CASE 83

Robust Optimization of a Real-Time Operating System Using Parameter Design

Abstract: In this study, a series of L_8 parameter design experiments provide the strategy for selecting the real-time operating system (RTOS) features that will provide optimal performance. Since these parameters were selected specifically for the radio controller architecture and used a realistic test bed, the final product will behave in a predictable and optimal fashion that would not be possible without using Taguchi techniques. This application of Taguchi techniques is not unique to a radio controller but rather can be used with any RTOS and architecture combination. These experiments can form the template for any software engineer to enhance RTOS performance for other applications.

1. Introduction

ITT Industries A/CD is a world leader in the design and manufacture of tactical wireless communications systems. ITT was awarded a major design contract for a very high frequency (VHF) radio. This radio requires a complex controller. As a result of a trade study, the CMX real-time operating system (RTOS) was selected as the scheduler for the radio controller. The CMX RTOS provides many real-time control structure options. Selecting these structures based on objective criteria early in the design phase is critical to the success of this real-time embedded application.

2. Background

Although the RTOS is a commercial off-the-shelf (COTS) product, there are still many features and options to select to achieve an optimal result. The features and options selected are highly dependent on the system architecture. For this reason, a parameter design was undertaken to optimize the selection of these factors.

The VHF radio combines a collection of independent concurrent processes that run using preemptive priority scheduling. In a real-time system, events enter the system through the hardware interrupt. When a hardware interrupt occurs, the software is interrupted automatically. The software services this interrupt through the use of an interrupt service routine (ISR). Once the interrupt is serviced, program control is returned to the processing point prior to the interrupt. The software architecture will rely on the communication between various tasks and ISRs to control the radio.

RTOS vendors typically provide a set of generic communications structures. These structures are then combined to form systems and architectures. Vendors strive to provide a varied set of functions to increase market dominance. Three common communications requirements are (1) ISR to task, (2) task to task, and (3) task to hardware. The system architecture requires these three mechanisms. In each case, there are two operating system constructs for achieving acceptable interprocessor communication. Each is implemented differently and

will therefore affect the system performance differently. To choose the correct constructs, a parameter design is required. Another major concern is the speed at which the processor operates. The faster the processor runs, the more power it draws. It is also critical that the RTOS behave predictably over all processor speeds. Therefore, it is important to choose an experiment that combines the different interprocess communications structures and processor speed.

Figure 1 shows how events are processed in the software architecture. The VHF radio accepts commands and traffic events. Each type of event enters the system via an ISR. This ISR is a low-level assembly language routine that is invoked when a hardware interrupt register state changes. The event, having been registered, must be passed to the tasks that will process the message. There are four different types of message handlers: traffic, background commands, immediate commands, and posttraffic commands; each interacts and behaves differently. Background commands are handled when traffic is not communicating actively with the hardware. As the name implies, posttraffic commands wait until traffic is complete before running. Immediate commands are special tasks that terminate the current traffic and then execute.

Comparing the radio controller architectural requirements to RTOS features results in the following:

Communication Requirement	*RTOS Capability Meeting Requirement*
ISR to task	Mailbox versus queue
Task to task	Mailbox versus events
Task to hardware	Semaphore versus resource

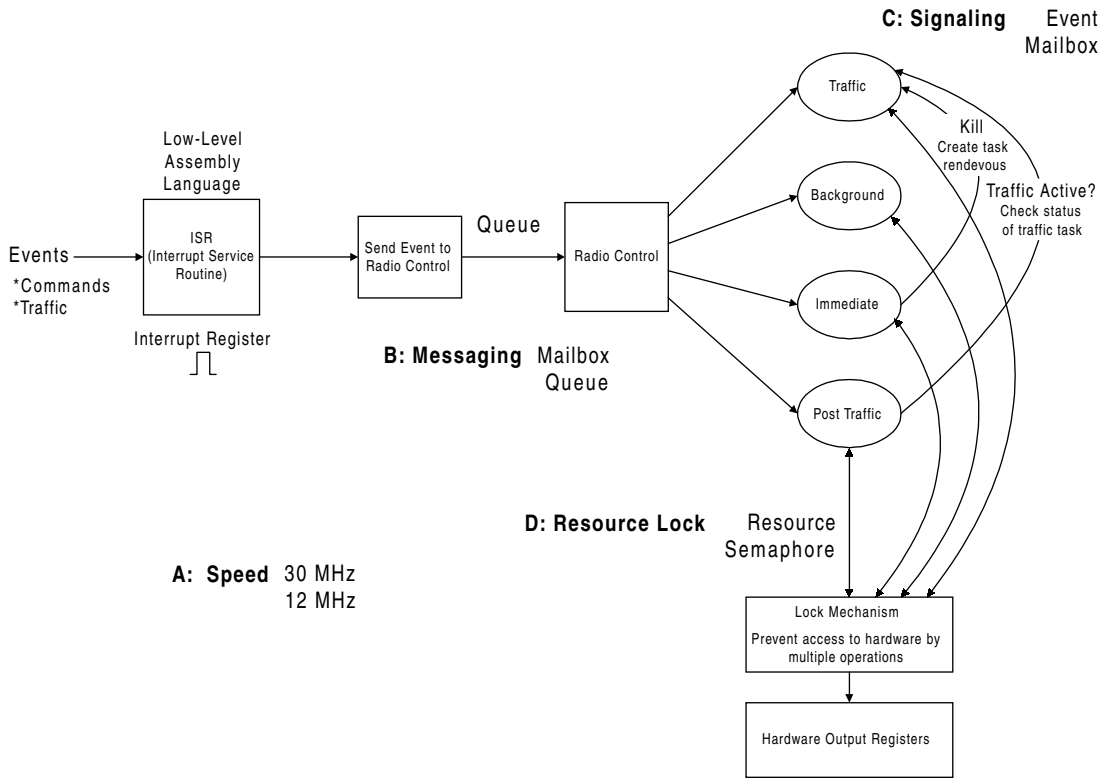


Figure 1
Event processing in the controller architecture

As part of the design phase, functional flows were constructed for every radio control and traffic operation. These functional flows focused on how the architecture processed commands. The ultimate test of the architecture is to handle each functional flow efficiently. Figure 2 shows an example of the volume functional flow.

There is a direct correlation between the functional flow and the architecture. The ISR can be seen as the input mechanism that starts a thread. The queue is represented by the queue column on the functional flow. It queues up requests going to radio control. The signaling mechanism in the architecture is shown in the queue request. Volume is a benign event that is performed as a background process by radio control. The resource lock is shown in the radio control column of the thread. It indicates where the hardware must be locked to prevent hardware contention. If contention were allowed to occur, the radio would lock up, thus failing catastrophically. Each of the primary communications mechanisms is identified clearly in the flow: messaging, signaling, and resource lock. The architecture shown in Figure 1 implements these mechanisms.

Since events are constantly being passed to radio control at a variable rate, they must be queued so that they are not lost. The RTOS provides two mechanisms for passing these events to radio control: mailboxes and queues. The various processing tasks must communicate between themselves to send terminate messages or wait for traffic to be complete. Interprocess communication can be achieved using mailboxes or events. Since each of the processing tasks must have exclusive access to hardware output registers, a lock must be implemented so that there is never simultaneous access to the output registers by more than one task. This can be accomplished by using semaphores or by defining a resource.

3. Objectives

The objective of this experiment was to determine the correct parameters, which will give maximum RTOS performance for the embedded application. The parameter diagram is given in Figure 3. The *noise factors* are as follows:

- *Interrupt mix:* The mixture of different types of interrupts entering the system
- *Interrupt rate:* The rate at which interrupts are entering the system

The control factors are as follows:

- *Processor speed.* This is the speed at which the processor is set. The radio design is trading off the performance at two speeds, 12.8 and 25.6 MHz (12- and 30-MHz oscillators are available and will be used for the parameter design)
- *Messaging.* The system architecture is based on the passing of messages between tasks. Requests enter the system via interrupts which are transformed into message requests, which are acted upon by the system. The two types of messaging schemes being investigated are mailboxes and queues. *Mailboxes* allow a task to wait for a message returning a pointer to the message passed. Any task can fill a mailbox through the use of a mailbox ID. Mailboxes are implemented as a first in, first out. Mailboxes can have multiple ownership. *Queues* are circular first in, first out and last in, first out with up to 32k slots. Slot sizes remain constant and are fixed at compile time. There are many manipulation functions for queues.
- *Signaling.* There are several signals, which must be implemented in the radio controller. An example is for an event to send a terminate signal to the current-running traffic thread. The two methods for efficiently sending messages to tasks are events and mailboxes. *Events* use a 16-bit mask to be used to interrupt a task or allow a task to wait for a specified event or condition to be present. It should be noted that this limits the design to 16 signals per task. *Mailboxes* can be used to allow a task to wait for and then check the message when it is received. This allows more information to be passed.
- *Resource locking.* A key problem with previous controllers was the inability to block competing operations from seizing a resource. This is a particular problem between the controller and the digital signal processors (DSPs). If a second command is sent to the DSPs before the current one is acted upon, the system can, and often will, lock up. By using a resource locking mechanism, this detrimental behavior

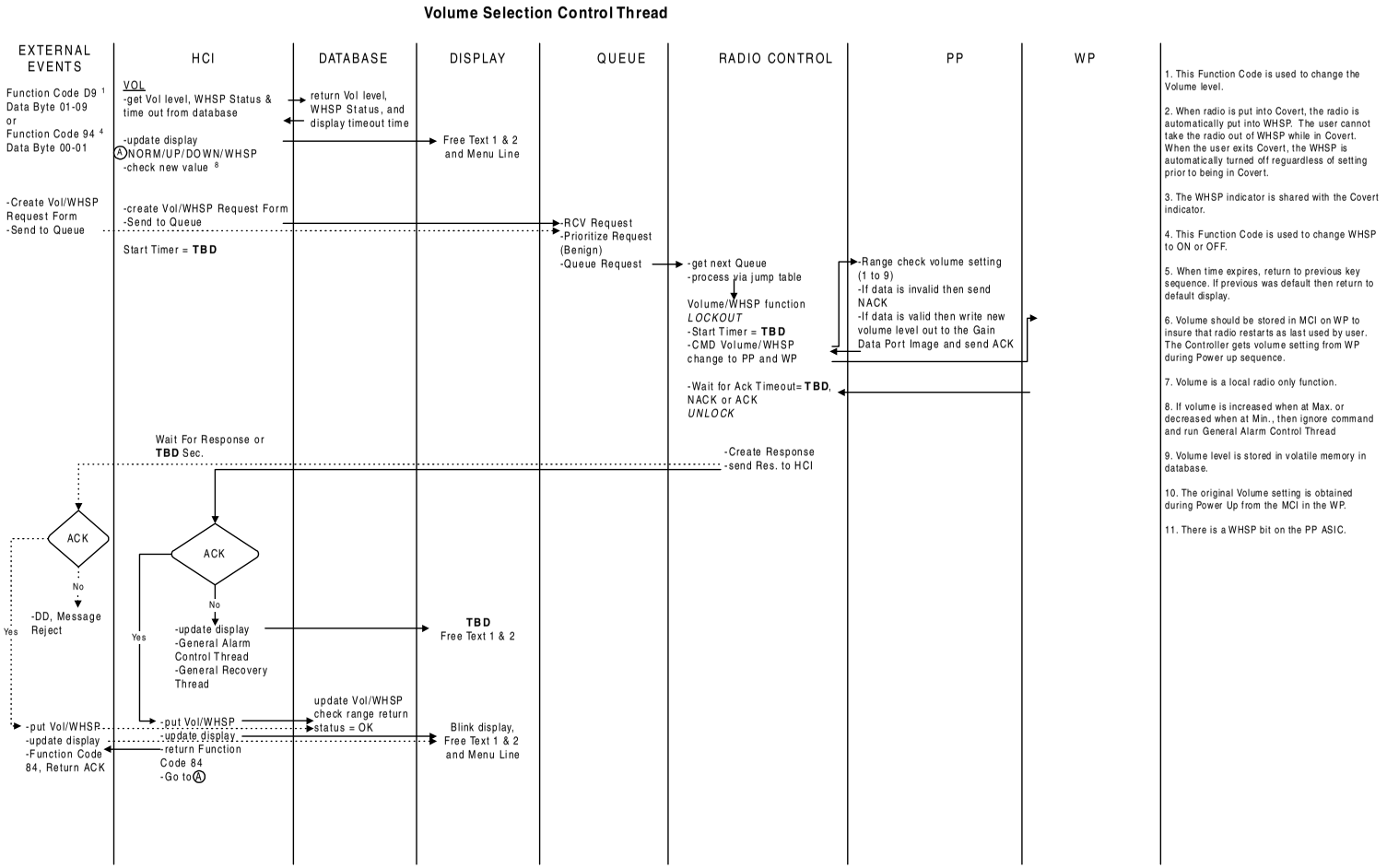


Figure 2
Volume functional flow

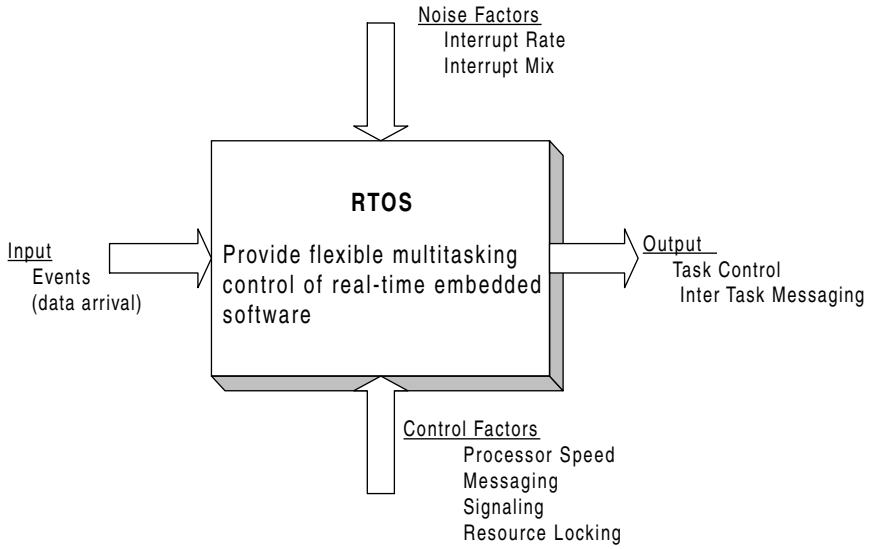


Figure 3
RTOS parameters

can be avoided. The two types of resource locking being investigated are resources and semaphores. *Resources* are RTOS commands that allow a resource to be locked, preventing simultaneous use by competing tasks. This is a binary-state mechanism that can provide a lock and unlock state. The highest-priority task requesting a resource is granted the resource. The locked procedure then inherits the priority of the requester. *Semaphores* are implemented as counting semaphores, which can be used as binary semaphores. Although more flexible because they are counting in nature, our design needs to prevent all secondary access.

The input signal is a series of events. Events enter the system as interrupts, which are then parsed to waiting tasks to execute particular traffic or control threads. The input is t_{ISR}. The *output* was measured as a series of times: (1) the time that the event is queued, (2) the time that the event starts execution and (3) the time that the event completes.

Ideally the system should process messages as fast as possible. Therefore, the ideal system is one that has no latency and completes work in the minimum time. The smaller-the-better method was used for the analysis.

All tests were run using CMX RTOS version 5.2. The RTOS was run on a Pentium PC with a serial Nohau Emulator Model 51XA-PC and a companion trace card EMUL51XA-PC/IETR 128-25. The 80C51XA was run at 12 and 30 MHz. The HiTech C compiler version 7.73 was used to compile and link the code. The emulator was configured for a 16-bit bus and 1-Mg code/data. The default CPU wait state of four or five instruction cycles was used.

4. Experimental Layout

Test conditions are as follows:

Average case:	7 times per sec
Worst case with margin:	30 times per second
Failure point:	*N* times per second (no idle time)

Control factor levels are shown in Table 1. Based on the control factors identified, an L_8 orthogonal array was chosen for the test configuration for the Taguchi experiments. The L_8 2^4 array was chosen to hold the four two-level factors (Table 2).

Table 1
Control factors and levels

	Level	
Control Factor	**1**	**2**
Speed (MHz)	12	30
Messaging	Mailbox	Queue
Signaling	Event	Mailbox
Resource lock	Resource	Semaphore

5. Experimental Results and Analysis

The data were analyzed using a smaller-the-better method. The signal-to-noise (SN) ratio was computed using

$$\mathrm{SN} = -10 \log \frac{\sum (t_{\mathrm{end}} - t_{\mathrm{ISR}})^2}{N}$$

The SN ratio quantifies the composite latency of all message types that the system is expected to handle. The delta time includes from the time the event enters the system until all processing is completed and the thread ends. This includes the time that the event is in the queue, waiting on other events to obtain a resource lock, and the time required to communicate with other tasks (i.e., terminate or wait message times).

Timing measurements were made of individual messages. The times captured were time entering the system, t_{ISR}; time message queued, t_{queue}; time processing starts, t_{start}; and time the processing ended, t_{end}.

The time delta that proved most useful was $t_{\mathrm{end}} - t_{\mathrm{ISR}}$, because it includes the time significant for all factors. The time between entering the system and being queued was in all cases insignificant, proving that the RTOS was handling this efficiently. The time between a message being queued and starting processing focuses on factor *B*, the type of queuing. Similarly, the time between starting and ending processing focuses on factors *C* and *D*. These times were used to validate the conclusions that were made. Therefore, the overall time delta was used since each factor is then represented by the experiment.

These measurements were made using message rates of 7 and 30 times per second. This proved to be the most valuable data collected, since individual threads as well as composite characteristics could be examined. Table 3 lists the results of the SN ratio calculations for 7 and 30 events per second.

Based on the results of the Taguchi experiments and the data reduction, the following factor effects plots were calculated for 7 and 30 events per second. The components of the factor effects plots are obtained by averaging the SN ratio values for each factor as specified in Figure 4. Examining the factors relating to speed, A_1 and A_2 are the averages of experiments 1 to 4 and 5 to 8, respectively.

Table 2
L_8 design experiment

No.	Speed	Messaging	Signaling	Resource lock
1	12	Mailbox	Event	Resource
2	12	Mailbox	Semaphore	Semaphore
3	12	Queue	Event	Semaphore
4	12	Queue	Semaphore	Resource
5	30	Mailbox	Event	Semaphore
6	30	Mailbox	Semaphore	Resource
7	30	Queue	Event	Resource
8	30	Queue	Semaphore	Semaphore

Table 3
SN ratio for $t_{end} - t_{ISR}$

No.	SN 7	SN 30
1	−27.7	−27.7
2	−27.7	−27.7
3	−32.0	−32.3
4	−32.0	−32.6
5	−21.7	−21.8
6	−21.8	−21.8
7	−28.7	−28.4
8	−28.5	−28.3

Factor Effects Summary

The overall objective of this parameter design was to optimize RTOS performance. Examining Figure 4 shows almost identical results for both 7 and 30 events per second. Choosing the correct parameters for 30 events per second will yield a design that is robust over the entire region of performance expected.

The factor effects plot shows that processor speed and messaging are the dominant factors and very comparable in amplitude. Therefore, choosing 30 MHz and using mailboxes for messaging will provide the optimal design.

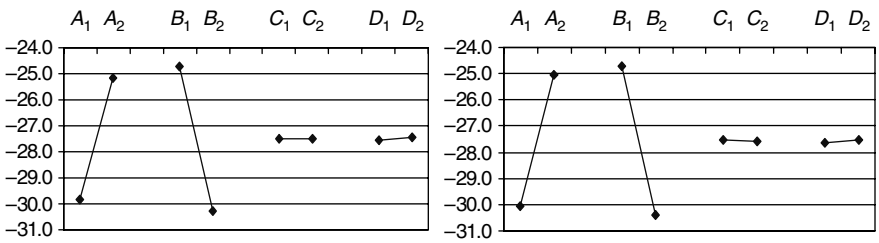
Further examination of the parameter design shows that signaling and resource locking were inconsequential. It appears the RTOS treats each of these methods in a way that guarantees similar performance. These factors can be chosen for ease of design and maintenance.

It is surprising that factor B is more significant than A. It was expected that clock speed was going to show strong dominance over the other factors. What this experiment showed is that the choice of how messages are passed is even more significant to performance than is speed. Choosing the wrong messaging type would totally eliminate the performance gained by a faster processor. Without the experiment, this parameter would have been chosen out of convenience, and significant performance might have been sacrificed.

6. Parameter Design Confirmation

Since parameters C and D are nondominant factors that can be chosen for other design considerations rather than for system optimization, the confirmation can be performed with the data already collected as part of the experiment. Choosing A_2 (30 MHz) and B_1 (mailboxes), one would expect to see a little more than a 10-dB performance increase over choosing A_1 (12 MHz) and B_2 (queues). Table 4 identifies the two scenarios.

As expected, the confirmation results were extremely consistent with the results expected. Since A and B are the dominant factors and C and D show little effect, the internal confirmation yielded positive confirmation that the factors cho-



(*a*) SN 7 Factor Effects (*b*) SN 30 Factor Effects

Figure 4
Factor effects for $t_{end} - t_{ISR}$

Table 4
Design confirmation using factors *A* and *B*

Confirmation Check	7 Events per Second	30 Events per Second
A_2 (30 MHz) and B_1 (mailboxes)	−21.7	−21.8
A_1 (12 MHz) and B_2 (queues)	−32.0	−32.5
Delta	10.3	10.7

sen are the correct factors to optimize the system properly.

7. Performance Analysis/System Stress Test

Flexibility, modifications, and engineering changes characterize the nature of software engineering systems. Software is commonly seen as easily changed compared to hardware, which requires radical modification. As a result, software is seen as the answer to increased product life, longevity, and product enhancement. The life cycle of software is often measured in months. Our parameter design focused on the current expected needs of the VHF radio. Optimum RTOS parameters were chosen that provide excellent performance within the expected range and include design margin.

It is the job of the software engineer to answer additional questions that affect the quality and products being designed. These questions aim at the heart of the reason that products become obsolete and require major upgrades prematurely. Margin is the ability of a system to accept change. Far too often, margin is measured as a static number early in the design phase of a project. Tolerances are projected and the margin is set. This answer does not take into account the long-term changes that a system is likely to see toward the end of its life or the enhanced performance required to penetrate new markets.

Additional information needs to be gathered. How will the system perform when stressed to its limits? Where does the system fail? Figure 5 shows how this system will perform past the expected limit of 30 events per second. The system was set to use mailboxes, events, and resources and run at 12 and 30 MHz. The message rate was increased steadily. The real simple syndication (RSS) of the event time was computed and plotted versus the message rate. Clearly shown in the figure is the fact that 12 MHz will lead to a much shorter product life since the performance cannot be increased significantly, and the steep failure curve. However, 30 MHz shows a system that can almost triple its target event rate and still maintain acceptable performance. The shallow assent of the event processing time RSS indicates that the system will behave more robustly. The 30-MHz system will degrade gracefully rather than catastrophically.

Initially, it was expected that the two graphs would have an identical shape displaced by the 2.5 times speed increase. This is not the case because the real-time interrupts controlling RTOS scheduling operate at a constant rate of 1 millisecond. Since the timer rate is constant, it has a larger effect on the slower clock rate than on the faster clock. This

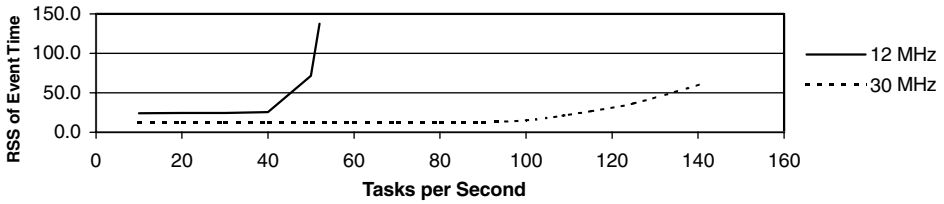


Figure 5
Performance versus event rate

is because the percentage of the processor that is required to process the timer for the 12 MHz is greater than for the 30 MHz. Therefore, when the speed is increased to 30 MHz, not only does the processing power increase but the percentage of the processor available to execute the threads also increases. Ultimately, this provides for a more robust system capable of performing more work to the user at 30 MHz than at 12 MHz.

Using this information, the software design engineer is equipped with one more piece of information from which a cost–benefit analysis can be performed. The cost of the additional part and power must be compared objectively against the benefit to the user in longevity of the radio and performance. The RSS of the event time is significant because, as it increases, the performance degrades to a point where the system is unacceptable. The user will become aware of this when the event time increases past 50 ms.

8. Quality Loss Function

The performance analysis solely looks at performance but fails to account for the cost. The quality loss function provides the software engineer with a way to evaluate objectively the cost to the user of the various clock speeds (Figure 6). The user wants a long-lasting radio that will meet the growing needs of the armed forces. If the radio ceases to meet military needs for expanding services, it will have to be replaced. This could result in a negative customer perception of ITT's radios. Additionally, this might result in customer inconvenience and damaged reputation for the company, which opens the door to increased competition, affecting market share. In addition, there is the financial impact of upgrading the processor to meet the requirements. All of these factors affect quality loss. Taguchi summarized this by saying: "The quality of a product is the (minimum) loss imparted by the product to the society from the time the product is shipped."

Consider the traditional way of viewing quality costs. If the radio meets the specifications that were provided, it has the desired quality. This view does not take into account that a product specification is merely a summary of the known requirements and uses and cannot begin to capture the future military needs. The quality loss function quantifies the quality loss in a formal manner.

The quality loss function is expressed as

$$\text{loss} = \frac{A_0}{\Delta_0^2} t^2 = 0.18t^2$$

where A_0 is the cost of replacing the controller module and Δ_0 is the response time that 50% of users would find unacceptable. In this case the quality loss function represents the cost of the quality loss as the response time grows. This occurs because the user becomes dissatisfied with the product as the response time increases. Table 5 shows the quality loss for the current system. The message traffic and re-

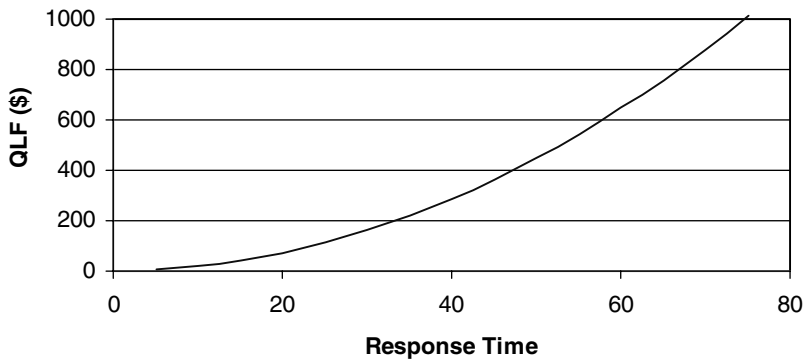


Figure 6
Quality loss function

Table 5
Current system

	Events per Second			
	7 (50%)	10 (40%)	20 (10%)	Quality Loss
12 MHz	24	24	24	$103.68
30 MHz	12	12	12	25.92

sulting loss expected were computed. Table 6 provides the same information for a future system with anticipated growth in processing functions. The quality loss is computed by summing the quality loss at each event rate times the percentage at that rate. As an example, the quality loss of the future system at 12 MHz is

$$\$241.29 = (0.5)[(0.18)(24^2)] + (0.4)[(0.18)(35^2)] + (0.1)[(0.18)(75^2)]$$

These data provide another view of the cost–benefit trade-off:

Current system: $77.76

Future system: $215.37

from which we can evaluate the present and future effectiveness of changing the processor speed. Clearly, if we look only at the current system, $77.76 becomes the cost breakpoint for upgrading the clock to 30 MHz. Upgrading becomes much more attractive when looking at the cost benefit for the future system. The future system provides a $215.27 cost breakpoint, which is a significant percentage of the total controller subsystem cost. The quality loss of the future system makes a compelling argument for upgrading to a 30-MHz processor now. This will return significant dividends in customer satisfaction and long-term market share.

It is important to note that had queues been chosen, there would have been a significant quality loss. Although it costs nothing to change to mailboxes, it results in a tremendous gain when amortized over 20,000 or 30,000 units.

Table 6
Future system

	Events per Second			
	20 (50%)	40 (40%)	50 (10%)	Quality Loss
12 MHz	24	35	75	$241.29
30 MHz	12	12	12	25.92

9. Conclusions

In conclusion, applying robust design techniques and Taguchi methods to the selection of RTOS features for the controller architecture has yielded an optimal selection of key communication structures. These techniques allowed for an optimal design using objective methods for parameter selection. In addition to gaining valuable insight into the RTOS internal operation, several key observations were made and problems discovered early in the design cycle. This study demonstrates that robust design techniques and Taguchi methods can be applied to software engineering successfully to yield optimal results and replace intuition and guesswork. This technique can be applied to all types of operating environments. Although this application used an embedded RTOS, it was applied successfully to the UNIX operating system.

Taguchi's techniques have not only allowed for selection of optimum performance parameters for the VHF radio but have provided a tool for expanding the functionality of this radio and future product lines. The performance analysis and the accumulation analysis provided valuable insight into the robustness of future enhancements. Looking at the quality loss function, a designer can objectively predict how this system will behave under increased workloads. The accumulation analysis shows that the parameter selected will behave optimally even at the system failure point. This gives added confidence that the correct parameters have been chosen not only for this immediate project but also for future projects.

Additionally, it was learned that the RTOS exhibits catastrophic failures in two ways when the queues become full. One type of failure is when requests

continue to be queued without the system performing any work. This is observed by the idle time increasing with tasks only being queued and not executed. The other type of failure is when the queue fills and wraps around, causing requests to become corrupted. By performing these experiments during the design phase, these failures can be avoided long before code and integration.

Finally, it was observed that when events have properties that couple them, as seen when one event waits on the result of another event to complete, the dominance of processor speed diminishes at the slower event rates. This confirms that the system would respond well to processor speed throttling at low event rates.

Using these optimization techniques resulted in a 10.7-dB improvement in system response time by using the optimal parameter set. Using optimal parameters reduces the message latency from 40.6 ms to 12.1 ms, a decrease of 28.5 ms. A performance analysis and quality loss function showed that there are significant cost and performance benefits in choosing 30 MHz over 12 MHz for both the current system and future systems. The quality loss for the current and future systems is $78 and $215, respectively, since the added cost of changing to a 30-MHz processor is minor, as compared to the quality loss it is a highly desirable modification. Finally, getting the equivalent power of a 30-MHz processor, with speed throttling during idle, will require only a 1% power increase with the same board area.

This case study is contributed by Howard S. Forstrom.