A NOVEL METHODOLOGY FOR TASK DISTRIBUTION IN HETEROGENEOUS RECONFIGURABLE COMPUTING SYSTEM

Mahendra Vucha[1] and Arvind Rajawat[2]

[1,2]Department of Electronics & Communication Engineering, MANIT, Bhopal, India.
[1]Department of Electronics & Communication Engineering, Christ University, Bangalore, India.

ABSTRACT

Modern embedded systems are being modeled as Heterogeneous Reconfigurable Computing Systems (HRCS) where Reconfigurable Hardware i.e. Field Programmable Gate Array (FPGA) and soft core processors acts as computing elements. So, an efficient task distribution methodology is essential for obtaining high performance in modern embedded systems. In this paper, we present a novel methodology for task distribution called Minimum Laxity First (MLF) algorithm that takes the advantage of runtime reconfiguration of FPGA in order to effectively utilize the available resources. The MLF algorithm is a list based dynamic scheduling algorithm that uses attributes of tasks as well computing resources as cost function to distribute the tasks of an application to HRCS. In this paper, an on chip HRCS computing platform is configured on Virtex 5 FPGA using Xilinx EDK. The real time applications JPEG, OFDM transmitters are represented as task graph and then the task are distributed, statically as well dynamically, to the platform HRCS in order to evaluate the performance of the designed task distribution model. Finally, the performance of MLF algorithm is compared with existing static scheduling algorithms. The comparison shows that the MLF algorithm outperforms in terms of efficient utilization of resources on chip and also speedup an application execution.

KEYWORDS

Heterogeneous Reconfigurable Computing Systems, FPGA, parallel processing, concurrency, Directed Acyclic Graph.

1. INTRODUCTION

Modern embedded systems are used for highly integrated handheld devices such as mobile phones, digital cameras, and multimedia devices. Hardware Software Co-design supports mixed hardware and software implementation to satisfy the given timing and cost constraints. Co-design is a flexible solution for the applications when hardware realization satisfies the timing but not the cost constraints whereas software solution is not fast enough. So, the hardware software co-design provides intensive solution for the modern embedded systems which are modeled with flexible computing platform like Field Programmable Gate Arrays (FPGA). The FPGA is flexible hardware that offers cost effective solution through reuse and also accelerate many multimedia applications by adopting their hardware at runtime. In real time, the tasks of parallel application must share the resources of FPGA effectively in order to enhance application execution speed and it can be achieved through effective scheduling mechanism. The aim of this paper is to introduce

DOI : 10.5121/ijesa.2015.5102

a disciplined approach to utilize the resources of embedded system, which are having reusable architectures, and also that meets the requirements of variety real time applications. Diverse set of resources like Reconfigurable Logic Units (RLUs) and soft core processors are interconnected together, with a high speed communication network, on a single chip FPGA that describes a new computing platform called Heterogeneous Reconfigurable Computing Systems (HRCS). The HRCS requires an efficient application scheduling methodology to share their resources in order to achieve high performance and also utilize the resources effectively. There are many researchers [3] [6] [7] [9] presented techniques for mapping multiple tasks to High Speed Computing Systems [26] with the aim of "minimizing execution time of an application" and also "efficient utilization of resources". In this paper, first we would describe review of the various existing task distribution methodologies for platforms like HRCS and proposed a novel task distribution methodology. In general, task distribution i.e. scheduling models are two types called static and dynamic. *Static Scheduling:* All information needed for scheduling such as the structure of the parallel application, execution time of individual tasks and communication cost between the tasks must be known in advance and they are described in [10] [11] [12] [13] [14]. *Dynamic scheduling:* The scheduling decisions made at runtime are demonstrated in [8] [20] [22] [24] [26] and their aim is not only enhancing the execution time and also minimize the communication overheads. The static and dynamic scheduling heuristic approaches, proposed by various researches, are classified into four categories: List scheduling algorithms [20], clustering algorithms [11], Duplication Algorithms [22], and genetic algorithms. The list scheduling algorithms [20] provides good quality of task distribution and their performance would be compatible with all real time categories. So, in this paper we have been motivated to develop *list scheduling algorithm* and generally it has three steps: task selection, processor selection and status update. In this paper, we have developed a list based task distribution model which is based on the attributes of the tasks of an application and computing platform. The remaining paper is organized as the literature review in chapter 2, problem formulation in chapter 3, task distribution methodology in chapter 4, implementation scheme in chapter 5, and results discussion in chapter 6 and finally paper is concluded in chapter 7.

2. LITERATURE REVIEW

The task distribution problems for CPU as well as for reconfigurable hardware have been addressed by many researchers in academic and industry. However, research in this paper is targeted to CPU – FPGA environment. The articles discussed in this session describe various task scheduling methodologies for heterogeneous computing systems. A computing platform called MOLEN Polymorphic processor described in [26] which is incorporated with both general purpose and custom computing processing elements. The MOLEN processor is designed with arbitrary number of programmable units to support both hardware and software tasks. An efficient multi task scheduler in [9] proposed for runtime reconfigurable systems and also it has introduced a new parameter called Time-Improvement as cost function for compiler assisted scheduling models. The Time-Improvement parameter described based on reduction-in-task-execution time and distance-to-next-call of the tasks in an application. The scheduling system in [9], targeted to MOLEN Polymorphic processor [26] and in which the scheduler assigns control of tasks and less computing intensive tasks to General Purpose Processor (GPP) whereas computing intensive tasks are assigned to FPGA. The task scheduler in [9] outperforms previous existing algorithms and accelerates task execution 4% to 20%. In [6], an online scheduling is demonstrated for CPU-FPGA platform where tasks are described into three categories such as

Software Tasks executes only on CPU, Hardware Tasks executes only on FPGA and Hybrid Tasks executes on both CPU & FPGA. The scheduling model [6] is integration of task allocation, placement and task migration modules and considers the reserved time of tasks as cost function to schedule the tasks of an application. An On-line HW/SW partitioning and co-scheduling algorithm [3] proposed for GPP and Reconfigurable Processing Unit (RPU) environment in which Hardware Earliest Finish Time (HEFT) and Software Earliest Finish Time (SEFT) are estimated for tasks of an application. The difference between HEFT and SEFT imply to partition the tasks and EFT used to define scheduled tasks list for GPP and RPU as well. An overview of tasks co-scheduling [7] [31] is described to µP and FPGA environment and it have been defined from different research communities like Embedded Computing (EC), Heterogeneous Computing (HC) and Reconfigurable Hardware (RH). The Reconfigurable Computing Co-scheduler (ReCoS) [7] integrates the strengths of HC and RH scheduling policies in order to effectively handle the RC system constraints such as the number of FFs, LUTs, Multiplexers, CLBs, communication overheads, reconfiguration overheads, throughputs and power constraints. The ReCoS, as compared with EC, RC and RH scheduling algorithms, shows improvement in optimal schedule search time and execution time of an application. Hardware supported task scheduling is proposed in [15] for Dynamically Reconfigurable SoC (RSoC) to utilize the resources effectively for execution of multi task applications. The RSoC architecture comprises a general purpose embedded processor along with two L1 data and instruction cache and number of reconfigurable logic units on a single chip. In [15], task systems are represented as Modified Directed Acyclic Graph (MDAG) and the MDAG defined as tuple G = (V, E^d, E^c, P), where V is set of nodes, E^d and E^c are the set of directed data edges and control edges respectively and P represents the set of probabilities associated with E^c. The conclusion of the paper [15] states that Dynamic Scheduling (DS) does not degrade as the complexity of the problem increase whereas the performance of Static Scheduling (SS) decline. Finally, the DS outperforms the SS when both task system complexity and degree of dynamism increases. Compiler assisted runtime scheduler [16] is designed for MOLEN architecture where the run time application is described as Configuration Call Graph (CCG). The CCG assigns two parameters called *the distance to the next call* and *frequency of calls in future* to the tasks in an application and these parameters acts as cost function to schedule the tasks. Communication aware online task scheduling for partially reconfigurable systems [17] distributes the tasks of an application to 2D area of computing architecture and where communication time of tasks acts as cost function to schedule the tasks. The scheduler in [17] describes the tasks expected end time as $t_c = t_{latest} + t_{config} + t_{comm1} + t_{exe} + t_{comm2}$, where t_{latest} is completion time of already scheduled task, t_{config} is task configuration time, t_{comm1} is data/memory read time, t_{exe} is task execution time and t_{comm2} is data/memory write time and it could run on host processor. HW/SW co-design techniques [18] are described for dynamically reconfigurable architectures with the aim of deciding execution order of the event at run time based on their EDF. Here they have demonstrated a HW/SW partitioning algorithm, a co-design methodology with dynamic scheduling for discrete event systems along with a dynamic reconfigurable computing multi-context scheduling algorithm. These three co-design techniques [18] minimize the application execution time by paralleling events execution and it could be controlled by host processor for both shared memory and local memory based Dynamic Reconfigurable Logic (DRL) architectures. When number of DRL cells is equal or more than three, the techniques in [18] brings better optimization for shared memory architecture compared to the local memory architectures. A HW/SW partitioning algorithm presented in [30] to partition the tasks as software tasks and hardware tasks based on their waiting time. A layer model in [20] provides systematic use of dynamically reconfigurable hardware and

also reduces the error-proneness of the system components. A methodology in [34] presented for building real time reconfigurable systems and they ensure that all the constraints of the applications are met. In [34, the Compulsory-Reuse (CR) tasks in an application are found and they are used to calculate the Loading-Back factor that support the reuse of resources. The various research articles addressed in this section describes task distribution of non real time systems in order to achieve optimized performance and throughput but they may miss deadlines in real time. In this article, we also focused on non real time systems with the objective to meet dead line requirements at runtime.

3. TASK DISTRIBUTION PROBLEM

Generally, a task distribution methodology consists of an application, targeted architecture and criteria to task distribution. So this chapter is intended to brief about task graph of an application, targeted architecture, performance criteria, motivation to the research and some necessary assumption.

3.1 Targeted architecture

Heterogeneous reconfigurable hardware is an emerging technology [1] [26] [32] [33] due to their high performance, flexibility, area reuse and also provides faster time-to-market solutions [1] [33] for real time applications compared to ASIC solutions. In this research, a computing platform is modeled on a single chip FPGA which consists of a soft core processor (i.e. microprocessor is configured on core of FPGA) and multiple Reconfigurable Logic Unit (RLU) as processing elements shown in figure 1. The soft core processor is static core in nature and it would execute software version of tasks in an application. The reconfigurable units RLU1, RLU2, RLU3, RLU4 and RLU5 support dynamic reconfiguration at runtime for hardware tasks of an application. The Cache memory is dedicated for soft core processor to store the instructions and input/output data while task execution. The shared memory stores the task executables and input/output data for both soft core and hard core (i.e. RLU computing elements) processing elements.

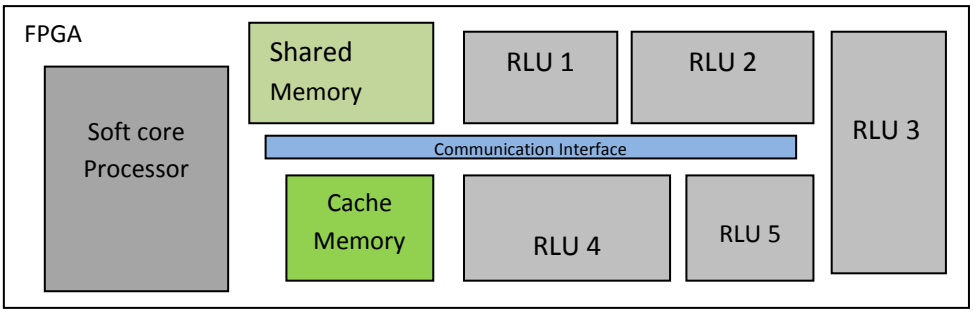


Figure 1. Target heterogeneous architecture

The soft core and hard core Processing Elements (PEs) in the targeted architecture are wrapped with communication interface so that provides interface between memory and PEs for data interchange. The hardware reconfiguration and tasks execution is managed by a task distribution model which would be the objective of this research and it can be demonstrated in coming

chapters. The RLUs independently execute the tasks and communicate with each other. The targeted platform can be implemented on Xilinx Virtex-5 and Virtex-6 or Altera FPGA devices. The FPGA Vendors provide specific design tools to develop custom computing platforms where as the Xilinx EDK development tool is used to develop a processor based reconfigurable system called Heterogeneous Reconfigurable Systems.

3.2 Application as Task Graph

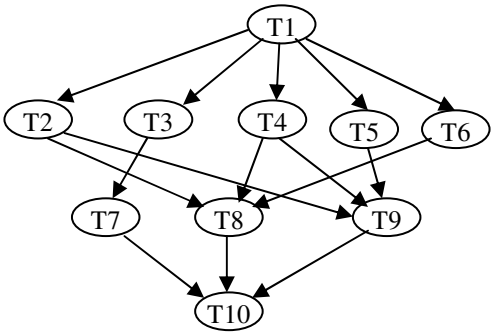
An application is represented by a waited Directed Acyclic Graph (DAG) G = (V, E), where V represents set of tasks $V = \{v_1, v_2, v_3 \ \ldots \ v_N\}$ and E represents set of edges E = {e_{11}, e_{12} …., e_{21} ,e_{22}, ….., e_{MN}} between the tasks. Each edge $e_{ij} \epsilon E$ represents precedence constraint such that task v_i should complete its execution before v_j. In a DAG, task without any predecessor is an entry task and task without successor is an exit task. Generally, the tasks execution of an application is non – preemptive but in this research we have considered that the tasks behavior would be either preemptive or non-preemptive. In this research, the tasks of an application, i.e. DAG, are waited with their attributes (stated as parameters in rest of the paper) like a_i task arrival time, d_i task deadline, w_i task area in terms of number of bit slices required, rc_i task reconfiguration time, he_i task execution time on FPGA and se_i task execution time on soft core processor, where i = 1, 2, 3 . . . . N and N is equal to number of tasks in DAG. Task arrival time a_i is the starting time of task execution and task deadline d_i would be maximum time allowed to complete the task. The tasks area w_i is described as the number of logic gates required for task execution on FPGA. The task configuration time rc_i is the time taken by FPGA to adopt their hardware to execute the task. In this paper, we have assumed that the configuration time is fixed for all tasks, since all RLUs configuration time is fixed in FPGA. The task execution time is the time taken by task to complete their execution either on µP called se_i or FPGA called he_i.

3.3 Performance Criteria

In this research, we intended to represent parallel applications as DAG and also they carry task parameters. The one or more task parameters may acts as cost function to distribute applications to the resources of computing architecture. Initially, the tasks of parallel applications have been executed on soft core processor as well on hard core FPGA in order to acquire their parameters. The acquired parameters like area (number of slices on FPGA), execution time are maintained in the form of cost matrix$[C]_{N\times2}$. The order of the execution time cost matrix is N×2, since there are N tasks assumed in an application and each task is executed on 2 processing elements i.e. soft core processor and FPGA. The cost matrixes of an application plays crucial role while distributing the tasks to the targeted architecture. In practical, the execution time of an application depends on Finish Time (FT) of the exit task and called as scheduled length of an application. In this research, the objective of task distribution model is to minimize the scheduled length of an application and also efficient utilization of Heterogeneous architecture resources. The FT of task depends on nature of resources used for computation and also on tasks arrival time. The FT of the task, FT $(t_i) = a_i + e_i$, where e_i is an execution time of a task t_i on soft core processor or FPGA i.e. $e_i \in [C]_{N\times2}$. The arrival time of task t_i is depends on finish time of the task t_{i-1} i.e. $a(t_i) =$ FT(t_{i-1}) and so on.

3.4 Motivational Example

The task graph shown in figure 2 is an example taken form [20] [35] and it is targeted to a heterogeneous reconfigurable platform having one CPU and three RLUs as computing elements.



Task (Node)	Area on FPGA	Execution Time in µsec	
		Soft core processor	FPGA
T1	200	14	12
T2	180	13	10
T3	120	11	9
T4	180	13	10
T5	150	12	9
T6	170	13	11
T7	90	7	5
T8	70	5	3
T9	250	18	15
T10	300	21	17

Figure 2 Task graph and its attribute table on soft core processor and FPGA

Generally, execution time of task graph depends on the processing elements on which their tasks are executed. In this research, the reconfiguration latency is assumed as constant and equal to zero. The task graph, in figure 2, is executed on different configurations of targeted platform as shown in figure 3. In the figure 3, along the x-axis represents execution time in millisecond and the y-axis represents platform configurations used for task graph execution. An application, shown in figure 2, is scheduled to single microprocessor and FPGA with single RLU and the execution time in the both cases are shown in figure 3(a) and 3(b) respectively. The application ideal schedulable length on CPU, figure 3(a), is 127 µsec. and it would be 101 µsec., when the application is mapped to FPGA with single RLU, figure 3(b). So, schedulable length of an application can be minimized when and only RLU acts as computing element. Since FPGA support partial reconfiguration, we could cluster the FPGA into multiple RLUs to support parallel task execution and it further reduces the schedulable length of an application. Application scheduled to multiple RLUs platform where tasks found the required area and the execution time is 65µs as shown in figure 3(c).

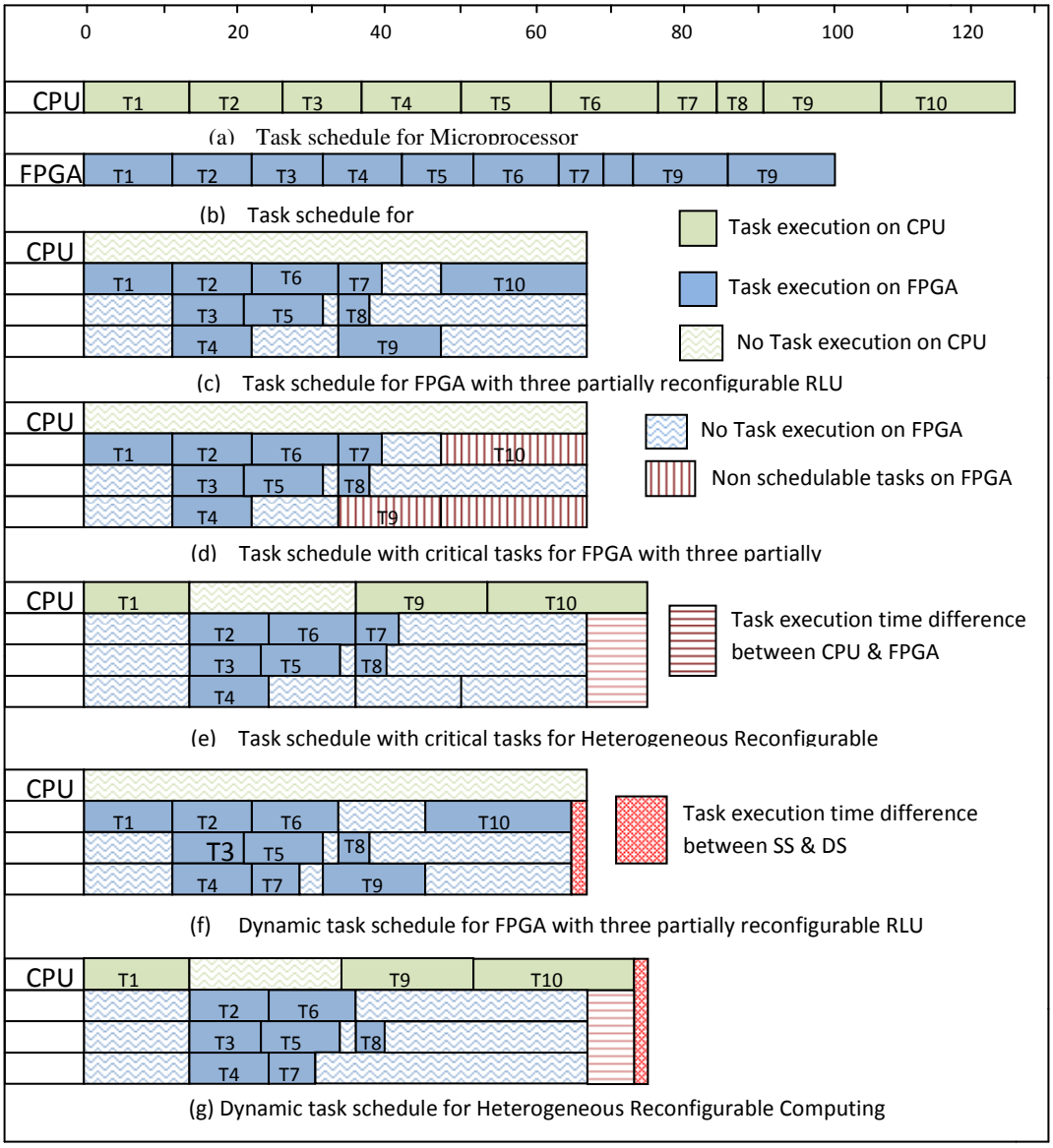


Figure 3. Scheduled length of task graph on heterogeneous Reconfigurable Computing Systems

In real time, tasks may require hardware area which could not available on FPGA and they can be called as critical tasks. The critical tasks may leads to infinite schedulable length (i.e. application could not fully executed) and such critical tasks can be executed on microprocessor due to microprocessor flexibility for software tasks. In this article, tasks which do not find the required area on FPGA can be treated as critical tasks. For example, when we consider RLUs area on FPGA is less than 200 then the tasks T1, T9 and T10 in task graph (Figure 2) becomes critical tasks and the application could not execute on RLUs of FPGA. The scenario of scheduling critical tasks to the platform where tasks do not find required area and its execution time is infinite as shown in figure 3(d). The execution time infinite indicates that the application is partially executed (i.e. tasks 9 and 10 are not completely executed) due to lack of resources and it can be

addressed effectively by introducing a soft core processor along with FPGA, where as the processor acts as a flexible computing element for critical tasks. The task schedule for such HRCS platform is shown in figure 3(e) and the execution time is 74 µs which is more than 3(c) but application is executed successfully. The dynamic task schedule of the application to the platform with multiple RLUs only is shown in figure 3(f) and its execution time is 63µs. Similarly, the dynamic task schedule of the application to the platform HRCS is shown in 3(g) and its execution time is 71 µs. From the figures 3 (c), (e), (f), (g), it is clear that the dynamic schedule enhances the application execution speed compared to static schedule. The idle time of RLUs and processors is used for execution of task of parallel applications. In this paper, we are intended to address dynamic scheduling techniques for real time applications.

3.5 problem statement and assumptions

An overview of different steps in scheduling of real time application to HRCS platform is described in figure 4. An application would be represented as weighted DAG and it is passed to task prioritization and HW/SW partitioning modules. The prioritization modules assigns priorities to the tasks of DAG based on their attributes and then the partitioning module partition the tasks into hardware and software tasks.

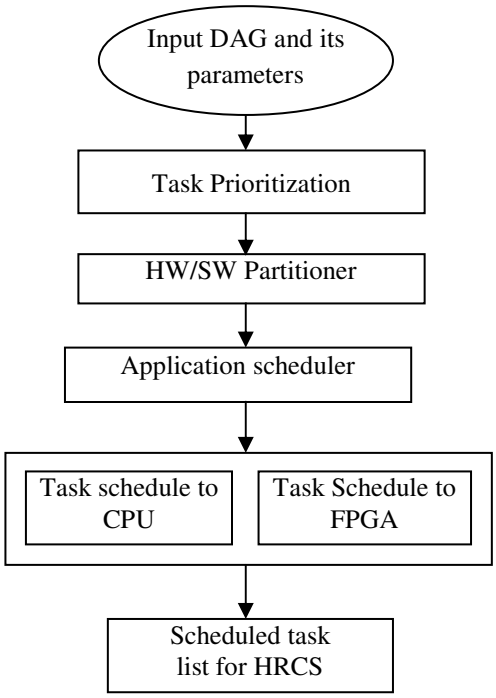


Figure 4. Flow chart of task distribution methodology

Task prioritization assigns priority to each task in such a way that meets the deadlines of an application. The HW/SW partitioner partition the tasks based on the resources availability and nature of task and the scheduler prepares the scheduled task list to either CPU or FPGA based on task parameters stated in section 3.2. These three sequential steps plays major role in distribution of tasks to HRCS i.e. the task graphs are distributed sequentially not concurrently. In this

research, tasks would be scheduled to CPU and FPGA concurrently i.e. task graphs are executed concurrently based on availability of the HRCS resources. Each processing element in HRCS intended to run only one task at a time and each task may be loaded to either CPU or FPGA. In HRCS, data among the tasks can be exchanged by shared memory and the tasks nature is assumed as either preemption or non preemption. Let's assume that the set of tasks $T = \{t_1, t_2, \ldots\ldots, t_N\}$ of an application are represented as weighted DAG and task arrival time would be stochastic in nature. The tasks set T in DAG would be partitioned into three types called software tasks (ST), hardware tasks (HT) and hybrid tasks (HST) based on their area (i.e. resources width in terms of number of bit slices) w_i and preemption nature, as stated below.

1. The set of tasks, which are preempted in nature and could not find required area on RC of HRCS, can be treated as software task set (ST). $ST = \{st_1, st_2, \ldots\ldots, st_m\}$, $st_i \in$ ST, $(1 \leq i \leq m)$, having the parameters a_i, d_i and se_i and run only on µP.
2. The set of tasks, which are non-preempted in nature and could find required area on RC of HRCS, can be treated as hardware task set (HT). $HT = \{ht_1, ht_2, \ldots\ldots, ht_n\}$, $ht_i \in HT$,$(1 \leq i \leq n)$, having parameters a_i, d_i, w_i, c_i and he_i and run only run on FPGA.
3. The set of tasks, which are preempted in nature and could find required area on RC of HRCS, can be treated as hybrid task set (HST). $HST = \{hst_1, hst_2, \ldots\ldots, hst_p\}$, $hst_i \in HST$ $(1 \leq i \leq p)$, having parameters a_i, d_i, w_i, c_i, se_i and he_i and run either on µP or FPGA.

In this research, the task parameters are estimated to the platform HRCS statically by different techniques. Task area width w_i is estimated with the help of synthesis tools like XILINX ISE, Synopsys Design Compiler. Task hardware configuration parameters like c_i, he_i are estimated by configuring them to hard core processor i.e. FPGA, whereas software execution time se_i is estimated by executing the task on soft core processor i.e. µP. The partitioned tasks are prioritized based on the level of tasks and then scheduled them to either to soft core processor or hard core processor. In this article, we made decision to direct software tasks to µP and hardware tasks to FPGA permanently but the hybrid tasks are directed to either µP or FPGA based on resources availability at that instant of time.

4.PROPOSED METHODOLOGY FOR TASK DISTRIBUTION TO HETEROGENEOUS COMPUTING SYSTEMS

The execution time of real time applications always depends on targeted platform and its computing elements. Distribution of the tasks of real time applications becomes complex when there are multiple heterogeneous computing elements present in computing platform and it has been addressed by many researchers for various computing platforms. In this article, we intended to describe a task distribution methodology to the platform where CPU and FPGA would be computing elements. Usually task distribution takes place in three steps task prioritization, HW/SW partition and application schedule as shown in figure 4. In our research, the task prioritization generates the priorities for the tasks based on their level in task graph. Task partitioner partition the tasks into software tasks, hardware tasks and hybrid tasks based on the resources required and availability. The scheduler prepares task distribution list to the hard core processor and soft core processor based on tasks dead line. The behavior and Pseudocode for these three steps are described in the following sub sections.

4.1 Task prioritization

Initially task graph is represented as adjacency matrix which shows dependency of tasks in a task graph. The adjacency matrix is used to find Level of individual tasks in the task graph. The Level of tasks in task graph acts as cost function for task prioritization. In any task graph, source task gets highest priority and sink task gets lowest priority in order to maintain dependency between tasks. The Pseudocode for task prioritization is described in algorithm 1.

Algorithm 1: Pseudocode for Task prioritization

```
for each task_graph, no_tasks, no_next_task do
        Compute adjacent Matrix
        Compute Level of task
        while ((Level)i > 0)
                Assign_Priority (task, Level);
                Sort (priority_list)
         end
 end
```

Tasks may have equal priority since more than one task may exist at same level in a task graph. The tasks T2, T3, T4, T5 and T6 in task graph, shown in figure 2, are at same level and they have equal priority according to algorithm 1. In the task graph (figure 2), task T1 has first priority, tasks T2, T3, T4, T5 and T6 assigned with second priority, tasks T7, T8 and T9 assigned with third priority and finally task T10 assigned with forth priority. The prioritized tasks are sorted according to their priority increasing order and then moved for HW/SW partition module.

4.2 Task Partition

The prioritized tasks are partitioned into software task, hardware tasks and hybrid tasks based on resources available and preemption nature of the task. The pseudocode for task partition is described in algorithm 2.

Algorithm 2: Pseudocode for Task Partition

```
 for initial prioritized task_graph, no_tasks do
        HT_queue = 0;  HST_queue = 0;   ST_Queue = 0;
        if ( ( area (Ti ) < available_RLU) & preemption_nature = false) then
                HT_queue = Ti
        else if ( ( area (Ti ) <  available_RLU) & preemption_nature = true) then
                HST_queue = Ti
         else
                ST_Queue = Ti
         end
10 end
```

The initial prioritized tasks of task graph would be accepted by the task partition module as input. The ST_Queue, HT_Queue and HST_Queues are used to store software tasks, hardware tasks and hybrid tasks respectively. Initially these queues would be empty and intended to store partitioned tasks in their priority increasing order. The tasks which are non-preempted in nature and could find reconfigurable area on HRCS are sent to HT_Queue, tasks which are preempted in nature and could find reconfigurable area on HRCS are sent to HST_Queue, and tasks which are preempted in nature and could not find reconfigurable area on HRCS are sent ST_Queue. Finally, these partitioned tasks are available in their respective queues and they could be distributed to computing platform HRCS by tasks scheduling module.

4.3 Task Scheduling

The pseudocode in section 4.1 and 4.2 depict the behavior of prioritization and partition methodologies. These methodologies receive task graphs as well as their attributes as input and prepares initial schedule portioned task list. In this section, the dynamic task scheduling policy is demonstrated as combination of prioritization and resource management. The initial scheduled tasks in algorithm 1 &2 are further prioritized based on the cost function called Minimum Laxity First (MLF) and availability of the resources. The pseudocode of task scheduling model is described in algorithm 3.

Algorithm 3: Pseudo code for Task scheduling

```
1 while (no_task graphs >0)
2         for initial scheduled and partitioned task graph, no_tasks do
3                 Compute MLF of individual tasks in task graph
4                 Assign_Priority (task, MLF)
5                         Sort (priority_list)
6         end
7         RLU_Impementation_Queue = 0; CPU_Impementation_Queue = 0;
8         for prioritized tasks, no_tasks do
9                 if ( Ti ∈ HT_Queue) then
10                                Ti -> RLU_Implementation Queue
11                else if ((Ti ∈HST_Queue) &(RLU_available ) then
12                        Ti -> RLU_Implentation Queue
13                else
14                        Ti ->  CPU_Implementation Queue
15                end
16        end
17        while (RLU_Implementation_Queue != empty)
18                Wait for RLU
19                Assign task to available RLU
20        end
21        while (CPU_Implementation_Queue != empty)
22                Wait for CPU
23                Assign task to CPU
24        end
25 end
```

The task scheduler accepts the partitioned tasks as input and computes a parameter called Minimum Laxity First (MLF) for all the individual tasks which are in same level of the task graph. The expression for MLF is $\boldsymbol{t_{MLF} = d_i - e_i - a_i}$. The MLF acts as cost function to prioritize parallel tasks and then prioritized tasks are scheduled as their priority increasing order. The RLU_Implementation Queue and CPU_impementation Queue are used to store the task for execution on hard core processor and soft core processor respectively. The RLU_Implementation Queue stores the tasks which could execute on hardcore processor (RLUs) and the CPU_Implementation queue stores the tasks which could execute on softcore processor (CPU). Tasks in HT_Queue and also tasks in HST_Queue for which reconfigurable area available are sent to RLU_Implementation Queue. Similarly tasks in ST_Queue and tasks in HST_Queue for which reconfigurable area is not available are sent to CPU_Implementation Queue. Finally, tasks in RLU_Implementation Queue and CPU_Implementation Queue are executed on hard core processor (RLU) and soft core processor (microprocessor) respectively.

5. IMPLEMENTATION SCHEME

In this section, we describe the computing environment, real time application and methods followed for application execution. The reconfigurable computing brings flexibility for execution of wide range of application and also enhances the execution speed. So, in this research we have described a heterogeneous computing environment on a single chip FPGA called Heterogeneous Reconfigurable Computing System (HRCS). The HRCS contains a soft core processor and multiple hard core processors i.e. Reconfigurable Logic Units (RLUs) as processing elements. The soft core processor executes application in traditional method like fetch, decode and execute whereas the hard core processor reconfigures its architecture according to the behavior of an application task. The described HRCS platform is realized on Virtex 5 FPGA with a MicroBlaze as soft core processor and partially reconfigurable RLUs as hard core processing elements. In the realized heterogeneous reconfigurable platform, the MicroBlaze is equipped with a BRAM, Instruction and Data Cache memories for storing program as well data while executing an application. BRAM also acts as shared memory for RLUs to store input and output data. These functional blocks MicroBlaze, RLUs (custom hardware for application tasks), BRAM, Cache and general purpose I/O devices are interconnected through communication protocols like Processor Local Bus (PLB) and First In First Out (FIFO) as shown in figure 5.

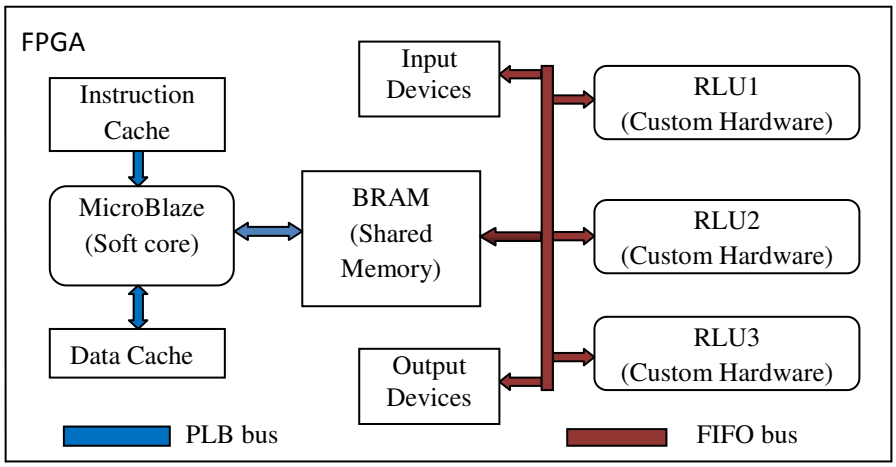


Figure 5. On-chip Heterogeneous Reconfigurable Computing System

We have selected set of task graphs extracted from multimedia applications and executed on hard core and soft core processing elements which are constructed on Virtex 5 FPGA platform. The task graph of JPEG is shown in figure 6(a) and it can be used as input to task distributing model.

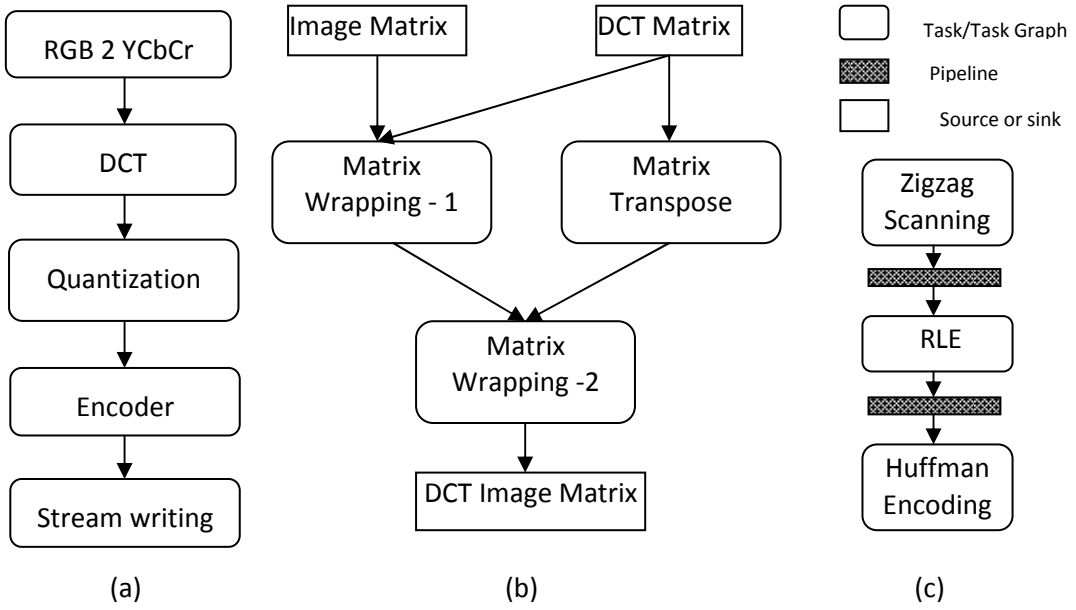


Figure 6 (a) Tasks and their dependency in JPEG (b) Task and their dependency in DCT (c) Task and their dependency in Encoder

The JPEG task graph has five levels where task T1 for RGB_to_ YCbCr at level 1, task graph has three tasksT2, T3, T4 shown in figure 6(b) for Discrete Cosine Transform (DCT) at level 2, task T5 for quantization at level 3, task graph has three tasks T6, T7, T8 shown in figure 6(c) for Encoder at level 4 and task T9 for Stream_writing at level 5. In the figure 6(b), the task T6 for matrix wrapping – 1 and T7 for matrix transpose are in the same level they could be executed concurrently. The figure 6(c) shows encoder where pipeline is introduced while hardware implementation of the task graph which increases the throughput of the task graph. The tasks in encoder can be executed sequentially. The dataflow diagram of the procedure which is followed to design HRCS and implement the real time application on HRCS is describe in figure 7.

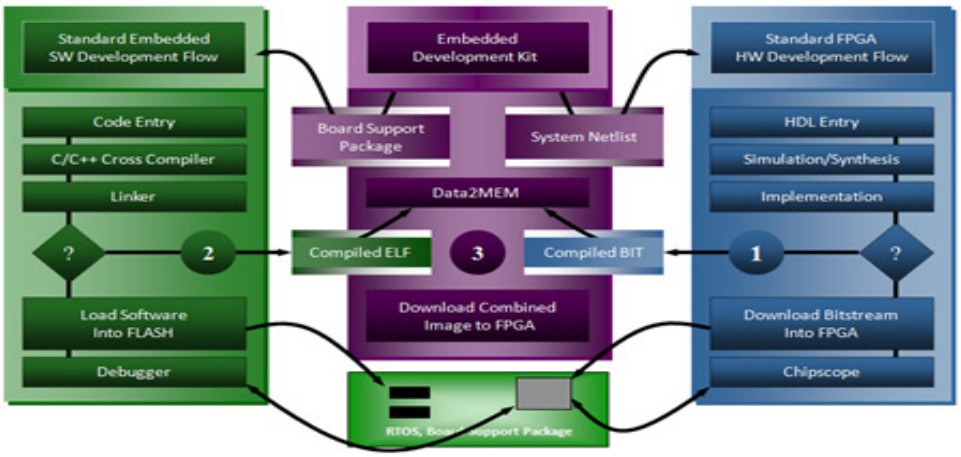


Figure 7 Xilinx Tool flow to design HRCS and task graph execution on HRCS

The Embedded Development Kit (EDK) demonstrates board support package design. So, in this research, we have been using the Xilinx EDK to realize the HRCS platform on Virtex 5 FPGA where MicroBlaze soft core is configured in part of the reconfigurable area of FPGA and the rest reconfigurable area is divided into multiple hard core processing elements. Standard embedded SW development flow supports execution of the applications on soft core processor where as the standard FPGA HW development flow supports execution of the application on hard core processor.

6. RESULTS & DISCUSSION

This section is intended to describe FPGA device utilization for configuring HRCS, evaluate the performance of the task distribution methodology described for HRCS and finally HRCS resources utilization while executing for real time applications.

6.1 Device Utilization for Heterogeneous Reconfigurable Computing System

The Virtex 5 FPGA (XC5VLX110T) device consists of 17,280 slices, 64 DSP slices and 5,328Kb of RAM memory blocks to design high performance embedded systems. Each slice contains 4 LUT, 4 FF, arithmetic logic gates, multiplexers and fast look ahead carry chain. The Virtex 5 device used to configure the resources of HRCS such as soft core processor, RLUs and Memory with the necessary communication ports. The device utilization for configuration of various modules in HRCS summarized in table 2

.

Table 2. Virtex 5 FPGA (XC5VLX110T) Device utilization chart

Resource	Module	No. slices
Soft core (MicroBlaze)	MicroBlaze	1599
Hard core (Reconfigurable Logic Unit)	RLU1	500
	RLU2	500
	RLU3	500
Memory	DDR2_SDRAM (256MB)	1687

	dlmb_cntlr (8KB)	7
	Ilmb_cntlr (8KB)	4
Communication Interfaces (Bus controllers)	dlmb	1
	ilmb	1
	Mb_plb	96
Debug Module	Mdm (64KB)	97
Timing and reset circuits	proc_sys_reset	30
	xps_timer (64KB)	187
I/O interfaces	RS232_uart (64KB)	97
	DIP_Switches_8bit (64KB)	67
	LEDs_8bit (64KB)	71
Heterogeneous Reconfigurable Computing System		5,444

The table 2 depicts the number of slices used for each module in HRCS and the HRCS consumed 5,444 slices on virtex 5 FPGA. In coming sections we will demonstrate the tasks distribution to targeted HRCS architecture and their performance.

6.2 Performance Estimation of application targeted to HRCS

Initially, the real time application JPEG is taken as input to analyze the effectiveness of the task distribution methodology. So, the JPEG is represented as task graph and their dependencies are shown in figure 8.

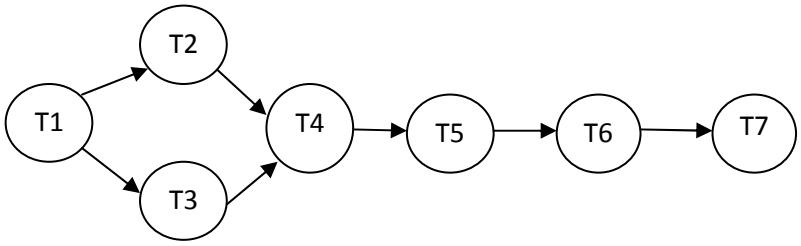


Figure 8. Task graph of JPEG

As stated in figure 7, the behavior of the tasks in JPEG is described in C++ and also in HDL in order to execute them on soft core and hard core processing elements. The C++ code of the task is cross compiled to the soft core processor MicroBlaze and that generates executable file. The executables of the tasks are stored in program memory and then executed on MicroBlaze in order to acquire attributes, i.e. ececution time, of the tasks on MicroBlaze. Similarly, HDL code of the tasks is synthesized for the device Xilinx Virtex 5 to generate gate level netlist and that produces configuration files required for task execution on FPGA. These configuration files are stored in memory and configured them to FPGA and then executed in order to acquire the attributes, like area required and execution time, of the tasks on FPGA. The acquired attributes of JPEG task graph (shown in figure 6) is shown in table 3 where the task in a task graph in first column, level of the tasks in second column, area required on FPGA in third column, tasks execution time on FPGA in forth column and the execution time of tasks on soft core processors, i.e. core 2 duo and MicroBlaze, in column 5 and 6. Finally, the column 7 represents the enhancement in computational complexity of MicroBlaze compared to GPP i.e Core 2 duo Processor.

Table 3: Attributes of JPEG Task graph on HRCS

Tasks in task graph	Level of the tasks	Hardware ET		Software ET		Enhancement in computational complexity on MicroBlaze
		Area required (slice LUT)	Execution time(ns)	Core 2 duo [1.2GHz] (µsec)	(MicroBlaze) [125MHz] (msec)	
Gray conversion (T1)	1	64	4.7	1.1	0.4	37.87
Matrix Transpose (T2)	2	64	4.372	1	0.4	41.67
Wrapper 1(T3)	2	128	6.081	10	6	62.5
Wrapper 2 (T4)	2	128	6..081	10	6	62.5
Quantization (T5)	3	64	5.259	3	2.6	90.28
Encoder (T6)	4	192	14.32	20	7.3	38
Memory Read/Write (T7)	5	64	4.372	1	0.4	41.67
JPEG	-	-	45.185	46.1	23.1	52.19

The table 3 shows execution time of the application JPEG on soft core processor and hardcore processor. From the table, it is clear that the performance of the JPEG task graph is effective on FPGA reconfigurable area compared to soft core processor. The computation time of the application is more on MicroBlaze compared to Core 2 Duo processor. The computation time (number of CPU clock cycles) of an application is equal to multiplication of clock speed of the processor and execution time of that application on the processor. So, the custom soft core MicroBlaze takes more computation time compared to general purpose Core 2 duo soft core processor. The comparisons between the soft cores in terms of computation time and number of clock cycles are shown in figure 9.

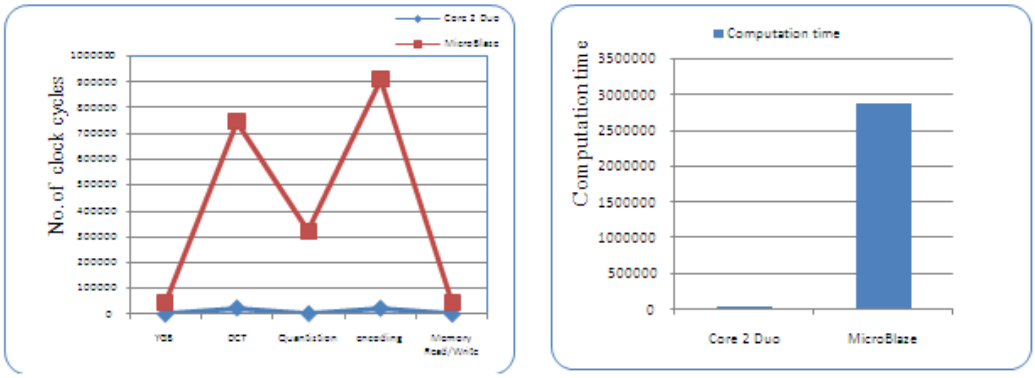
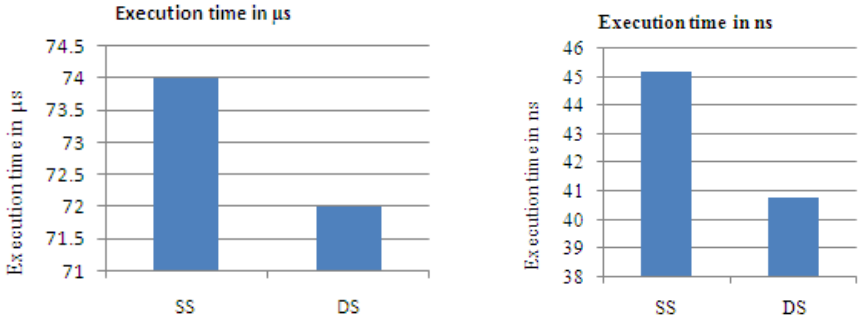


Figure 9 (a) No. of clock cycles consumed for execution of JPEG (b) Computation time of the application JPEG on MicroBlaze and Core 2 duo processor

The figure 9 depicts that the JPEG execution is effective on Core 2 Duo compared to MicroBlaze but the MicroBlaze provides cost effective solution in terms of power and area. Since, the core 2 duo runs at 1.2Ghz, whereas MicroBlaze runs at 125MHz, the core 2 duo dissipates more power and also consumes more area. So, the MicroBlaze would be the soft core processor to design low power computing platform for real time applications.

The task distribution methodologies distribute the tasks of an application statically as well as dynamically to the resources of computing platform. As we have stated, the both task distribution methodologies are implemented and verified for the represented task graphs. In this research, the dynamic task distribution methodology follows the MLF algorithm for task distribution. Initially, hypothetical task graph shown in figure 2 is distributed statically as well dynamically to the platform HRCS which is configured on FPGA and then the JPEG task graph is distributed for execution. The execution time, in both static and dynamic scheduled policies, of hypothetical task graph and JPEG task graphs on the platform HRCS is shown in figure 10.



(a) (b)

Figure 10 (a) Execution time of hypothetical task graph on HRCS (b) Execution time of JPEG task graph on HRCS

From the figure 10, it is clear that the execution time of the task graphs is less in dynamic scheduling policy compared to static scheduling policy. So the task distribution with dynamic scheduling enhances the application execution and also utilizes the resources of computing platform HRS effectively. The real time application JPEG alone is small enough to effectively utilize the HRCS resources. So, an advanced application OFDM also considered along with JPEG and distributed to HRCS in order to utilize the resources effectively. The task graph of OFDM transmitter consists of 6 tasks as shown in figure 11. The tasks in OFDM transmitter are source generator (T1), serial to parallel (S/P) converter (T2), Quadrature Amplitude Modulation (QAM) a constellation mapper (T3) , IFFT (T4), parallel to serial (P/S) converter (T5) and cyclic prefix insertion blocks (T6).

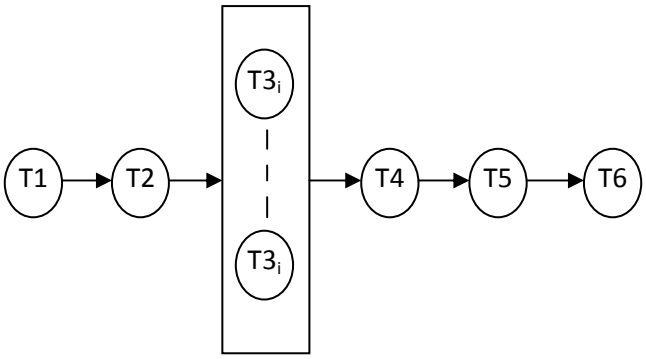


Figure 11. OFDM Transmitter Task graph

The task 3 is QAM and it could perform 16 QAM operations to generate 16 symbols which are further transmitted to 16 bit IFFT. The tasks, in OFDM transmitter, are executed individual on

MicroBlaze as well on FPGA and captured their attributes. The attributes of OFDM transmitter task graph are listed in table 4. Form the table 4, it is clear that the hardware realization enhances the OFDM transmitter execution. The execution speed of OFDM transmitter on MicroBlaze processor is slower than the general purpose processor and DSP processors. The core 2 duo and DSP processors dissipates more power since they run at 1.2GHz and 225 MHz speed and also bulky due to their heat sink. But, the MicroBlaze is available on the same chip FPGA where RLU resides and hence it would be effective soft core processor for many power sensitive real time applications.

Table 4. Attributes of OFDM Transmitter

Tasks in Task Graph	Hardware Realization Virtex-5 FPGA		Software realization MicroBlaze @ 125MHz)	Core 2 Duo @ 1.2GHz	TMS320C6713 DSK @225MHz
	Execution Time (ns)	Area (No. of slices)	Execution Time (µs)	Execution Time (µs)	Execution Time (µs)
Binary generator (T1)	576	192	2191.7	500	1270
S/P conversion(T2)	1.216	13	192.4	300.5	8.012
16-QAM(T3)	6	16	128.5	4.5	0.864
16-IFFT(T4)	8.646	334	444.0	143.5	142.20
P/S conversion(T5)	4.14	74	192.4	300.5	8.012
Cyclic Prefix (T6)	4.73	1917	382.3	6	8
OFDM Transmitter	600.73	2546	30012.3		

So the tasks, in OFDM transmitter task graph, are distributed to HRCS platform statically as well dynamically. The execution time of the OFDM transmitter on the platform HRCS is shown in figure 12. Finally both JPEG and OFDM transmitter task graphs together distributed to the HRCS platform and the execution time in both static scheduling and dynamic scheduling policy is shown in figure 13.

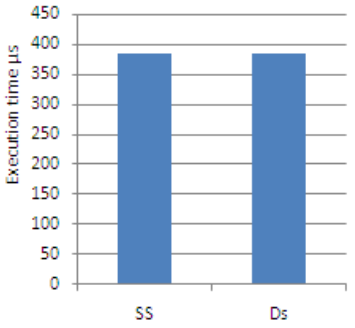


Figure 12. Execution Time of OFDM Transmitter

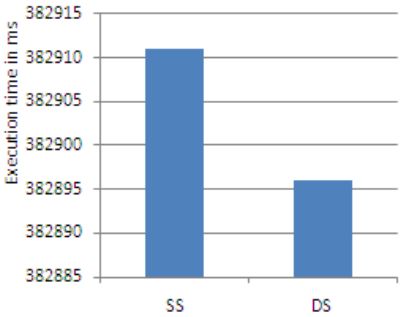


Figure 13. Execution Time of JPEG & OFDM together

The figure 12 shows that the execution time of OFDM transmitter and it is same in both Static and Dynamic scheduling policies because there are no parallel tasks in the task graph of OFDM transmitter. The figure 13 shows the execution time when both JPEG & OFDM transmitter task graphs together distributed to the HRCS platform. From the figures 10, 12 and 13, the dynamic

task distribution policy with MLF algorithm enhances the execution time compared to static task distribution policy in [15] [20]. The application execution time difference between dynamic scheduling policy and static scheduling policy depends on the task parallelism in the task graph and execution time of the parallel tasks. The resource utilization of HRCS in both task distribution scenarios would be demonstrated in coming section.

6.3 Resources utilization

The resources utilization of the platform HRCS is estimated based on the tasks allocated to individual resources of the platform and time spent in executing the tasks. The expression used to calculate the resources utilization is as follows. $\boldsymbol{Resource\ utlization} = \frac{\sum \boldsymbol{n \times et}}{\boldsymbol{N \times ET}}$, where n = number of resources active in a time slot et, et = task execution time slot, N = Total number of resources in computing platform, ET = Total execution time of an application. The resource utilization is calculated for both JPEG and OFDM transmitter task graphs targeted to the platform HRCS. The JPEG utilized 26.7% of HRCS resources when tasks are distributed statically whereas it is 28.9% when they are distributed dynamically. Similarly, the OFDM utilize 25% of the resources in both task distribution policies due to lake of parallelism in OFDM transmitter task graph. The resource utilization is 25% in the case of both JPEG and OFDM transmitter task graphs together distributed to the platform.

7. CONCLUSION

In this paper, we have proposed a novel dynamic task distribution methodology named as MLF algorithm for on chip heterogeneous computing platform. An on chip Heterogeneous Reconfigurable Computing System (HRCS) is constructed on Virtex 5 FPGA device for application execution. The HRCS contains MicroBlaze as soft core processor and multiple RLUs configured on FPGA as hardcore processor. The real time applications JPEG and OFDM transmitter task graphs design parameters are obtained by executing them on the resources of HRCS. The obtained attributes of task graphs acts as cost functions in MLF algorithm and the task graphs are distributed statically as well as dynamically to the resources of HRCS. From the result, it is concluded that the application execution is effective on HRCS when parallel tasks are targeted. The MLF dynamic task distribution methodology enhanced the execution speed of the JPEG applications by 9.6% over static task distribution. On the other side, the execution of the task graphs takes less time on GPP and DSP processors compared to the On-chip soft core processor MicroBlaze but GPP and DSP processors dissipates more power due to their high clock speeds. So the MicroBlaze can acts as soft core processes in design of low power computing platforms for real time applications.

REFERENCES

[1] Reiner Hartensstein, “Microprocessor is no more General Purpose: Why future reconfigurable platforms will win” invited paper of the International conference on Innovative Systems in silicon.ISIS’97, Texas, USA, pp 1- 10, October 8-10, 1997.

[2] D. Wang, S. Li and Y. Dou, Loop Kernel Pipelining Mapping onto Coarse-Grained Reconfigurable Architecture for Data-Intensive Applications, in Journal of Software, Volume 4, no-1,p81-89, 2009.

[3] Maisam M. Bassiri and Hadi Sh. Shahhoseini, Online HW/SW partitioning and co-scheduling in Reconfigurable Computing Systems, in 2nd IEEE International Conference on Computer Science and Information Technology, 2009, ICCSIT 2009,pp 557-562.

[4] J Lyke, Reconfigurable Systems: A generalization of Reconfigurable computational strategies for Space Systems, IEEE Aerospace conference Proceedings, vol. 4, pp 4-1935, 2002.

[5] David B. Stewart, Pradeep K. Khosla "Real time Scheduling of Dynamically Reconfigurable Systems" In Proceedings of the IEEE International Conference on Systems Engineering, Dayton Ohio, pp. 139-142, August 1991.

[6] Liang LIANG, Xue-Gong ZHOU, Ying WANG, Cheng-Lian PENG, Online Hybrid Task Scheduling in Reconfigurable Systems, in Proceedings of the 11th International Conference on Computer Supported Cooperative Work in Design, pp 1072 – 1077, in 2007.

[7] Proshanta Saha, Tarek El-Ghazawi, Software/Hardware Co-Scheduling for Reconfigurable Computing Systems, in International Symposium on Field-Programmable Custom Computing Machines, pp 299-300, 2007.

[8] Solomon Raju Kota, Chandra Shekhar, Archana Kokkula, Durga Toshniwal, M. V. Kartikeyan and R. C. Joshi, "Parameterized Module Scheduling Algorithm for Reconfigurable Computing Systems" in 15th International Conference on Advanced Computing and Communications, pp 473-478,.2007.

[9] Mahmood Fazlali, Mojtaba Sabeghi, Ali Zakerolhosseini and Koen Bertels, Efficient Task Scheduling for Runtime Reconfigurable Systems, Journal of Systems Architecture, Volume 56, Issue 11, Pages 623-632, November 2010.

[10] Yun Wang and Manas Saksena, Scheduling Fixed –priority Tasks with Preemption Threshold, in Sixth International Conference on Real-Time Computing Systems and Applications, RTCSA '99, pp 328-335, 1999.

[11] Ali Ahmadinia, Christophe Bodda and Jurgen Teich, A Dynamic Scheduling and Placement Algorithm for Reconfigurable Hardware, ARCS 2004, LNCS 2981, pp. 125 – 139, 2004.

[12] Xue-Gong Zhou, Ying Wang, Xun-Zhang Haung and Cheng-Lian Peng, On-line Scheduling of real time Tasks for Reconfigurable Computing System, International Conference on Computer Engineering and Technology, PP 59-64, 2010.

[13] Maisam Mansub Bassiri and Hadi Shahriar Shahhoseini, A New Approach in On-line Task Scheduling for Reconfigurable Computing Systems, in proceedings of 2nd International Conference on Computer Engineering and Technology, pp. 321-324, April 2010.

[14] Klaus Dane and Marco Platzner, A Heuristic Approach to Schedule Real-Time Tasks on Reconfigurable Hardware, in proceedings of International Conference on Field Programmable Logic and Applications, pp 568 – 578, 2005.

[15] Zexin Pan and B. Earl Wells, Hardware Supported Task Scheduling on Dynamically Reconfigurable SoC Architectures, IEEE Transactions on VLSI Systems, vol. 16, No. 11, November 2008.

[16] Mojtaba Sabeghi, Vlad-Mihai Sima and Koen Bertels, Compiler Assigned Runtime Task Scheduling on A Reconfigurable Computer, International Conference on Field Programmable Logic and Applications, 2009 (FPL 2009) pp 44 – 50, September 2009.

[17] Yi Lu, Thomas Marconi, Koen Bertels and Georgi Gaydadjiev, A Communication Aware Online Task Scheduling Algorithm for FPGA-based Partially Reconfigurable Systems, 2010 18th IEEE Annual International Symposium on Field Programmable Custom Computing Machines, pp 65-68, sept.2010.

[18] Juanjo Noguera and Rosa M. Badia, HW/SW Co-design Techniques for Dynamically Reconfigurable Architectures, IEEE Transaction on Very Large Scale Integration (VLSI) Systems, Vol. 10, No. 4, pp 399 – 415, August 2002.

[19] Boris Kettelhoit and Mario Porrmann, A layer Model for Systematically Designing Dynamically Reconfigurable Systems, International Conference on Field Programmable Logic and Applications, pp. 1-6, August 2006.

[20] Haluk Topcuoglu, Salim Hariri and Min-You Wu, Performance Effective and Low-Complexity Task Scheduling for Heterogeneous Computing, IEEE Transactions on Parallel and Distributed Systems, Vol. 13, No. 3, pp. 260 – 274, March 2002.

[21] Mohammad I. Daoud and Nawwaf Kharma, A High Performance Algorithm for Static Task Scheduling in Heterogeneous Distributed Computing Systems, Journal of Parallel and Distributed Computing, Vol. 68, no. 4, pp. 299-309, April 2008.

[22] S. Darba and D.P. Agarwal, Optimal Scheduling Algorithm for Distributed Memory Machines, IEEE Trans. Parallel and Distributed Systems, Vol. 9, no. 1, pp. 87-95, Jan. 1998.

[23] C.I. Park and T.Y. Choe, An Optimal Scheduling Algorithm Based on Task Duplication, IEEE Trans. Computers, Vol. 51, no. 4, pp. 444-448, Apr. 2002.
[24] Y.K. Kwok and I. Ahmad, Dynamic Critical Path Scheduling: An effective Technique for Allocating Task Graphs to Multiprocessors, IEEE Trans. Parallel Distributed Systems, Vol. 7, no. 5, pp. 506-521, May 1996.
[25] An Improved Duplication Strategy for Scheduling Precedence Constrained Graphs in Multiprocessor Systems, IEEE Trans. Parallel and Distribution Systems, Vol. 14, no. 6, June 2003.
[26] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K.L.M Bertels, G. K. Kuzmanov and E. M. Panainte, The Molen Polymorphic Processor, IEEE Transaction on Computers, Vol. 53, Issue 11, pp. 1363-1375, November 2004.
[27] John Lehoczky, Lui Sha and Ye Ding, The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average case behavior, Proceedings of Real Time Systems Symposium, pp. 166-171, Dec. 1989.
[28] C. L. Liu and James W. Layland, Scheduling algorithms for multiprogramming in a Hard Real Time Environment, Journal of ACM, Vol. 20, no. 1, pp. 46-61, 1973.
[29] Wei Zhao, Krishivasan Ramamrithm and John A. Stankovic, Scheduling tasks with Resource requirement in Hard Real Time Systems, IEEE Trans. Software Engineering, Vol. SE-13, No. 5, May 1987.
[30] Maisam M. Bassiri and Hadi Sh. Shahhoseini, A HW/SW Partitioning Algorithm for Multitask Reconfigurable Embedded Systems, IEEE International Conference on Microelectronics, 2008.
[31] Proshanta Saha and Tarek El-Ghazawi, Extending Embedded Computing Scheduling Algorithms for Reconfigurable Systems, 2007 3rd Southern Conference on Programmable Logic, pp 87 – 92, February 2007.
[32] Mahendra vucha & Arvind Rajawat, An effective Dynamic Scheduler for Reconfigurable High Speed Computing Systems, IEEE International Advance Computing Conference, pp. 766 – 773, February - 2013.
[33] Juan Antonio Clemente, Javier Resano, and Daniel Mozos, An approach to manage reconfigurations and reduce area cost in hard real-time reconfigurable systems, ACM Trans. Embedded Computing Systems, Vol. 13, No. 4, February 2014.
[34] Mahendra Vucha and Arvind Rajawat, A Case Study: Task Scheduling Methodologies for High Speed Computing Systems, International Journal of Embedded Systems and Applications(IJESA), Vol. 4, No. 4, December 2014.
[35] E. Ilavarasan P. Thambidurai, R. Mahilmannan, Performance Effective Task Scheduling Algorithm for Heterogeneous Computing Systems, Proceedings of the 4th International Symposium on Parallel and Distributed Computing, 2005.

AUTHORS

Mahendra Vucha received his B. Tech degree in Electronics & Comm. Engineering from JNTU, Hyderabad, India in 2007 and M. Tech degree in VLSI and Embedded System Design from MANIT, Bhopal, India in 2009. He is currently working for his PhD degree at MANIT, Bhopal, India and also working as Asst. Professor in Christ University, Dept. of Electronics and Communication Engineering, Bangalore (K.A), India. His areas of interest are Hardware Software Co-Design, Analog Circuit design, Digital System Design and Embedded System Design.

Arvind Rajawat received his B.E degree in Electronics & Communication Engineering from Govt. Engineering College, Ujjain, India in 1989, M. Tech degree in Computer Science Engineering from SGSITS, Indore, India in 1991 and Ph. D degree from MANIT, Bhopal, India. He is currently working as Professor in Dept. Electronics and Communication, MANIT, Bhopal (M.P), India. His areas of interest are Hardware Software Co-Design, Embedded System Design and Digital VLSI Design.