Chapter 5

RTOS SELECTION

5.1 Introduction

Embedded systems are inventions that have taken more than a hundred years to evolve into the present shape. The way they have manifested themselves in our lives, is nothing less compared to the effect that either 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 application specific. These systems are fast becoming the integral parts of various commercial products like: mobile phones, watches, flight controllers etc. The developer needs to select a suitable Real Time Operating System (RTOS) based on these applications. There is a strong and compatible relationship between the system hardware and the software, primarily the RTOS that ensure hard real time deadlines.

5.2 Related Work

Decision-making occurs in all fields of human activities such as: scientific, technological, and nearly every aspect 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 both qualitative and quantitative significant research work,
particularly on selecting the RTOS [24, 26 and 42]. Selecting the RTOS for specific applications such as: space, security, process industry, communications, data acquisition, etc., demands highly specific requirements.

As in the case of high-level languages, RTOSs allow applications to develop faster. They require a little more overheads, but as the technology improves, the overheads seem to diminish. Greg Hawley [24], provided the criteria for selection of RTOS, based on the processor and on the type of requirements. He has also considered many other parameters such as: company profile, licensing policy, technical support etc.

Philip Melanson, Siamak Tafazoli [42], proposed a selection methodology for the RTOS. It describes the elimination criteria for the selection of RTOSs to a very specific space application, and ranks the existing commercial RTOSs, that are available in the market but which have not provided the generic framework for RTOS selection. The criteria for selection of RTOS need to be much more flexible and much less specific [43]. Ger Scoeber, in his paper, How to Select Your RTOS [26], proposed a framework for selection of RTOS for a class of applications and its characteristics that meet the application. However, it does not provide the methodology to select the RTOS based on the designers or developers requirements. This problem has been addressed in this chapter.

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 [29, 34].

Normally, any engineering problem will have a large number of solutions within the existing feasible solutions. The designer's task is to get the best possible solution out of the feasible solutions that are available. The complete set of feasible solutions constitutes
feasible design space, whereas the progress made towards the optimal design space involves some kind of search within the space. The search is of two kinds, viz., 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 [29, 34, 37].

Obenland’s [41] 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 [40] paper illustrates the different methods for estimating execution time of both user level and operating system overhead. Timmerman [39] describes the framework for evaluation of real time operating systems. In this system, A Genetic Algorithm based approach for selection of the RTOS is proposed.

5.3 RTOS — Real Time Operating System

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. An efficient
RTOS can be defined as, one that has a bounded (predictable) behavior under all system load scenarios (simultaneous interrupts and thread execution).

A real time operating system is an operating system that allows one to specify constraints on the rate of processes, and one which 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 with 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 RTOSs 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 of a particular application. Selecting an appropriate RTOS that meets all the designer requirements is a very critical task. There is a 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 a 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. In RT system, each individual deadline should be met.
Types of RTOS

RTOSs are broadly classified into three types, namely: the Hard Real Time RTOS, Firm Real Time RTOS, and Soft Real Time RTOS, which are described below:

**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; and

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

Embedded systems are continuously increasing their hardware and software complexity, moving to single-chip solutions (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.

5.4 RTOS Selection

Ranking RTOS is both a tricky and difficult task because there are so many competent choices that are available in the market [44]. The developer can choose from 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 [43] 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[44] and 50% of developers are migrating to another RTOS for their next project. This influences importance of the selection of right RTOS for a particular project so that it meets all the requirements, and fulfills its intended task.

Literature survey revealed most of the authors used the elimination criteria, which are manual and which takes more time, and also require the detailed specifications of all the existing commercial RTOSs.

In order to select the RTOS, the designer first identifies the parameters for selection based on the application and the intended requirements are provided to the systems through an interactive GUI shown in Figure 6.1. 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.

5.4.1 RTOS Parameters

Among the different parameters for selecting the RTOS, the ones used in our system are:
semaphores, etc), 13. Cost, 14. Development host, 15. Task Switching Time, and 16 Royalty Fee. There are also other parameters like target processor support, Languages supported, Technical Support etc., are also important which are considered by the developer. We have used the Simple Genetic Algorithm (SGA) for selecting RTOS, which is described in the following section. Our system will output a set of RTOSs, from which one will be selected by considering the processor support, languages supported, and Technical Support, which are also equally important.

5.4.2 Genetic Algorithm for RTOS Selection

There is hardly any specific algorithm found for the RTOS selection process except the elimination criteria, which is both difficult for the developer and time consuming, as 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 decades, GA’s have been extensively used for optimization and search tools in various domains, which include 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.

In principle, GAs are adaptive procedures that find solutions to problems by an evolutionary process based on natural selection. In practice, GAs are iterative search algorithms with various applications. In general, GAs 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.

In general, GAs make use of different operators. In implementation, we use the selection, crossover, and mutation operators which are described in the following:

Selection or Reproduction

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

Crossover

This operator takes two randomly chosen parent (chromosomes) individuals as inputs and combines them to generate two children. This is performed by choosing two crossing points in the strings of the parents, and then exchanging the values between these two points, as shown in the Figure 5.1. The crossover operator provides a powerful exploration capacity by exchanging the information from two parents.

Mutation

The crossover operator may intern 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 a new individual by modifying gene values of an existing individual as shown in Figure 5.2.

![Figure 5.1: Crossover operation](image)

![Figure 5.2: Mutation Operation](image)
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 its fitness. Now, we apply GA for decoding new strings, calculate fitness, and then generate a new population. The GA used in this system is as follows:

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 found good enough

**Population**

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

**Fitness Function**

The fitness function is the weighted sum of the parameters given in section 5.4.1, each of which contributes to the "goodness" of the final selection of RTOS. Fitness of a chromosome is evaluated by using the fitness function \((FF)\), which is given by

\[
FF = \sum_{i=0}^{16} (W_i P_i) - 5.1
\]

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 5.4.1 by assigning appropriate weights. The weights can be changed depending on the application. To meet the desired specifications, the user has to specify the weights for each parameter, so that an appropriate RTOS will be selected. In the fitness function, \( W_i \) refers to the weights assigned by the user.

Consider, now, the fitness values. The parameters of RTOS given above have different values for different RTOS. The different values are mapped to a scale and the value on the scale is the fitness value. 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. Using the fitness function \( FF \), defined earlier, we evaluate the overall fitness value of the Chromosomes can be evaluated.

In this implementation process, two-point cross over is used, which implies that the cross over operation as described in section 5.4 is performed at two places, which are selected randomly. It helps to avoid the generation of chromosomes, which are replicas of their parents. The cross over itself is performed using fifteen bits of the selected chromosomes for cross over. 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. Roulette Wheel Selection as shown in the Figure 5.3 is used to generate the population. Based on the chromosome fitness function value, the survival of the chromosome is selected. In this process, if the chromosome fitness function value is less than 19 %, the chromosome will not survive for next generation. Further, if the
chromosome fitness function value is in between 19 to 35 %; the, only one copy is considered for the next generation, else two copies are considered for the next generation.

![Roulette Wheel Selection](image)

Figure 5.3: Roulette Wheel Selection

**Accuracy Percentage**

If an 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:

\[
\text{Accuracy percentage} = \frac{y}{x} \times 100.
\]

In our system the user can specify the accuracy percentage. Thus, if none of the RTOS that are available in the system are matching exactly, it is still possible to choose an RTOS that is close to the required one. Accuracy percentage also acts as stopping criteria for the SGA.
5.5 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 are given in Table 5.1.

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Name of the Parameter</th>
<th>Value</th>
<th>Assigned Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Development Methodology</td>
<td>Cross</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>RTOS Supplied as</td>
<td>Object</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Development Host</td>
<td>UNIX</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Standard</td>
<td>POSIX .1</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>Kernel ROM</td>
<td>280K/4M</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>Kernel RAM</td>
<td>500K/4G</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>Priority Levels</td>
<td>512</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>Multiprocess Support</td>
<td>NO</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>Multiprocessor Support</td>
<td>NO</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>MMU Support</td>
<td>NO</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>Royalty free</td>
<td>NO</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>Standard phone support</td>
<td>Paid</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>Preferred phone support</td>
<td>Paid</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>Base Price</td>
<td>7490$</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>Maintenance fee</td>
<td>16% of List Price</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>Task switching</td>
<td>3us to 19us</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 5.1: Weights and Values Specified by the User

In addition, the user is asked to specify the percentage of accuracy for instance 80%. For the above specification, the decoded binary Chromosome is:

0.0.0000.0000.0000.0000.0000.0000.0.0.0.0.0.0.0.0.0.001.000.000

the Fitness Function value is calculated as 34. Each chromosome is represented with the 36-bit length binary string. Each decimal point separates the one specific characteristic of the RTOS that represents its values. 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.
.5.6 Results and Discussions

We have implemented the SGA and experiments were conducted 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. In the following Figure 5.4, Ch1 to Ch14 represent the Chromosome Number (Ch Num 2), Further F Val (Fitness Value) of G1 to F Val of G3 are fitness values of Generation 1, 2, and 3 respectively.

![Fitness Values of Chromosomes with Different Generations](image)

Figure 5.4: Fitness Values of Chromosomes with Different Generations.

Figure 5.4 shows the chromosomes' fitness values with various generations. Has been it is found that the fitness values of the chromosomes are more stable in generation three and they takes more CPU time. Figure 5.5 depicts the CPU time/Population size and shows that more the population size the more the CPU time.
Once 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 which is shown in the Figure 5.6.

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, as shown in Table 5.2.

<table>
<thead>
<tr>
<th>Test No.</th>
<th>NO of Bits for Mutation</th>
<th>CPU Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>0.05</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>0.08</td>
</tr>
</tbody>
</table>

Table 5.2: Mutation Vs CPU Time.
Once again compare the results of the test cases; with constant population with 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 Figure 5.7.

![Percentage of Accuracy vs. CPU Time](image)

**Figure 5.7.** Percentage of Accuracy vs. CPU Time.

As shown by the outputs of various test cases, the Figures [5.4-5.7] depict the results of various parameters, which affect the output of the system in terms of CPU time. It has also been found that even though the output varies considerably from sample to sample, there is tremendous reduction in performance due to increased levels of percentage accuracy. As the number of bits used for crossover and mutation operations increases consequently, the efficiency of consumes the system reduces and consume more time. However, the difference is considerably small, and can be therefore ignored. However its 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.

By studying the output for the most optimal RTOSs based on user specifications, we
came to the conclusion that the most optimal RTOSs were 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 RTOSs matching the specified criterion were fewer. However a higher
percentage of accuracy means a more optimal solution. It gives the developer/ designer a
portal to decide on a real time operating system, which must suit his choice of parameters,
and is the most optimal one available for that purpose, with in a short time.

This method is comparatively more efficient than the elimination techniques, because it
did not consider all the specifications of the RTOS, except the specified ones of the
developer, and needed less time. This system also has a provision to provide percentage
accuracy (i.e., allows the developers to decide how much percentage of guarantee that the
selected RTOS has) and weights for all the specified parameters, so that it selects the
optimal one that exists has in the database.
Features of the Proposed System:

- 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 easily 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 accuracy required, which is also an efficient method compared to other methods which use 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.

Though there are certain advantages in the use of the system; there are also certain limitations.

- 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 to only those RTOSs already provided in the database.

- Although number of bits involved in the crossover and mutation operations is fixed, it also 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 the desired results.
5.7 Summary

Real time applications have become popular these days due to the complexities that exist in the system. To meet those complexities, the developers are given the invaluable task of developing the real time software. There are quite a large number of RTOSs that are available in the market, and one does get confused as to which one provides the more efficient embedded systems-design in terms of cost, power consumption, reliability, speed, etc.

In this chapter, we have 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 a large number of RTOSs. It has user-friendly graphical interface (GUI), through 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 of 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 user’s purpose. The designer has an option of choosing a from pre-defined input or can even specify his/her own input.