Selection of *RTOS for* an Efficient Design of Embedded Systems

S. Ramanarayana Reddy

† Department of Computer Science, Indira Gandhi Institute of technology, Indraprasta University, Delhi 110006, India

Summary

Real Time Operating System (RTOS) is a basic building block of most of the Embedded Systems (ES). There are wide ranges of RTOS's available to the designers/ developers of ES's ranging from RTOS for robotics to home appliances. Each application demands a specialized set of requirements and to meet these requirements the designer needs to select the RTOS, which meets the desired requirements. It is a critical task for him/her and time consuming because it involves to know all the specifications of different RTOS and there are various RTOS's available in the market that include micro kernels to commercial RTOS's . So it is the task of the designer to select the suitable RTOS from the vast list of RTOS's. The design space available to any RTOS is very large and there are countless set of characteristics such as Development Methodology, Scheduling Algorithms, Kind of Real Time (Soft of Hard), Priority Levels, Development Host, Standards followed, Kernel ROM size, Kernel RAM size, Multi process Support, Multiprocessor Support, Interrupt latency, Task Switching time, Kind of IPC mechanism, Memory management, Power management, Task management, Price etc. These characteristics will guide the designer for selecting the RTOS that meets the requirements. Selecting the RTOS based on these parameters is a multidimensional search problem with each dimension corresponds to a RTOS characteristic and it requires an exhaustive search with tremendous computing resources and time. In our framework of RTOS selection, we have used the Simple Genetic Algorithm (GA) with interactive GUI by which the developer can choose the right RTOS for a given application or a project efficiently.

Key Words
RTOS, GA, Real Time Systems, Embedded System Design

1. Introduction

Embedded systems are an invention that has taken more than a hundred years to take the present day shape. The way they have manifested themselves in our lives, is nothing less tan the effect that the discovery of fire or the invention of the wheel had on the evolution of mankind. An embedded computing system or embedded system includes a digital electronic system embedded in a larger system and it is an application specific. These systems are becoming an integral part of various commercial products like mobile phones, watches, flight controllers etc. The developer needs to select a right RTOS based on these applications. There is a strong and compatible relationship between the system hardware and the software, primarily the operating system to ensure hard real time deadlines.

The organization of the paper is as follows: in section 2 we survey the related work in selecting the RTOS and its important parameters. We explain the important parameters of a RTOS and its role in embedded systems design, in section 3. Section 4 describes the fundamentals of GA and it's operators. Section 5 will describe the selection of RTOS and in section 6, the example. In section 7, experimental results and discuses and finally, in section 8 we have provided the conclusions and directions for future work.

Related Prior Work

Decision making occurs in all fields of human activities, such as scientific, technological and every sphere of our life. Engineering design, which entails sizing, dimensioning and detailed element planning is also not exempt from its influence. The past decade has seen a significant research work on selecting the RTOS [17, 18, and 19]. Designers are impressive task when selecting the RTOS for specific applications like Space, Security, military, process industry, communications, robotics, Data Acquisition, consumer electronics and so on in which each application demands specific requirements.

Just like high-level languages, RTOS's allow you to develop applications faster [19]. They can require a little more overhead, but as the technology improves, the overhead seems to diminish. In Greg Hawley [19], he has provided criteria for selection of RTOS based on the processor and based on the requirements. He also considered many other parameters like, company profile, licensing policy, technical support etc.

In Philip Melanson, Siamak Tafazoli [17], a selection methodology for the RTOS market various method are adopted for space applications. This paper describes the elimination criteria for selection of RTOS to a very specific space application and ranked the existing commercial RTOS that are available in the market but they have not provided the generic framework for RTOS selection. In Ger Scoeber, how to select your RTOS [18] described the framework for selection of RTOS for a class of applications and its characteristics that meets the application but it doesn't provided the methodology to select the RTOS based on the designers/developers requirements which are incorporated in this paper. Criteria for selection of a RTOS need to be much more flexible and much less specific [20].

Since 1940, several optimization problems have not been tackled by classical procedures including: Linear Programming, Transportation, Assignment, Nonlinear Programming, Dynamic Programming, Inventory, Queuing, Replacement, Scheduling [3, 9] etc.

Normally, any engineering problem will have a large number of solutions out of the feasible solutions. The designer's task is to get the best solution out of the feasible solutions. The complete set of feasible solutions constitutes feasible design space and the progress towards the optimal design space involves some kind of search within the space. The search is of two kinds, namely deterministic and stochastic.

Non traditional search and optimization methods have become popular in engineering optimization in the recent past, and these algorithms include: *Simulated Annealing, Ant Colony Optimization, Random Cost, Evolution Strategy, Genetic Algorithms, Cellular automata* [3, 9, 12] *etc.* Obenland's [16] paper looks at POSIX in real time systems and POSIX thread extensions and compares the performance of the general purpose operating systems and two real time operating systems. Stewart's [15] paper illustrates different methods for estimating execution time of both user level and operating system overhead. Coarse gain timing measurements is calculated in software in real time granularity in milliseconds. Mana discusses Linux as a real time operating system and different approaches for real time Linux kernel. Timmerman [14] describes the framework for evaluation of real time operating systems. This article makes a really good point of comparing RTOS under different load conditions.

Yodaiken's [2] paper explains hard real time approach of RTLinux and it's one of the first papers written on RTLinux.

2. Need of RTOS for an Embedded System

Embedded systems are continuously increasing their hardware and software complexity moving to single-chip solutions [1] (SoC's). The RTOS in Embedded System mainly does the following tasks.

- It simplifies control code required to coordinate processes.
- It provides an abstraction interface between applications with hard real-time requirements and the target system architecture.
- Availability of RTOS models is becoming strategic inside hardware/software co-design environments.

3.1 RTOS

RTOS can be defined as "The ability of the operating system to provide a required level of service in a bounded response time." (POSIX Standard 1003.1). A real-time system responds in a (timely) predictable way to unpredictable external stimuli arrivals. To build a predictable system, all its components (hardware & software) should enable this requirement to be fulfilled. Traffic on a bus for example should take place in a way allowing all events to be managed within the prescribed time limit. RTOS should have all the features necessary to be a good building block for a Real Time system. However it should not be forgotten that a good RTOS is only a building block. Using it in a wrongly designed system may lead to a malfunctioning of the RT system. *A good RTOS can be defined as one that has a bounded (predictable) behavior under all system load scenarios (simultaneous interrupts and thread execution).* In RT system, each individual deadline should be met. There are various types of real-time systems

3.2 Types of RTOS

RTOS's are broadly classified in to three types, namely, the Hard Real Time RTOS, Firm Real Time RTOS and Soft Real Time RTOS, which are describes bellow.

Hard real-time: missing a deadline has catastrophic results for the system;

Firm real-time: missing a deadline entails an unacceptable quality reduction as a consequence;

Soft real-time: deadlines may be missed and can be recovered from. The reduction in system quality is acceptable.

3.3 The OS and RTOS

A real time operating system is an operating system that allows one to specify constraints on the rate of processes, and that guarantees that these rate constraints will be met, whereas an operating system is a low–level program that runs on a processor responsible for scheduling processes, allocating storage and interfacing to peripherals among many other things.
Basically an operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs. Real time operating systems are systems, which respond to any external unpredictable event in a predictable way and with strict timing constraints. Real time operating systems have become a very common phenomenon in real time applications in the present day scenario. Developers are given the task of making software with real time constraints. A large number of RTOS are available in the market making it difficult for the designers to decide which one to use, such that it provides the best overall benefits in terms of requirements a particular application. Selecting an appropriate RTOS which meets all the designer requirements is a very critical task. There are set of certain benchmarks, which could be used to examine an RTOS such as development hosts, priority levels, thread switch latency, response time etc. The important qualities that make the good RTOS are Multi-threaded and pre-emptible, Thread priority has to exist because no deadline driven OS exists, and support predictable thread synchronization mechanisms, and a system of priority inheritance must exist.

4. Algorithms for RTOS Selection

There is hardly any specific algorithm found for RTOS selection except the elimination criteria, which is difficult for the developer and time consuming as it mentioned in the related work. It is the first attempt to use a tool to select RTOS which uses the Genetic Algorithm. Over the last couple of decade, GA's have been extensively used for optimization and search tools in various domains, which includes all branches of engineering and Science. The basic reasons for the success of GA's are their broad applications, Parallelism, easy of use and global perspective[2]

In principle, GA's are adaptive procedures that find the solutions to problems by evolutionary process based on natural selection. In practice, GA's are iterative search algorithms with various applications. In general, GA's maintain a population of individual solutions to the problem. Each individual can be represented by a string called chromosome. During each iteration, or called generation, the individuals in the current population are rated for their fitness as a solution. The fitness function evaluates the "survival" or "goodness" of each chromosome. By applying the different genetic operators, new populations of candidate solutions are generated.

4.1 GA Operators

In general, GA's make use of different operators. In this implementation, we use the *selection, crossover, and mutation* operators which are described below.

4.1.1 Selection or Reproduction

Individuals in the population can be heuristically or randomly initialized. The population of the next generation is reproduces using a probabilistic selection process. Individuals with higher fitness will have the more chance to reproduce.

4.1.2 Crossover

This operator takes two randomly chosen parent individuals as input and combines to generate two children. This is performed by choosing two crossing points in the strings of the parents and then exchanging the allelic values between these two points as shown in the Figure 1. The crossover operator provides a powerful exploration capacity by exchanging the information from two parents.

Fig. 1. Crossover operation

A: 00 **111** 000
B: 11 **000** 111
Before crossover

A: 00 **000** 000
B: 11 **111** 111
After Crossover

4.1.3 Mutation

The crossover operator may lead to falling into a local minimum of the fitness function because a generated child tends to be very similar to its parents. In order to reduce this phenomenon, mutation operator is used. This operator creates new individual by modifying gene values of an existing individual as shown in the Figure 2**.**

Before Mutation: 110 **0** 011
After Mutation: 110 **1** 011

Fig. 2. Mutation Operation

Mutation provides the random search in the problem space and prevents complete loss of genetic features through selection and elimination. Thus mutation operator reduces the probability of falling into a local minimum of the fitness function.

After applying reproduction, crossover and mutation, the new population is ready for testing for fitness. Now, we apply GA for decoding new strings, calculate fitness, and then generate a new population.

5. Selection of RTOS

Ranking RTOS is a tricky and difficult because there are so many good choices are available in the market [21]. The developer can choose either commercial RTOS (44% developers are using) or open- source RTOS (20) or internally developed RTOS (17 %). This shows that almost 70% of developers are using the RTOS for their current projects [20] and are migrating from one RTOS to another due to various reasons. To handle the current requirements of the customers, developers are using 32 bit controllers in their projects in which 92% projects/ products are using RTOS[21] and 50% of developers are migrating to another RTOS for there next project. This influences importance of the selection of right RTOS to a particular project so that it meets all the requirements and fulfills its intended task.

In all of the related work authors have used the elimination criteria which are manual and it takes more time and need the detailed specifications of all the existing commercial RTOS's**.**

In order to select RTOS, the designer first identify the parameters for selection based on the application and the intended requirements are provided to the systems through an interactive user friendly GUI which is shown in below. The designer has the freedom to omit and or include parameters and also he/she can edit the database of RTOS for efficient selection under multi user environment. Subsequently, genetic algorithm is used to arrive at the RTOS taking into account the parameters that are specified.

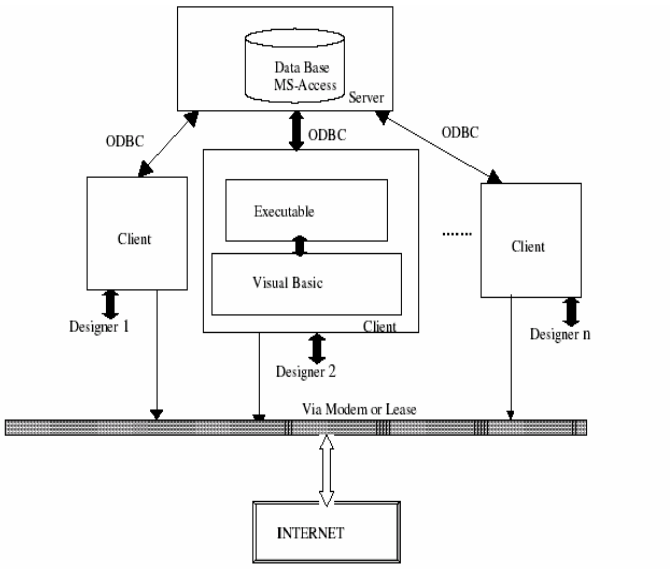

Fig. 3. Architecture of the System

5.1 RTOS Parameters

Among the different parameters for selecting the RTOS, the ones used in our system are: 1. Interrupt Latency, 2. Context switching 3. Inter task Communication (Message Queue Mechanism, Signal Mechanism, Semaphores), 4.Power Management (Sleep mode, Low power mode, idle mode, Stand by mode) 5. No. of Interrupt levels 6. Kernel Size 7.Scheduling Algorithms (Round Robin Scheduling, First Come First Serve, Shortest Job First, Preemptive Scheduling etc), 8.Interrupt Levels, 9 Maintenance Fee 10.Timers 11. Priority Levels 12.Kernel Synchronization (timers, mutexes, events, semaphores etc), 13. Cost, 14. Development host, 15. Task switching time and 16 Royalty Fee. There are more parameters like target processor support, Languages supported, Technical Support etc are also important which are considered by the developer. We have used the SGA for selecting RTOS, which is described in the following section. Our system will output a set of RTOS's from which one will be selected by considering the processor support, languages supported and Technical Support etc which are also important.

5.2 Genetic Algorithm

The genetic algorithm used in our system is given below.

Simple Genetic Algorithm (SGA)

1. *Randomly initialize population (t)*
2. *Determine fitness of population(t)*
3. *Repeat*
4. *select parents from population(t)*
   a. *Perform crossover on parents creating population(t+1)*
   b. *Perform mutation of population(t+1)*
   c. *Determine fitness of population(t+1)*
5. *Until best individual is good enough*

5.2.1 The Population

The population is created statically and stored in the system. From these, an initial population is created randomly by using a random function.

5.2.2 The Fitness Function

The fitness function is the weighted sum of the parameters given in section 3, each of which contribute the "goodness" of the final selection of RTOS

Fitness of a chromosome is evaluated by using the fitness function (FF) which is given by

$$FF = \sum_{i=0}^{16} (W_i F_i) \quad \text{Where}$$

W_i is the weight of i^{th} parameter and F_i is the fitness value of i^{th} parameter

Let us first consider the weights. Each application of an embedded system will have specialized requirements. The requirements can be characterized using the parameters specified in section 5.1 by assigning appropriate weights. The weights change depending on the application. For example, for children toys, cost may be the main criteria and hence will have maximum weight while for robotic applications response time would be the parameter with maximum weight. To meet these specifications, the user has to specify the weights for each parameter so that an appropriate RTOS will be selected. In the fitness function, W_i is the weights assigned by the user.

Consider, now, the fitness values. The parameters of RTOS given above have different values for different RTOS. For example, the interrupt latency can be 5ns for one RTOS and 15ns for another. The different values are mapped to a scale and the value on the scale is the fitness value. For example, if the scale for interrupt latency is 5 to 15 then, for the RTOS with 5ns as interrupt latency, the fitness value is 1 as it is better to have low interrupt latency. Since the values of these parameters are available beforehand for the RTOS that are available in the market, the fitness values are precompiled at the time of generating the database of RTOS. However, the designer can alter the values if needed.

Now, by using the fitness function FF defined earlier, we evaluate the overall fitness value of the Chromosomes.

5.2.3 The Operations

Cross Over

In our algorithm, two-point cross over is used, which means that the cross over operation as described in section 4.1.2 is performed at two places, which are selected randomly. It helps to avoid the generation of chromosomes, which are replica of their parents. The cross over itself is performed using fifteen bits of the selected chromosomes for cross over.

Mutation

In our algorithm, mutation is performed on five bits of a chromosome, which are selected randomly by using random function. We have chosen five bits to overcome the problem of local minimum.

5.2.4 The New Population

The population is generated by using the Roulette Wheel Selection, which is shown in the Figure 4. Based on the chromosome fitness function value, the survival of the chromosome is selected In our system, if the chromosome fitness function value is less then 19 %, the chromosome will not be survived for next generation. And if the chromosome fitness function value is in between 19 to 35 %, the only one copy is considered for next generation, else two copies are considered for the nest generation.

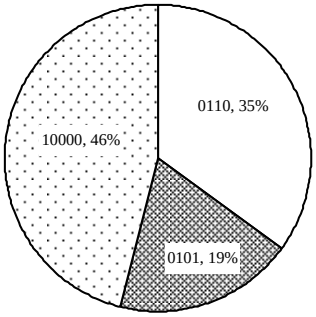

Fig.4. Roulette Wheel Selection

5.2.5 Accuracy Percentage

If a RTOS that is chosen matches all the specified parameters then the accuracy percentage is said to be 100%. In terms of fitness function, the accuracy percentage is defined as follows:
First the chromosome corresponding to the parameters specified by the user is created. The fitness function value for this chromosome is computed. Let it be x. Let y be the fitness function value of a generated chromosome. The accuracy percentage of this chromosome is

Accuracy percentage = y /x * 100

In our system the user can specify the accuracy percentage. Thus, if none of the RTOS which are available in the system are matching exactly, it is still possible to choose an RTOS which is close to the required one. Accuracy percentage acts also as stopping criteria for the SGA.

6. Example

We have developed a graphical user interface so that the user can specify the weights for the parameters of the RTOS for his application. The parameters specified by the user using the GUI are given below.

Development Methodology – Cross	Weight – 1
RTOS Supplied as – Object	Weight – 2
Development Host – UNIX	Weight – 3
Standard – POSIX .1	Weight – 4
Kernel ROM – 280K/4M	Weight – 5
Kernel RAM – 500K/4G	Weight – 6
Priority Levels – 512	Weight – 7
Multi process Support – No	Weight – 8
Multiprocessor Support – No	Weight – 9
MMU Support – No	Weight – 10
Royalty free – No	Weight– 11
Standard phone support – Paid	Weight – 12
Preferred phone support – Paid	Weight – 13
Base price – 7495$	Weight – 14
Maintenance fee – 15% of list price	Weight– 15
Task switching time – 4us to 19us	Weight – 16

In addition the user is asked to specify the percentage of accuracy. Let it be 80%.

For the above specification, the decoded binary Chromosome is
0.0.0000.0000.0000.0000.000.0.0.0.0.0.0.001.000.000
and for which the Fitness Function value is calculated as 34. For each chromosome is represented with the 36 bit length binary string. Each decimal point separates the one characteristic of the RTOS which represents the values of it. Hence we require 36 bits to represent the entire chromosome.
The first population is generated randomly along with its fitness function values corresponding to each chromosome. This process is repeated until its desired accuracy achieved. In this example the *RTOS that specifies the given description is: VxWorks.*

7. Results

We have implemented the SGA with Visual Basic, and experiments were conduced on Intel P4, 1.8 GHz with 128 MB of RAM. In this section we compare the results of the above example with different population sizes taking a constant crossover and mutation rate with 50% accuracy. Here, Ch1 to Ch14 represents the Chromosome Number (Ch Num), F Val G1 to F Val G3 are Fitness value of Generation 1, 2 and 3 respectively.

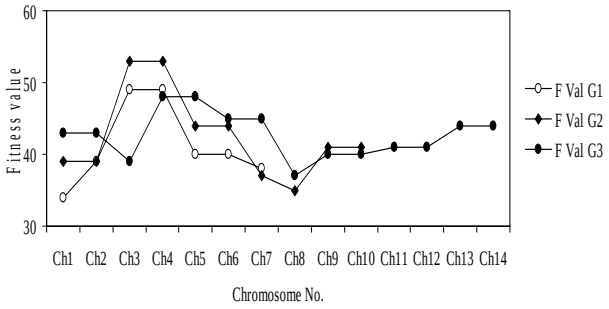

Fig. 5. Fitness values of chromosomes with different generations.

Figure 5 shows the chromosomes fitness values with various generations and it is found that the fitness values of the chromosomes are more stable in generation three and it takes more CPU time. Figure 6 depicts the CPU time/Population size and it shows that more the population size and more the CPU time.

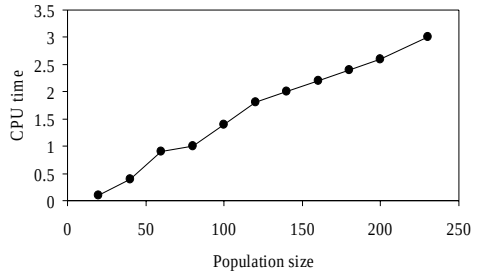

Fig. 6. Population Vs CPU Time

Again we compare the results of the test cases with constant population sizes taking a variable crossover rate and constant mutation rate of 5 bits per chromosome with 50% accuracy is shown in the figure7.

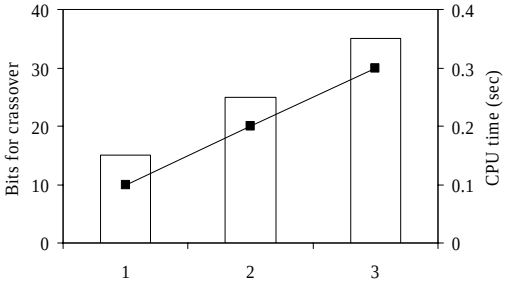

Fig. 7. No. of Bits for crossover Vs CPU Time

Again we compare the results of the test cases with constant population sizes taking a constant crossover rate of 15 bits per chromosome and a variable mutation rate with 50% accuracy which is shown in table 1.

Test No	NO of Bits for Mutation	CPU Time (sec)
1	5	0
2	10	0.05
3	15	0.08

Table 1 .Mutation Vs CPU Time

Again we compare the results of the test cases with constant population sizes taking a constant crossover rate of 15 bits per chromosome and a constant mutation rate of 5 bits per chromosome with variable accuracy percentage and the results are depicted in the figure 8.

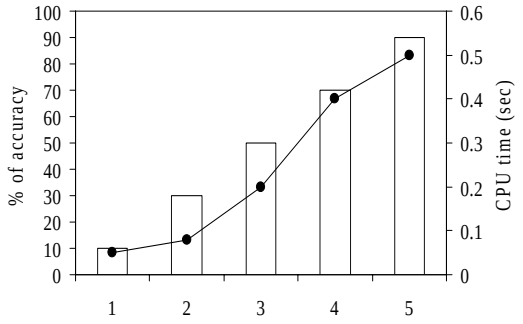

Fig. 8. Percentage of accuracy Vs CPU time

7.1 Discussions

As shown by the output of various test cases, the graphs depict the result of various parameters which affect the output of the system in terms of CPU time and find that even though the output varies considerably from sample to sample, there is a tremendous reduction in performance due to increased levels of percentage accuracy. As the number of bits used for crossover and mutation operations increases, the efficiency of the system reduces and consumed more time. However, the difference is very small considerably and can be ignored.

However the effect on performance cannot be ignored due to large population sizes. As more and more generations are developed, they become better than the previous generations, and thus a large population of chromosomes is developed. Due to the large population size, finding an optimal RTOS takes more time than usually required. This is because a large number of chromosomes have to be crossed over, mutated and compared for the final results.

One of the most challenging aspects was to represent the chromosomes in terms of binary strings. We have used automatic allocation of each parameter to variable length binary digits. We have used the concept of separators to distinguish between the binary conversions for various parameters.

We have also used a weight system to calculate the fitness number. The user assigns weight according the degree of effectiveness of each parameter. A higher value will result in a higher fitness number if that parameter has that specific value for an RTOS.

Operations like crossover, mutation, fitness number etc are used often in the processing of chromosomes. These have been coded in the form of functions that are global to all modules.

Studying the output for the most optimal RTOS based on user specifications, we came to the conclusion that the most optimal RTOS was strictly dependent on the test metrics parameters like scheduling priorities, timing constraints, RAM and Rom size, Development methodology, Development host etc.

It was seen that when the percentage accuracy was 50%, the results were obtained most easily. As the accuracy of percentage increased the RTOS's matching the specified criterion were fewer. However a higher percentage of accuracy means a more optimal solution.

Our analysis gives the developer/ designer a portal to decide on a real time operating system which must suits his choice of parameters and is the most optimal one available for that purpose, with in a short time.

This method is efficient then the elimination techniques because

- It never considers all the specifications of the RTOS but it only consider the specified ones of the developer.
- It doesn't require much time.
- This system has a provision to provide percentage accuracy (It helps to allow the developers that how much % of guarantee that the selected RTOS is) and weights for all the specified parameters so that it selects the optimal one that exits in the data base.

8. Conclusion

Real time applications have become a popular these days due to the complexity in the system. To meet those complexities, the developers are given the invariable task of making the real time software. There are quite large umber of RTOS are available in the market and one dose get confused as to which one such that it provides the efficient embedded systems design in terms of cost, power consumption, reliability, speed etc.

In this paper, we described a Simple Genetic Algorithm that is designed to find the suitable RTOS for a specific application. The methodology described for RTOS selection is unique and efficient for large number of RTOS's. It has user-friendly graphical interface (GUI) though which the designer can alter the specifications and specify the new requirements for RTOS selection for a given application. It generates the optimal RTOS based on the requirements that are entered by the user keeping in mind the amount of accuracy required. This is done with the help of genetic algorithms. Our analysis and the developed system gives the user a portal to decide a real time operating system which most suits his choice of parameters and is the most optimal one available for that purpose. The designer has an option of choosing from pre-defined input or can specify his/ her own input.

8.1 Advantages of the system

The main advantages of the system are:

The user gets an appropriate RTOS just by giving the specifications and the desired accuracy and the whole search based on those specifications is carried out by the system and hence the result is provided through an easy designed interactive GUI.

The user has the option of specifying the accuracy percentage to carry out his search which could vary depending on the level of strictness required, which is an efficient method compared to other methods which uses the elimination criteria

The user has the provision of selecting more than one option in each parameter thus making his search more advanced in terms of parameters.

Choosing the most appropriate RTOS can still result in significant cost savings, improved level of technical support and high levels of product integration.

8.2 Limitations of the system

Though there are advantages of the system there are some limitations which can be eliminated in future work.

Only sixteen parameters have been taken into consideration for carrying out the search and the user cannot increase or add more parameters to this list and can be dynamic.

The search is restricted as it runs only for those RTOS's already provided in the database as entered by the administrator/user and not for all.

The number bits involved in the crossover and mutation operations are fixed. Fifteen bits are taken for crossover and five for mutation and can be made variable.

It is necessary that the user be aware of the priority/weight of each sub option in each parameter in order to obtain desired results.

Acknowledgements
The first author (SRNR) would like to express his thanks to the Principal (IGIT) and the Dean (SC&SS, JNU) for providing necessary facilities to complete this research work.

9. References

[1] RTOS for FPGA-A White paper by Colin Walls, Accelerated Technology, Embedded Systems Division, Mentor Graphics Corporation (www.mentor.com/fpga).
[2] Victor Yodaiken and Michael Barabanov "design Document about RTLinux in FSMLabs 1997.
[http://citeseer.ist.psu.edu/408922.html]

[3] Koza, John R., *Genetic* Programming: *On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: The MIT Press, 1992.

[4] Cramer, Nichael Lynn: "A Representation for the Adaptive Generation of Simple Sequential Programs" Proceedings, International Conference on Genetic Algo, July 1985 [CMU], pp183-187.

[5] Wayne Wolf, Computers as Components: Principles of Embedded Computing System Design, Morgan Kaufmann Publishers, 2001.
[6] Frank Vahid and Tony Givargis, Embedded System Design: A Unified hardware/ Software introduction, John Wiley& Sons, 2002.
[7] Scott Rosenthal, Selecting an embedded Processor involves both simple and non-technical criteria, June, 1997.
[8] Sharad Agarwal, Edward Chan, Ben Liblit, Processor Characteristic Selection for Embedded Applications via Genetic Algorithms, December,1998.
[9] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.
[10] Khawar M. Zuberi, Kang G. Shin, (2001). " EMERALALDS: A Small- memory Real Time Microkernel", IEEE Trans on Software Eng.; Vol.27, No.10, Oct.pp. 909 - 929.
[11] Shanil Mechant, Kalen Dedhia,"Performance computation of RTOS, thesis, Dept of electrical Eng., Columbia University.
[12] Mehrdal Dianati, Insop Song, Mark Treiber, "An Introduction to Genetic Algorithms and Evalution Stragies" Univ. of Waterloo, Canada.
[13] Quagliarella. D, Periaux. J, Poloni. C, Winter. G, "Genetic Algorithms and Evolution Strategy in Engineering and Computer Science".
[14] Martin Timmerman, "RTOS Evaluations", Real Time Magazine 98(3) March 1998.
[15] David Stewart, "Measuring Execution time and Real Time Performance", Embedded Systems conference, San Francisco, April 2001.
[16] K. Obenland, " Real Time Performance of Standards based Commercial Operating Systems, Embedded Systems conference, San Francisco, April 2001.
[17] Philip Melanson, Siamak Tafazoli, "A selection methodology for the RTOS market", DASIA 2003 conference, Prague Czec Republic, June 2003,.
[18] Ger Scoeber, "How to select your RTOS" Bits and Cips Micro-event: Embedded operating Systems, Jan 29th 2004.
[19] *Greg Hawley*, "Selecting a Real-Time Operating System" Embedded Systems Programming Magazine. [www.embedded.com]
[20] Ljerka Beus-Dukic, "Criteria for Selection of a COTS Real-Time Operating System: a Survey"[ljerka.beus@unn.ac.uk]
[21] Jim Turly, "Embedded Systems survey: Operating systems up for grabs", Embedded Systems Design, May 24 2005.

S. Ramanarayana Reddy received M. Tech degree, from Jawaharlal Nehru University, New Delhi in 2002. He is working as a Lecturer (from 2002) in the Dept. of Computer Science, IGIT, IP Univer., Delhi. His research interest includes embedded systems, system programming, and real-time systems. He is a member of CSI, VLSI Society of India and AMIE.