for 50 to 400 node task graphs and its execution time is in the range of hundreds of seconds as compared to around two seconds for our co-synthesis approach.

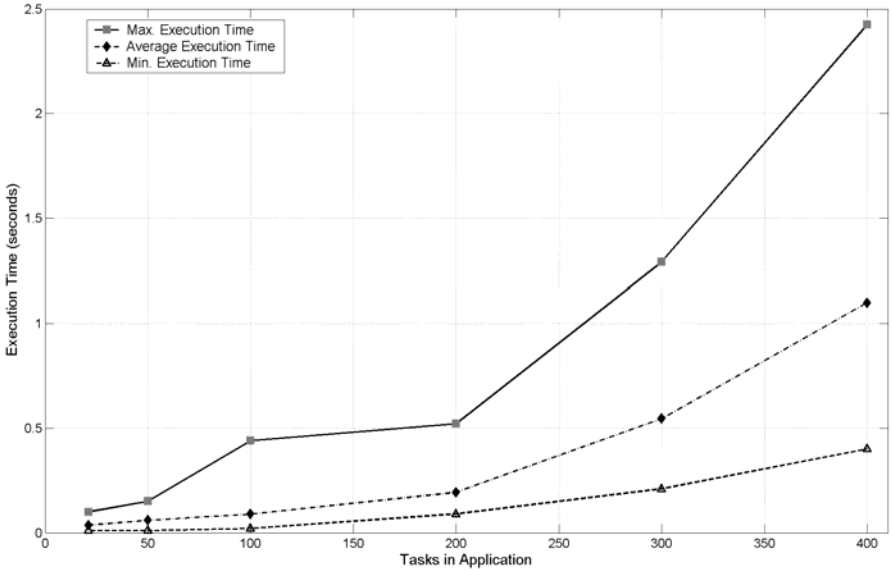


Fig. 14. Co-synthesis Algorithm Execution Time

6. CONCLUSIONS

A co-synthesis approach for multiprocessor embedded architectures is presented in this paper. The application is specified as an acyclic task graph and the co-synthesis algorithm has access to a library of PEs with the profiling information for each task. The timing and hardware area constraints are also fed to the algorithm. The co-synthesis methodology selects suitable PEs and creates pipeline stages for task execution. It performs co-synthesis by adding PEs to the target system in an iterative manner. The novel features of our approach are the

creation of pipeline stages during task allocation and scheduling as well as producing regular-topology architectures including hypercube, mesh, trees, etc. The initial irregular architecture generated by our approach can be considered for application specific MPSoC architectures. Our co-synthesis formulation can be also be extended to include shared memory multiple busses communication networks rather than a point-to-point communication network employed in this paper.

Different experiments have been conducted to demonstrate the efficacy of our approach. In the first experiment, MPEG encoder architecture is produced, which is tested with a wide range of timing and area constraints and our co-synthesis technique is able to generate pipelined architecture as well as PE schedules in less than a few seconds. Other experiments are also conducted for embedded system presented by large-size random task graphs consisting of 50 to 400 tasks. The methodology assumes coarse-grained task graphs, therefore graph with 400 tasks represent large-scale and complex embedded applications. Our co-synthesis algorithm is capable of generating efficient and sub-optimal embedded architectures. It is also compared with the Wolf's architectural co-synthesis technique [12] and for large-scale embedded systems, our co-synthesis framework produces better results in terms of regular topology and economical systems.

ACKNOWLEDGMENT

This research is partly supported by a grant from NSERC Canada. The authors would like to acknowledge the support from CMC in terms of co-design tools and prototyping systems. The authors appreciate the constructive comments from anonymous reviewers that have improved the paper.

REFERENCES

[1] G. De Micheli, "Computer aided hardware-software co-design," IEEE Micro, vol. 14 pp. 10-16, 1994.
[2] T. Benner, R. Ernst, D. Herrmann, T. Scholz and W. Ye, "The COSYMA system," in: HW/SW-Codesign*:* Principles and Practice, Kluwer Academic Publishers, Boston, 1997.
[3] D. Gajski, F. Vahid, S. Narayan and J. Gong, "SpecSyn: An environment supporting the specify-explore-refine paradigm for hardware/software system design," IEEE Tran. on Very Large Scale Integration Systems, vol. 6, pp. 84-100, 1998.
[4] F. Balarin, A. Sangiovanni-Vincentelli, K. Suzuki, P.D. Giusto, A. Jurecska, C. Passerone, E. Sentovich, B. Taabbara, M. Chiodo, H. Hsieh and L. Lavagno, "The POLIS approach," in: HW/SW-Codesign of Embedded Systems, Kluwer Academic Publishers, Norwell MS, 1997.
[5] T.Y. Yen and W. Wolf, "Performance estimation for real-time distributed embedded systems," IEEE Trans. on Parallel and Distributed Systems, vol. 9, pp. 1125-1136, 1998.
[6] L. Cai and D. Gajski, Transaction level modeling: an overview, in: Proc. IEEE/ACM/IFIP Int. Conf. Hardware Software Codesign and System, October 2003, pp. 19-24.
[7] K. Van Rompaey, D. Verkest, I. Bolsens and H. De Man, "CoWare–a design environment for heterogeneous hardware/software systems," in: Proc. European Design Automation Conference, September 1996, pp. 252-257.
[8] S. Vercauteren, B. Lin and H. De Man, "Constructing application-specific heterogeneous embedded architectures from custom hardware/software applications," in: Proc. Design Automation Conference, Las Vegas, NV, June 1996, pp. 521-526.
[9] S. Prakash and A.C. Parker, "SOS: Synthesis of application specific heterogeneous multiprocessor systems," Journal of Parallel and Distributed Computing, vol. 16, pp. 338-351, 1992.
[10] B.P. Dave, G. Lakshminarayana and N. K. Jha, "COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems," IEEE Tran. on VLSI Systems, vol. 7, pp. 92-104, 1999.

[11] B.P. Dave and N.K. Jha, "COHRA: Hardware-software cosynthesis of hierarchical heterogeneous distributed embedded systems," IEEE Tran. on Computer Aided Design of Integrated Circuits and Systems, vol. 17, pp. 900-919, 1998.
[12] W. Wolf, "An architectural co-synthesis algorithm for distributed, embedded computing systems, IEEE Tran. on VLSI Systems," vol. 5, pp. 218-229, 1997.
[13] S. Banerjee, T. Hamada, P.M. Chau and R.D. Fellman, "Macro pipelining based scheduling on high performance heterogeneous multiprocessor systems," IEEE Tran. on Signal Processing, vol. 43, pp. 1468-1484, 1995.
[14] Yu-Kwong Kwok and I. Ahmad, "Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors," IEEE Tran. on Parallel and Distributed Systems, vol. 7, pp. 506-521, 1996.
[15] P.D. Hoang and J.M. Rabaey, "Scheduling of DSP programs onto multiprocessors for maximum throughput," IEEE Tran. on Signal Processing, vol. 41, pp. 2225-2235, 1993.
[16] S. Bakshi and, D.D. Gajski, Partitioning and pipelining for performance constrained hardware/software systems," IEEE Trans. on Very Large Scale Integration Systems, vol. 7, pp. 419-432, 1999.
[17] K.S. Chatha and R. Vemuri, "Hardware-software partitioning and pipelined scheduling of transformative applications," IEEE Tran. on Very Large Scale Integration Systems, vol. 10, pp. 193-208, 2002.
[18] A. Jerraya and W. Wolf, Multiprocessor System-on-Chips, Morgan Kaufman Publishers, 2005.
[19] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," ACM Computing Surveys, vol. 28 pp. 1-51, 2006.
[20] D. Bertozzi, A. Jalabert, S. Murali, R., Tamhankar, L. Benini and G. De Micheli, "NoC synthesis flow for customized domain specific multiprocessor systems-on-chip," IEEE Tran. on Parallel and Distributed Systems, vol. 16, pp. 113-129, 2005.
[21] S. Murali and G. De Micheli, "SUNMAP: A tool for automatic topology selection and generation for NoCs," in Proc. Design Automation Conf., San Diego, California, 2004, pp. 914-919
[22] A. Jalabert, S. Murali, L. Benini and G. De Micheli, "XpipesCompiler: a tool for instantiating application specific network-on-chip," in Proc. Design Automation and Test in Europe Conf., 2004, pp. 884-889.
[23] R. Duncan, "A survey of parallel computer architectures," IEEE Computer vol. 23, pp. 5-16, 1990.
[24] L. Benini and G. De Micheli, "Networks on chips: a new soc paradigm," IEEE Computer vol. 35, pp. 70-78, 2002.
[25] D. Wiklund and D. Liu, "SoCBUS: switched network on chip for hard real time embedded systems," in Proc. Int. Symp. Parallel and Distributed Processing, 2003, pp. 78-85
[26] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in Proc. Design Automation Test Europe Conf., 2000, pp. 250-256.
[27] F. Karim, A. Nguyen, S. Dey and R. Rao, "On-chip communication architecture for OC-768 network processors," in Proc. Design Automation Conf., 2001, pp. 678-683.
[28] G.N. Khan , J. Levman and J. Alirezai, Hardware-software co-Synthesis of fault tolerant heterogeneous embedded computer systems," in Proc. IEEE Canadian Conf on Electrical and Computer Engineering, Ottawa, May 2006.
[29] G. N. Khan and U. Ahmed, "Hardware-software cosynthesis of multiprocessor embedded architectures," in Proc. 4th IEEE Int. Symp. Embedded Computing. (AINA-Workshops) Niagara Falls, Canada, pp. 804-810, 21-23 May 2007.
[30] K. Goossens, J. Dielissen and A. Radulescu, "Aethereal network on chip: Concepts, architectures, and implementations," IEEE Design and Test of Computers, vol. 22, pp. 414-421, 2005.
[31] D. Le Gall, "MPEG: A video compression standard for multimedia applications," Communications of the ACM vol. 34, pp. 46-58, 1991.
[32] J. Henkel and R. Ernst, "High-level estimation techniques for usage in hardware/software co-design," in Proc. Asia South Pacific Design Automation Conf., Yokohama, Japan, 1998, pp. 353-360.