6-1-2010

Optimizing Reconfigurable Hardware Resource Usage in System-on-a-Programmable-Chip with Location-Aware Genetic Algorithm

Sin Ming Loo
Boise State University

JingXia Wang
ShenZhen Polytechnic

Optimizing Reconfigurable Hardware Resource Usage in System-on-a-Programmable-Chip with Location-Aware Genetic Algorithm

Sin Ming Loo[*],
Boise State University, Boise, Idaho 83725, USA

JingXia Wang[†]
ShenZhen Polytechnic, ShenZhen, CHINA

Abstract

This paper presents static task scheduling using location-aware genetic algorithm techniques to schedule task systems to finite amounts of reconfigurable hardware. This research optimizes the use of limited reconfigurable resources. This scheduling algorithm is built upon our previous work [12-14]. In this paper, the genetic algorithm has been expanded to include a feature to assign selected tasks to specific functional units. In this reconfigurable hardware environment, multiple sequential processing elements (soft core processors such as Xilinx MicroBlaze [22] or Altera Nios-II [1]), task-specific core (application specific hardware), and communication network within the reconfigurable hardware can be used (such a system is called system-on-a-programmable-chip, SoPC). This paper shows that by pre-assigning (manually or randomly) a percentage of tasks to the desired functional units, the search algorithm is capable of finding acceptable schedules and maintaining high resource utilization (>93 percent, with two processors configuration).

Key Words: FPGA, scheduling, hardware/software codesign, reconfigurable hardware.

1 Introduction

Scheduling algorithms, whether static or dynamic, have been designed around an available target system (usually constructed ahead of time) that is made up of a processor, application-specific integrated circuits, and programmable hardware connected together using some form of bus or switch interconnection network. Many systems extensively utilize off-the-shelf processors and dedicated hardware to perform their intended function. The hardware remains fixed from the time of its fabrication. The flexibility in the system is restricted to the software portion of the system. With the increased popularity and availability of reconfigurable hardware in the late 90s, the hardware itself has become flexible. The reconfigurable environment targeted in this research is one in which the application can dictate the structure of the processor, the high-speed logic sections, and the interconnection medium. With all functional units and an interconnection network embedded in one reconfigurable device, this is called system-on-a-programmable-chip (SoPC). The goal of SoPC is to exploit the synergism that is possible when both the hardware and software portions of the design are performed concurrently and cooperatively.

Reconfigurable hardware has been progressively replacing application-specific hardware in small volume designs. The use of reconfigurable hardware allows the system to be re-designed and upgraded without non-recurring engineering costs because the system can be reconfigured in the field after deployment. This flexibility allows the hardware structures for given portions of an application to be specialized and optimized to achieve a performance that can be orders of magnitude greater than that which can be achieved within most traditional processing systems employing a Von Neumann-style architecture. Unfortunately, reconfigurable resources within off-the-shelf reconfigurable hardware (number of pins for input/output, flip-flops, look-up tables, etc.) are limited. For many applications, this limitation means that it is impossible to configure the reconfigurable hardware such that all portions of the design are implemented for optimal performance (for example, optimize for speed). This is because such performance optimal implementations would probably consume resources orders of magnitude more than can be made available.

One desired compromise is to use the reconfigurable hardware by cognizant of the space/time trade-off. This compromise translates into determining how much concurrency should be employed in order to meet the performance requirements of the application without exceeding the resource limits of the reconfigurable hardware. The key to finding this effective balance is to develop techniques that can determine, within the confines of the resource limitations, which portions of the problem must have increased levels of concurrency to meet the overall performance constraints and which portions of the problem can be implemented sequentially to save room for the higher performing portions of the design. This is a resource constraint problem with an added twist to it. Because of reconfigurable hardware, the technique is able to determine which functional unit (trade-offs of the use of processor core versus application-specific hardware) should be employed in order to meet the overall design and resource constraints.

[*] Department of Electrical and Computer Engineering,. Email: smloo@boisestate.edu.

[†] Department of Electrical Engineering.

The remainder of this paper is organized as follows. First, a quick survey of reconfigurable hardware utilization research is presented. Then, an overview of reconfigurable system design framework is given. The reader is introduced to the search space complexity of this SoPC scheduling problem. An overview of genetic algorithm is presented. An example, extracted from Space Shuttle Turbo Pump, is used for scheduling discussion. The next section discusses how the percentage of pre-assigned tasks can influence the scheduling solutions. This expanded set of simulations has been accomplished using synthetic task systems. Finally, some general conclusions are presented.

2 Previous Research

The research into methods to take advantage of reconfigurable hardware has been concentrated in areas of scheduling algorithm, operating system, compiler techniques, and dynamic reconfiguration techniques. An earliest deadline first scheduling technique is used to schedule tasks onto reconfigurable hardware [5]. In this research, the target reconfigurable hardware is partitioned into slots. The paper reported finding feasible schedules with system utilization of up to 70 percent. The use of state feedback control has also been presented [20]. Embedded operating systems have been designed and implemented to manage reconfigurable resources. Basically, this runtime system performs online task and resource management [4, 19]. The use of the operating system allows dynamic scheduling and dynamic placement of hardware tasks into reconfigurable hardware.

Another set of methods to take advantage of reconfigurable hardware has been borrowed from the compiler world. Resano [17] has developed pre-fetch and replacement techniques to reconfigure the hardware dynamically. Their techniques manage the resources by exploiting a novel encoding scheme. The technique developed can validate the feasibility of the scheduling/placement quickly, increase resource utilization, and improve the parallelism [16]. Yet another study [13] looks into how loop unrolling can take advantage of reconfigurable resources. This research shows that significant performance improvements can be achieved through combining both intra- and inter-task parallelism.

Numerous papers presented in the area of reconfigurable hardware utilization research described partial or dynamic reconfiguration techniques [2, 10-11, 15]. The goal of such techniques is using partial or dynamically reconfiguring technique to share the hardware in time. However, as of the writing of this paper, partial/dynamic reconfiguration time is in the order of milliseconds, which makes the applicable of these techniques questionable in real world reconfigurable hardware. These techniques, coupled with an embedded operating system, can be very powerful in assigning the reconfigurable hardware for task execution.

Others are looking into how task placement can be optimized for computation and, at the same time, decrease energy usage [2, 9-11, 15]. In one case, a genetic algorithm has been designed to minimize both task execution schedule length and power consumption [15].

The research presented in this paper concentrates on well-defined task systems. The goal is to find a feasible schedule within confined reconfigurable resources. The task system selected in our simulation will require more than twice of that provided (in the simulation). The genetic algorithm determines the placement of tasks and what task-specific logics will be implemented. The algorithm determines the configuration of reconfigurable hardware without the use of dynamic reconfiguration. Thus, this research is a much more limited domain of the reconfigurable scheduling research.

In this paper, reconfigurable hardware is used to implement both the high-speed logic that has been designed to execute the most time-intensive portions of the application problem, and traditional Von Neumann-style processing cores to save valuable hardware resources. In this arrangement, the processsor cores, the application specific modules, the input/output logic, and the routing among each of these hardware entities are contained within the finite resources of the reconfigurable logic. The trade-off is to determine the number and types of each of these entities that will best meet the needs of the application and fit within the available reconfigurable hardware resources. Simply placing all the functionality in application-specific modules will probably never represent a valid solution because of the finite resource constraints. Conversely, placing all the functionality in a single large processing core unit that will be implemented within the reconfigurable logic is also not desirable, since it is subject to poor performance. The goal of this paper is to show solutions to this space/time trade-off in those cases where the application can be decomposed into a well-behaved system of tasks that can be implemented directly in hardware or executed sequentially on one or more processing cores. This paper also demonstrates a genetic algorithm implementation that allows certain tasks to be assigned either randomly or as specified by the user.

3 Reconfigurable System Design Framework

The components of reconfigurable system design framework (RSDF) are shown in Figure 1. The heart of this framework is the scheduler. Inputs to the scheduler include the hardware library, task system, resource and design constraints. The hardware library supports three types of functional units that can be placed in reconfigurable hardware. There are also inputs that reflect the hardware resource limits associated with the reconfigurable hardware medium, and various design constraints that reflect the required performance of the application. The purpose of the scheduler is to generate a complete task schedule and a system-on-a-programmable-chip (SoPC) high-level hardware system description that satisfies all of the given constraints. The task execution schedule describes the task execution sequence of the system from a global point of view for a single major frame of execution. It does so in a manner that satisfies the precedence and resource constraints. The high-level hardware system description is created at the same time the task schedule is generated. It consists of the number and type of core processors to be implemented and the inter-reconfigurable logic communication

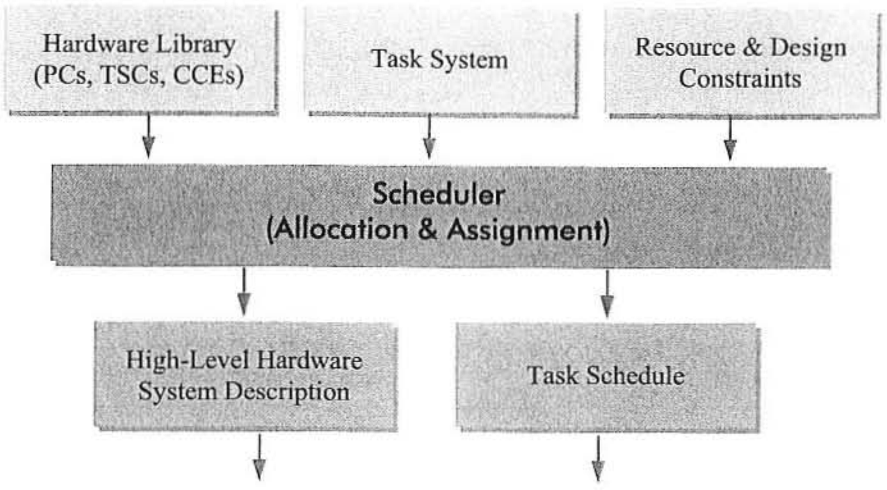

Hardware Library (PCs, TSCs, CCEs)
Task System
Resource & Design Constraints
Scheduler (Allocation & Assignment)
High-Level Hardware System Description
Task Schedule

Figure 1: Reconfigurable system design framework. This framework defines the elements for reconfigurable resources to be utilized efficiently. A complete task execution schedule and SoPC description are produced so that a system can be implemented.

topology of the system. In the following sections, the characteristics of these various elements are discussed.

3.1 Hardware Library

The hardware library represents a high-level description of the candidate logic modules that can be used to implement an application. It supports three types of functional units that can be placed in reconfigurable hardware. These functional units include the processor cores (PCs), task specific cores (TSCs), and communication core elements (CCEs).

In this model, PCs represent distinct elements in the hardware library because they have the general capability to support the Von Neumann-style sequential execution of more than one task. In general, the number of tasks that they can execute is limited by the internal program and data memory present within the PC. This is because the model assumes that all memory elements are explicitly specified as part of the PC-type definition in the hardware library. This means that there is an added dimension to the resource utilization problem. Each PC uses a fixed amount of hard resources every time an instance of it is implemented in the reconfigurable hardware. Some of these hardware resources are used for internal program and data storage. The amount of program/data storage thus in effect becomes a "soft" resource limit that will directly affect the number and type of tasks that the PC can execute. This is because each task in the system has assigned to it a projected "soft" resource usage requirement for each type of PC that is present in the hardware library.

TSCs are another type of functional unit that may be present in the hardware library. Unlike PCs they are not general purpose in nature, but perform the specific function that is associated with the task. CCEs are the final type of functional unit present in the hardware library. The model supports both synchronous (buffered) and asynchronous (non-buffered) CCEs.

It is assumed that the functional units themselves utilize a common asynchronous protocol and dedicated communication ports to communicate with each other. Synchronous links between functional units are composed of CCEs that are primarily made up of routing resources. Asynchronous communication is made possible by incorporating buffered communication elements. In this way, the interface between PCs and TSCs is uniform regardless of whether synchronous or asynchronous communication elements are used.

3.2 Task System

The other input to the reconfigurable system design framework is the task system, where the application task structure and performance information as well as the soft resource requirements for each task are maintained. For this portion of the model, it is assumed that the application problem has been decomposed into a set of tasks that can execute in a deterministic manner as software processes on the sequential processing units or as hardware functions within the reconfigurable logic. These tasks are considered to be well-defined in that the execution time can be determined at the time of task creation for all software and hardware manifestations. Also, all tasks are considered to be non-preemptive in nature. In this scenario, the edges in the task system contain both data and control flow information, which guarantee the correct system operation. In this work, it is assumed that a well-defined system can be unrolled into a directed acyclic graph (DAG) part and a communication part.

3.3 Resource and Design Constraints

The third set of inputs to a reconfigurable system design framework is the design constraints. There are two types of constraints. The first is the amount of hardware resources that are available for use in the reconfigurable medium. This is the global size constraint. The second constraint specifies the level of performance that the system must possess. This is the global timing constraint -- it is in effect the maximum acceptable length of the schedule. This constraint is mandatory in real-time systems where it can be viewed as representing the global deadline associated with the major frame of the application's task system. In general, the more stringent the performance requirement (i.e., the shorter the required schedule length) or the smaller the global resource constraint, the harder it will be to create an acceptable static schedule and a SoPC high-level hardware description capable of fitting within the finite resources of the reconfigurable medium.

3.4 High-Level Hardware System Description

One of the outputs produced by the scheduler is the high-level hardware system description (HLHSD). This description indicates both the number and type of processing cores, task cores, and communication core elements that are to be employed by the system and the interconnection structure that is used to interface the various functional units into a complete system. This representation can easily be translated for hardware synthesis into a structural representation of these components within a hardware description language.

3.5 Task Schedule

The second output produced by the scheduler is the task schedule. The task execution schedule contains the order of execution of the given tasks and the order of execution of the tasks within each PC. The schedule length is used to determine if the implementation will meet the mandated performance requirements specified in the design constraints.

4 Search Strategy and Scheduling Example

4.1 Genetic Algorithm

A standard genetic algorithm was implemented to permute the scheduling data structure, by treating it as the symbolic string to which genetic operations can be applied [7, 13]. In this scheme, each individual member of the population is reresented by a separate data structure, and the RES scheduling strategy acts as the fitness function. In the initialization step, a *population* of ζ structures is randomly initialized with task assignment and priority values. A task pre-assignment feature has been added to the scheduler to maintain designated task execution location as requested by the user. Each individual in the population represented by a separate data structure is called a *candidate*. The data structure represents a two-dimensional chromosome where each row contains the two genes that control task assignment and priority order. The initialization process initializes those tasks, pre-assigned as required by the user, and randomly creates and initializes the rest that make up the initial population. The makeup of the population continuously evolves over time. (The genetic algorithm implementation for this paper is described in previous research [13]; please refer to that paper for details of this implementation.) In this paper, we extended the previous research to include the concept of ask mobility factor. Each task is assigned a mobility factor, in this case randomly (between 1 and 100). During the scheduling and assignment simulations, we can set the mobility factor for each simulation. For example, if the mobility factor is set to 85 for a task system of 100 tasks, 15 tasks have been pre-assigned to their respective functional units. The genetic search algorithm will find a feasible schedule using the 85 tasks that haven't been assigned.

4.2 Example: Space Shuttle Turbo Pump Continuous Simulation Task System

We consider a simple application to illustrate how the location-aware scheduling in reconfigurable hardware can be applied to a real-world example (Figure 2). The particular example considered came from the general area of continuous or dynamic system simulation. Such simulations conform closely to the RSDF task system model outlined in Section 2. It is a system of non-preemptive tasks whose precedence relationships are irregularly structured. Tasks in these systems are governed by and-join precedence semantics and can easily be decomposed into a single major frame.

Continuous systems are generally described mathematically using a set of multi-order nonlinear differential equations that form a classical initial value problem. To solve this problem, the set of equations is often decomposed into an equivalent set of first-order differential equations that are solved in an iterative manner using numerical integration techniques. In this model, the state of the entire simulation is always a function of the variables that store the results of the integration. These state variables are given an initial value at the beginning of the simulation after which they are fed back to the next iteration of the simulation. In this model, each-iteration represents a major frame. The system is often modeled using additional sets of variables and equations that depend in some way upon the system state variables. These equations must be executed in a specific partial order within each major frame to ensure that all data dependencies between equations are always met. Depending upon the manner in which the tasks are defined, these variables are often used to transmit the data that must be communicated between the tasks during each major frame.

The specific example investigated is based upon one of the early models of the Space Shuttle Main Rocket Engine's (SSME) High Pressure Turbo Pump system [18, 21]. In this model, each task is defined as a major declared or state variable equation. This example is being used to show how tasks are pre-assigned to determine an execution schedule and a hardware configuration that satisfies the design constraints.

The task system for the SSME Turbo Pump as shown in Figure 2 contains 30 tasks. In this example, it was assumed that there were to be two types of PCs present in the hardware library. The execution time on PC type 1 was obtained by profiling an existing SSME Turbo Pump simulation on a 25 MHz T805 transputer system [18]. The execution time for each task was then divided by two because it was assumed that reconfigurable hardware could now support a 50 MHz T805 compatible PC. This appears to be a conservative estimation of performance considering the current state of FPGA

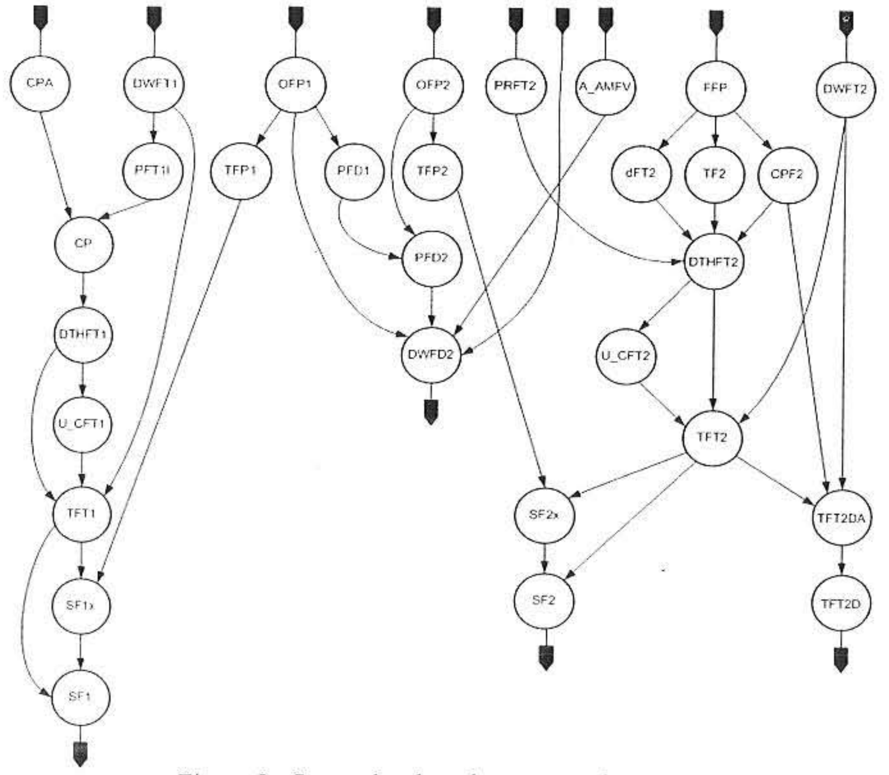


Figure 2: Space shuttle turbo pump task system

technology. The soft resource requirements for each task were estimated based upon the actual object code size of the software tasks when the simulation was compiled to run on the original T805 system. This is only a loose approximation. PC type 2 was derived directly from PC type 1. It was a version that was to have essentially the same execution characteristics as PC type 1, but with some additional hardware that accelerated the computation of a few select operations and reduced the soft resources required to complete these operations. The program/data memory resources of PC type 2 were assumed to be less than PC type 1. This made for an interesting trade off, since PC type 2 was a bit more powerful, but could support fewer tasks than PC type 1. Table 1 contains task execution times and required resources.

The hardware resource requirements for each TSC module were obtained in a somewhat arbitrary manner. They were synthetically generated by applying the $\upsilon = A\tau^x$, 2-D VLSI space/time trade-off equation presented earlier [8], where υ is a space/time trade-off constant, A is the hard resource utilization for an implementation, τ is the execution time of an implementation, and x is a uniformly generated random value that was in the range of 1 to 2. The procedure was to first randomly generate the TSC execution time under the constraint that the TSC execution time would be some value less than the fastest PC execution time. Then the resource utilization of the TSC was calculated using the $\upsilon = A\tau^x$ equation. The goal was to create a system that had a real-work derived structure that would be constrained in a manner where it is impossible for all task-specific components to exist within reconfigurable hardware at the same time.

This 30-task example is used to show how pre-assigned tasks can be beneficial in finding a better schedule with better schedule length. Using the task system as shown in Figure 2, a proper format text file was created for the RSDF tool. Forty simulations (with different random seed) were completed for each percentage of pre-assigned tasks. The results are shown in Table 2. The resource constraints were set in a way that no one functional unit can take on all the tasks. In fact, if all the tasks are to be assigned to just one functional unit, 350 percent of the resources will be needed.

Such resource constraints setup guarantees parallel processing where PC1, PC2, and TSCs will be used together. Table 2 shows the results when 0 percent, 10 percent, 20 percent, and 50 percent of the tasks are pre-assigned before using the RSDF tool to find a feasible and legal task schedule that will fit within the limited resources. For each row, 40 simulations (with different random seed) were completed. The table contains the minimum, average, and maximum schedule length of the 40 simulations.

We started the experiment by randomly pre-assigning the tasks. Our findings of these simulations showed that when none or a very small (less than 2 percent) number of tasks are pre-assigned, the schedule lengths found are better. When more tasks are pre-assigned, the schedule length increased. This is predictable as the number of pre-assigned tasks increases, the number of "good solutions space" decreases. Thus, worse schedules were found because the pre-assigned location may not be the best assignment from the viewpoint of schedule length and resource utilization. Another plausible explanation for the schedule length is that the 30-task system

Table 1: Shuttle turbo pump task information

Task name	PC1		PC2		TSC	
	Execution time (unit time)	Program memory required in bytes	Execution time (unit time)	Program memory required in bytes	Execution time (unit time)	FPGA resources required (CLB)
DTHFT1	72	2824	30	1158	5	4313
TFP2	17	727	17	727	12	2879
U_CFT2	2	77	2	77	2	17856
CPA	16	634	16	634	3	7423
OFP1	2	77	2	77	2	13354
CPF2	11	474	11	474	3	8682
PFT1I	5	180	5	180	2	11955
DTHFT2	73	2973	20	855	6	5822
CP	7	261	7	272	4	4779
DWFT1	16	626	16	626	13	1556
SF2x	2	83	2	83	2	16168
DWFT2	163	5954	80	3452	12	2003
TFP1	15	649	15	649	14	2442
SF2	3	112	3	112	1	40000
PFD1	17	634	17	634	13	1475
TF2	11	443	11	443	10	1475
SF1	3	122	3	122	2	10283
PRFT2	56	2280	56	2280	44	218
TFT2DA	4	152	4	152	2	12572
TFT1	11	431	11	431	10	1627
TFT2D	4	153	4	153	3	5265
dFT2	10	420	10	420	2	10400
DWFD2	56	2153	30	1177	13	809
TFT2	11	403	11	403	10	692
A_AMFV	20	786	20	786	8	1195
SF1x	2	85	2	85	2	15104
U_CFT1	2	87	2	87	1	40000
FFP	2	82	2	82	2	19980
OFP2	2	85	2	85	2	19742
PFD2	17	613	17	613	14	1505

Total resource usage if all tasks are to assign to: PC1: 24580, PC2: 17329, TSC: 281575
Total of the available resource in the simulation: PC1: 6000, PC2: 5000, TSC: 50000
If there is no resource limit, the schedule length of the critical path is: 67

Table 2: Pre-assigned turbo pump tasks example simulation results

% of tasks pre-assigned	Schedule length (40 simulations for each %)			**Note:** Pre-assigned tasks (task name, assigned functional unit)
	Min	Ave	Max	
0%	67	70.4	77	
10%	67	67.6	71	(DWFT2, TSC), (U_CFT1, PC2), (SF2, PC1)
20%	67	67.3	69	(DWFT2, TSC), (U_CFT1, PC2), (SF2, PC1), (OFP2, PC1), (DWFD2, TSC), (U_CFT2, PC1)
50%	73	78.5	86	(DWFT2, TSC), (U_CFT1, PC2), (SF2, PC1), (OFP2, PC1), (DWFD2, TSC), (U_CFT2, PC1), (CPA, PC2), (DWFT1, PC1), (A_AMFV, TSC), (FFP, PC2), (TFP1, TSC), (PFD1, PC1), (dFT2, TSC), (SF1x, PC1), (TFT2D, TSC)

[1] Number of generations=1000, population size=25, mutation probability=5%, recombination probability = 100%, 0-elitism, proportional-roulette-wheel selection. The average is for 40 cases.

has small solution space compared to a task system of 100 tasks. With the previous findings, we selectively pre-assigned tasks. By selectively pre-assigning tasks, better schedules (with better schedule length) can be found. The results are shown in Table 2.

4.3 Search Space Complexity

Precedence and resource constraint scheduling problems closest to the one being investigated in this work have been shown to be NP-Complete [6]. The scheduling problem addressed in this paper (and our previous publications [12-14]) is even more complex than most precedence and resource constraint scheduling problems. The complication is due to the use of scheduling theory to determine the configuration (determine the schedule length, assign tasks to functional units, and determine the resource utilization) of SoPC. The goal is to find a configuration (high-level hardware system description) that is realizable and has a schedule length that meets the design constraints.

The search space that must be transversed for any assignment or scheduling problem is extremely large. For example, just to find the optimum assignment of tasks to functional units in a system that conforms to the RSDF using exhaustive techniques requires m^n assignment operations, where m is the number of functional units to which a task can be targeted, and n is the number of tasks to be scheduled. This is the problem without placing constraints on where a task would be assigned. The scheduler will decide optimum placement during the scheduling process. Thus, in order to find an optimum assignment through an exhaustive search for a 100-task system assuming an active set of hardware elements that consists of three PCs and two TSCs per task, requires $(3+2)^{100} = 7.8886 \times 10^{69}$ operations. If each assignment operation can be completed in 0.5 nanoseconds, it would take 1.2507×10^{53} years to find such an optimum assignment within the available resources. This calculation does not include the hard and soft resource constraint check time or the time it takes to formulate a complete sequencing or scheduling of the tasks on the individual PCs. From this analysis, it is obvious that only a small subset of the search space can ever be transversed. The key is to utilize a technique that can perform this search in a highly efficient manner.

The version of the problem presented in this paper is where the designer can selectively place the desired tasks at their "optimum" execution location (pre-assigned!). The reasons for placement can be as simple as the functional unit having the suitable access to input/output interface or just that the designer knows such placement will result in a better overall SoPC configuration. With this pre-assigned scheduling feature added to the allocation and scheduling process, the complexity reduces to qm^n, where q is between 0 and 1. In this paper, we set q to 0.5, 0.8, 0.9, and 1.0; this is similar to pre-assigned 50 percent, 20 percent, 10 percent, and 0 percent of the tasks, respectively. With such constraint, the scheduler has a smaller legal search space and a better solution can be determined more quickly.

5 More Comprehensive Simulations and Results

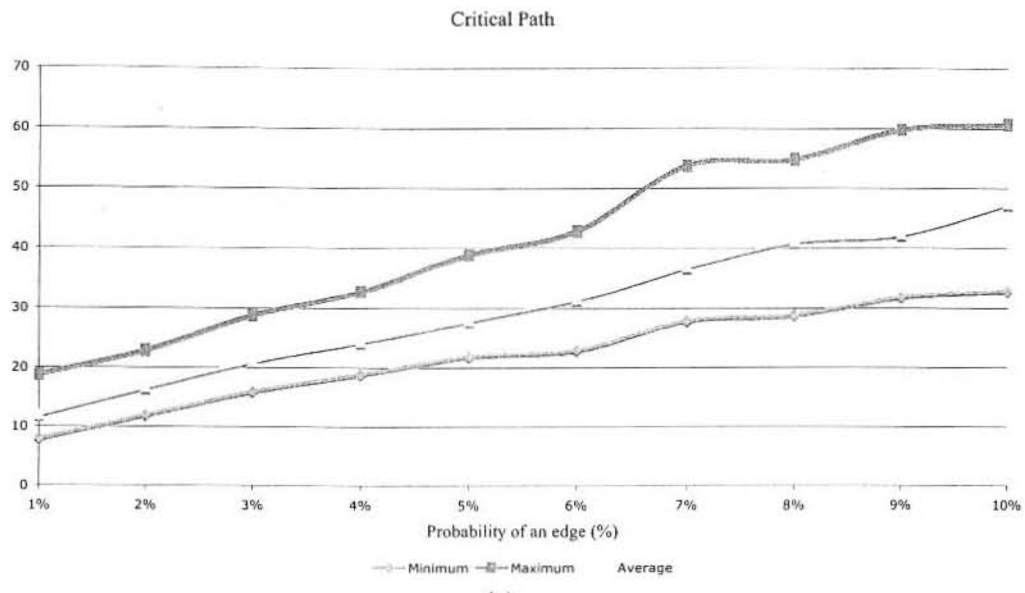
In the previous section, we used a 30-task system to show the working of the tasks pre-assigned technique. In this section, we use synthetic task systems to test how well the technique will stand up to pre-assigned tasks.

In parallel processing, it is common to evaluate the effecttiveness of competing assigning, mapping, and sequencing heuristics by applying a common set of randomly-generated task systems and comparing the performance of the resulting assignments or schedules in a statistical manner [12-14]. Using synthetic task graph generation techniques and parameters as described in [14], task systems were generated to test the effectiveness of our genetic algorithm implementation when some portion (0 percent, 5 percent, 10 percent, 15 percent, 20 percent, 25 percent, 30 percent, 35 percent, 40 percent, 45 percent, and 50 percent) of the tasks were pre-assigned.

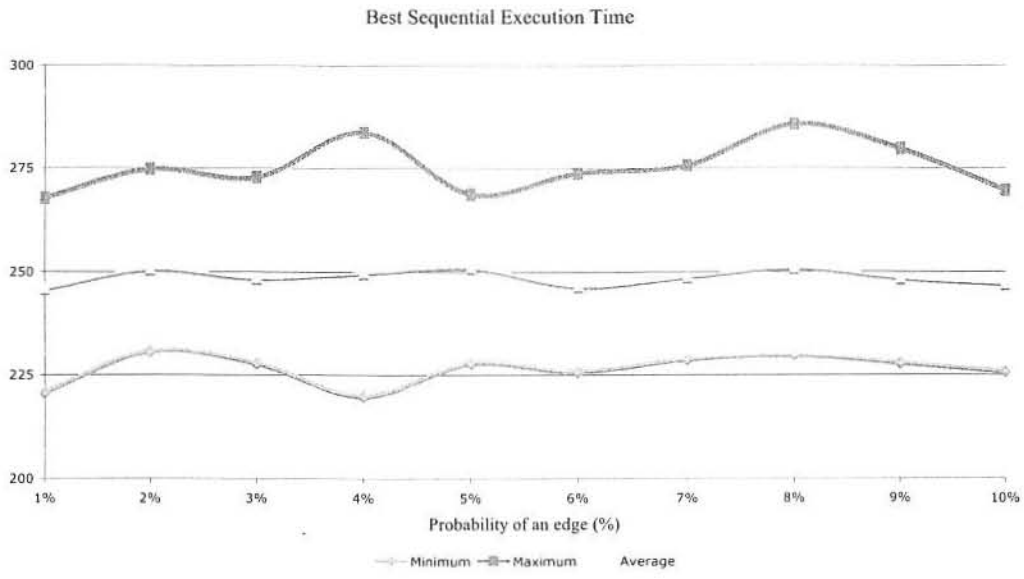
The genetic algorithm was used to find a feasible allocation within the available SoPC resources. Four hundred systems were generated with 100 tasks per task system (40 task systems for each edge probability). The task graph generation technique was presented in [14]. For each task system, eleven simulations were completed for each category of pre-assigned tasks (0 percent, 5 percent, 10 percent, 15 percent, 20 percent, 25 percent, 30 percent, 35 percent, 40 percent, 45 percent, and 50 percent). This meant that for each probability value, 440 simulations were completed. As for the target PCs, a configuration of two soft-processor cores (based on Xilinx Microblaze which utilizes 410 CLBs and 510 CLBs) were chosen. Each processor core consists of 4 Kbyte and 8 Kbyte of data memory, respectively. The number of CLBs used for the simulation was set to 15,304. It is noted that for the task systems to be implemented optimally, more than 2.5 times of the resources will be required. The resources constraint promotes space-time trade-off. The genetic algorithm searches through which functional unit should be used for each task. The characteristics of the task systems are shown in Figure 3. Figure 3(a) shows that as the probability of an edge increases between two nodes, the critical path time increases (or the best possible parallel schedule length increases). This critical path time is calculated by scheduling the tasks using as soon as possible algorithm without resources constraints. The best sequential time is for the tasks systems as shown in Figure 3(b). This is calculated by summing the shortest execution of each task using the optimal functional unit (again these numbers are determined without resources constraints being introduced). The implementing of such a schedule is not possible because it will require more than 250 percent of available resources.

The simulations were completed using an Apple PowerMac G5 (with Dual 2.5 GHz PowerPC G5 processors and 4 Gbyte of memory) running OS X 10.4.8. Each simulation took seven minutes with parameter settings as shown in Table 2. The simulations were set to find (optimize) the best schedule length within 1,000 generations (loops).

Figure 4 shows the best schedule length found. The results show that when 0 percent to 25 percent of tasks are pre-



(a)

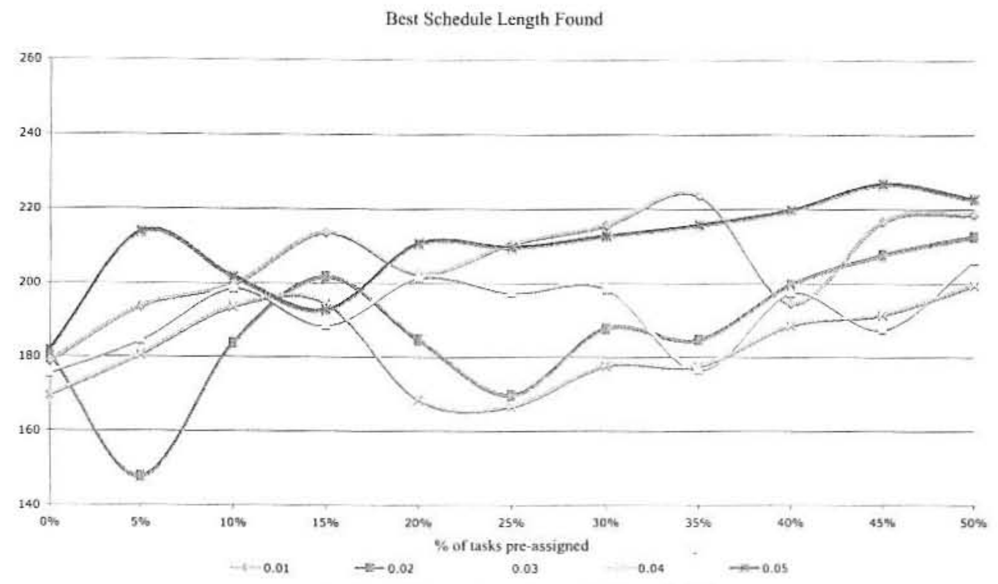


(b)

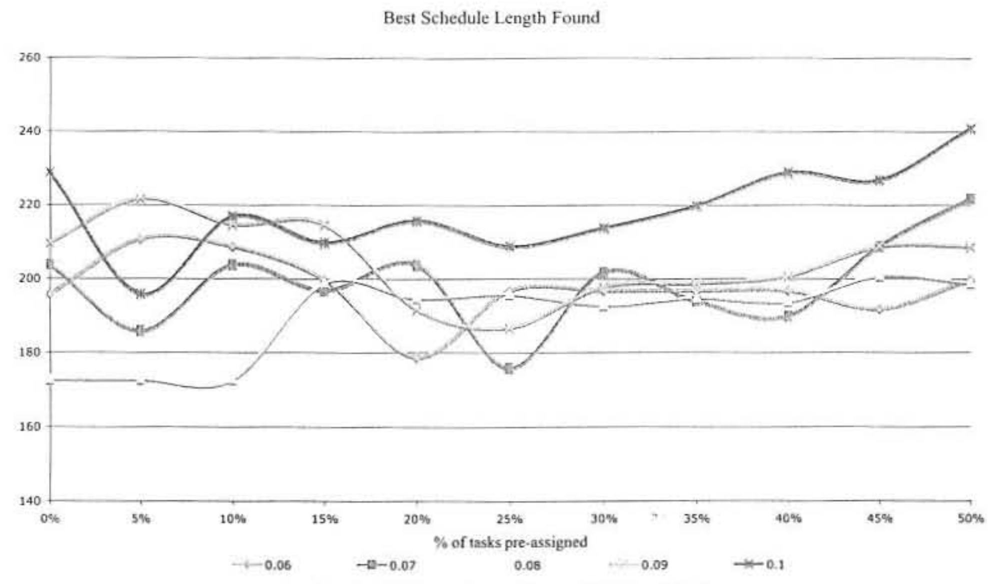
Figure 3: Synthetic task systems characteristics. (a) shows that parallel execution time increases as the number of edges increases. There are 40 task systems per probability. (b) shows sequential execution time of task systems

assigned, better schedule can be found. There are two instances when no task pre-assigned has the best solutions (0.01, and 0.05, probability an edge exists between two nodes). As shown in Figure 4, the schedule length found (out of 40 cases) gets worse with the increased percentage of tasks pre-assigned, which is because a larger percentage of tasks pre-assigned decreases the solution space. Comparing the results to Figure 3, it can be seen that the best schedule length found with resources constraints is better than the best sequential schedule length without resources constraints. This shows the flexible and capability of genetic algorithm in finding good solutions within the confined resources constraints. It is noted that the simulation runs achieved reconfigurable resources utilization of over 93 percent.

It is important to note that when the pre-assigned percentage values are in the 40 percent to 50 percent range, there were up to three simulations (out of 40) with no feasible schedule at the end of 1000 iterations. Figure 5 shows the average schedule length of each probability value from 0 to 50 percent tasks pre-assigned, in 5 percent increments. (The average is calculated from 40 schedule lengths when available; in a few cases, only 37 solutions were found). The plots show that the average



(a) Probability values from 0.01 to 0.05



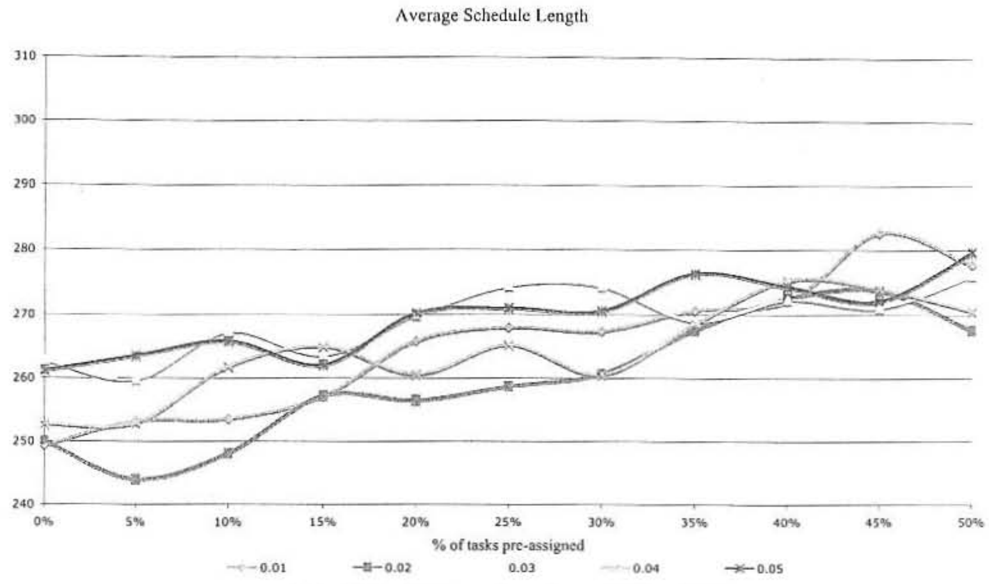
(b) Probability values from 0.06 to 0.10

Figure 4: The best schedule lengths found are presented in two plots for ease of reading. Each curve represents the probability of an edge existing between two nodes. There are 40 task systems per probability value. The pre-assigned percentage increases from 0% to 50% with 5% increment. Each dot on the plots indicates the best schedule length among the 40 simulations

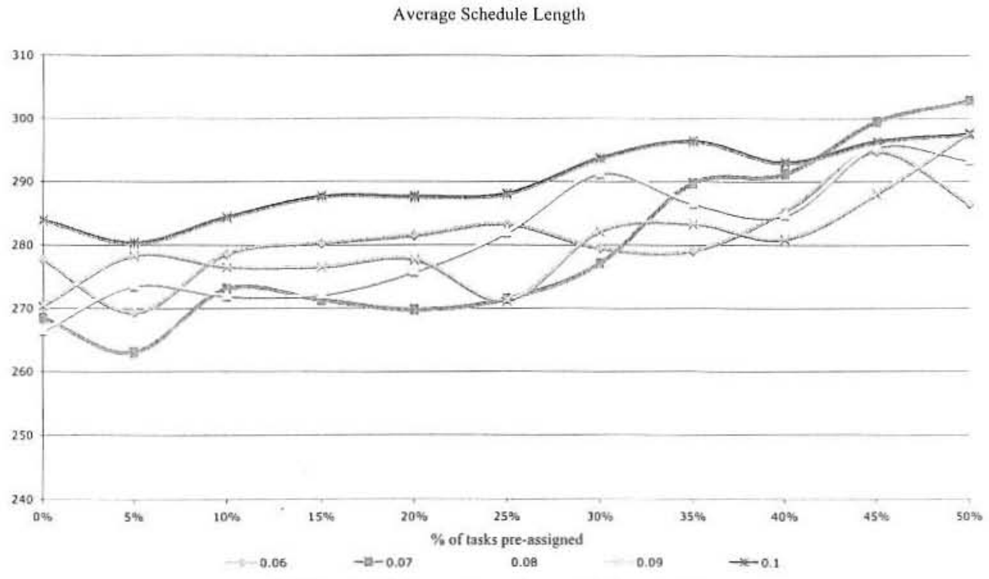
schedule length increases with the increasing percentage of tasks locked at desired functional units.

6 Conclusions

This paper shows that pre-assigning a number of tasks can help to determine a better schedule. It was shown that when 0 to 25 percent of tasks are pre-assigned, better schedules could be found and take advantage of the limited reconfigurable resource at the same time. Better schedule length can be easily found because the search space has been reduced by the pre-search task assignment phase. However, we also showed that when a large percentage of tasks is pre-assigned (locked to an execution unit), the execution schedule found is not as good as when only a minor percentage of the tasks is pre-assigned. We show that if the user decided to assign the tasks to the desirable functional units, the tool can take such assignments into consideration and determine a feasible schedule that can be implemented within finite resource reconfigurable hardware.



(a) Probability values from 0.01 to 0.05



(b) Probability values from 0.06 to 0.10

Figure 5: Average schedule length of each probability values when the percentage of pre-assigned tasks increases. As the number of pre-assigned tasks increases, the average schedule length increases

References

[1] Altera Nios Processor, http://www.altera.com/nios, 2006.

[2] S. Banerjee, E. Bozorgzadeh, and N. Dutt, "Physically-Aware HW-SW Partitioning for Reconfigurable Architectures with Partial Dynamic Reconfiguration," 42nd Design Automation Conference, Anaheim, CA, USA, June 13-17, 2005.

[3] Pascal Benoit, Lionel Torres, Gilles Sassatelli, Michel Robert, and Gaston Cambon, "Automatic Task Scheduling/Loop Unrolling using Dedicated RTR Controllers in Coarse Grain Reconfigurable Architectures," 19th IEEE International Parallel and Distributed Processing Symposium, Denver, CO, April 4-8, 2005.

[4] Yuan-Hsiu Chen and Pao-Ann Hsiung, "Hardware Task Scheduling and Placement in Operating Systems for Dynamically Reconfigurable SoC," *Proceedings of International Conference of Embedded and Ubiquitous Computing*, Nagasaki, Japan, pp. 489-98, Dec. 6-9, 2005.

[5] K. Danne and M. Platzner, "Partitioned Scheduling of

Periodic Real-Time Tasks onto Reconfigurable Hardware," *Proceedings of 20th IEEE International Parallel and Distributed Processing Symposium*, Rhodes Island, Greece, pp. 25-29 April 2006.

[6] M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness," W. H. Freeman San & Co. Ltd, Francisco, 1979.

[7] J. Holland, *Adaptation in Natural and Artificial Systems*, Ph.D. Dissertation, Ann Arbor, MI, University of Michigan, 1975.

[8] Kai Hwang, *Advanced Computer Architecture Parallelism Scalability Programmability*, McGraw-Hill Inc., New York, 1993.

[9] T. T.-O. Kwok and Yu-Kwong Kwok, "Practical Design of a Computation and Energy Efficient Hardware Task Scheduler in Embedded Reconfigurable Computing Systems," *Proceedings of 20th International Parallel and Distributed Processing Symposium*, Rhodes Island, Greece, April 25-29, 2006.

[10] J. Levman, G. Khan, J. Alirezaie, and K. Raahemifar, "Hardware-Software Co-Synthesis of Partially Reconfigurable Embedded Systems Optimized for Reduced Power Consumption," CCECE 2003 - Canadian Conference on Electrical and Computer Engineering - Toward a Caring and Humane Technology, Montreal, Que., Canada, May 4-7, 2003.

[11] Shang Li, Robert P. Dick and Kiraj K. Jha, "SLOPES: Hardware-Software Cosynthesis of Low-Power Real-Time Distributed Embedded Systems with Dynamically Reconfigurable FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(3):508-526, March 2007.

[12] S. M. Loo, B. Earl Wells, and J. Winningham, "A Genetic Algorithm Approach to Static Task Scheduling in a Reconfigurable Hardware Environment," *Proceedings of the ISCA 18th International Conf. of Computers and Their Applications*, Honolulu, HI, USA, pp. 36-39, 2003.

[13] S. M. Loo and Earl Wells, "Applying Stochastic Static Task Scheduling to a Reconfigurable Hardware Environment," *International Journal of Computers and Their Applications*, 12(2):57-75, June 2005.

[14] S. M. Loo and J. Winningham, "A Task Graph Set for Evaluation of Reconfigurable Hardware Scheduling Algorithms," *Proceedings of the ISCA 19th International Conference of Computers and Their Applications*, Seattle, Washington, USA, pp. 159-163, 2004.

[15] M. Purnaprajna, Marek Reformat, and Witold Pedrycz, "Genetic Algorithms in Hardware-Software Partitioning," *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*, Las Vegas, NV, USA, pp. 123-129, Jun 21-24, 2004.

[16] Ji Qi, Xi Li, Nan Hu, Xue-Hai Zhou, Yu-Chang Gong, and Feng Wang, "Algorithms of Resource Management for Reconfigurable Systems Based on Hardware Task Vertexes," Chinese Institute, of Electronics *Electronica Sinica*, 34(11):2094-2098, November, 2006.

[17] J. Resano, "A Specific Scheduling Flow for Dynamically Reconfigurable Hardware," 14th International Conference of Field-Programmable Logic and Applications, Antwerp, Belgium, August 30-September 1, 2004.

[18] Rockwell International Corporation – Rocketdyne Division, Space Transportation System Technical Manual – Space Shuttle Main Engine, E41000 RSS-8559-1-1-1, NASA Technical Report RS007001, September 1983.

[19] C. Steiger, H. Walder, and M. Platzner, "Operating Systems for Reconfigurable Embedded Platforms: Online Scheduling of Real-Time Tasks," *IEEE Transactions on Computers*, 53(11):1393-1407, Nov. 2004.

[20] T. Ushio and K. Onogi, "Scheduling of Periodic Tasks on a Dynamically Reconfigurable Device using Timed Discrete Event Systems," Eighth International Workshop on Discrete Event Systems, Ann Arbor, MI, USA, 10-12 July 2006.

[21] B. Earl Wells, Kenneth G.Ricks, and John M. Weir, "Parallel Simulation of a Large Scale Aerospace System in a Multicomputer Environment," *IEEE Transactions on Aerospace and Electronic System*, 33(2):507-523, April 1997.

[22] Xilinx MicroBlaze Processor, http://www.xilinx.com/microblaze, 2006.

Sin Ming Loo received his Ph.D. in Computer Engineering from University of Alabama at Birmingham and the University of Alabama in Huntsville in 2003. From 1998 to 2003, he was involved in research projects which included development of space plasma simulation model utilizing parallel processing on 256-processor HP Exemplar X2000 and 128-processor SGI Origin, built and maintained 80-node Pentium-4 Beowulf cluster, and development of digital system rapid prototyping course materials. Dr. Loo is presently an Associate Professor of Electrical and Computer Engineering at the Boise State University. His research interests include scheduling, parallel processing, distributed sensor networks, embedded system, hardware/software codesign, and reconfigurable computing.

JingXia Wang received her M.S. in Electrical Engineering from the University of WuHan Surveying and Mapping Technology in 1994. From 1994 to 2003, she was employed as an Assistant Professor and currently is an Associate Professor in the Department of Electrical Engineering in Shenzhen Polytechnic. From 2005 to 2006, she worked in the Department of Electrical and Computer Engineering at Boise State University in USA as a visiting scholar. Her research interests include embedded system, reconfigurable computing, computer architecture, hardware/software codesign.