Accounting for Clock Frequency Variation in the Analysis of Distributed Factory Control Systems

William Henderson  David Kendall  Adrian Robson
School of Computing and Mathematics
University of Northumbria
Ellison Place
Newcastle upon Tyne, UK
EMail: {william.henderson,david.kendall,adrian.robson}@unn.ac.uk
Phone: 0191 227 3512, Fax: 0191 227 3662

Abstract

Micro-controllers are now widely deployed as components in distributed systems. Temporal predictability is important for such embedded systems - software must execute within specified bounds to maintain the safe operation of the system as a whole. Micro-controllers require regular clock pulses, usually provided by an external quartz crystal. Clocks exhibit random variation in resonant frequency from one component to another, a temperature sensitivity and gradual changes with time. This means that the execution speed of software will vary when implemented on different processors of the same class. We highlight the vulnerability of standard scheduling analysis in the presence of clock frequency uncertainty when applied to the performance prediction of distributed embedded systems. We propose a modified scheduling analysis to account for the inevitable range of processor clock rates in embedded distributed systems and confirm our analysis by a small empirical study using CAN (Controller Area Network) for inter-processor communication.

1 Introduction

Micro-controllers are now widely deployed as components in domestic equipment, in industrial automation controllers, in automobiles, trains, aircraft, lift controllers, communications systems and elsewhere. Some of these examples can be classified as hard real-time because software executed on the micro-controllers must meet strict time deadlines to fulfill its purpose. Indeed, failure to meet a timing deadline may result in large financial loss or even loss of life. For example, software employed to control functions such as airbag deployment or anti-skid braking in automobiles must execute on each and every occasion within strict deadlines; failure to do so would constitute a serious risk to automobile users.

Micro-controllers require regular clock pulses, usually provided by an external quartz crystal device which oscillates at a fixed frequency. Current quartz devices are available with high stability and close tolerance. Nevertheless, clocks exhibit random variation in frequency from one component to another (resulting from inevitable manufacturing variations), a temperature sensitivity and gradual changes with time. This means that the execution speed of software will vary when implemented on different processors chips of the same class and will also be sensitive to both operating temperature and time. These variations are small and in many cases will not have a significant impact on system performance.

A well-established means of predicting the performance of real-time systems is a schedulability analysis which is applicable to both uniprocessor systems [1] and distributed systems [13]. In this approach it is assumed that the computation times of tasks and real-time operating system scheduling functions can be determined a-priori. Usually it is the worst-case performance or longest execution time that is measured/computed since this is required to demonstrate that a given system is able to respond within a deadline. For some applications, where periodic output is required, the shortest execution time is also of interest. It is assumed that computation time bounds completely characterise the execution behaviour of particular software components on given processors and that these can be used to predict behaviour on any processor/real-time operating system of the same type.

In our analysis we assume the use of “simple” processors and preclude performance enhancing features such as pipelining, caching or DMA. Although these processor features improve average performance, they complicate the analysis required to predict performance and result in wide bounds on computation times [7]. The execution time of a software function on a simple processor is determined by the number and nature of instructions executed on a given path (frequently the longest and shortest paths...
The clock speed of the computer processor and the access time of memory devices determine the duration of instructions. Thus, software computation times may be determined by instruction counting. This may be performed by analysing object code or high-level language programs [4] assuming a particular clock frequency, thus fixing the duration of each instruction in the instruction set. This frequency will be the nominal clock frequency appropriate for the target processor on which the code is to be executed. Alternatively, computation times may be determined by measurement on a particular processor. The execution time of a software component may be measured by executing it in isolation, i.e., not in an environment where it may be preempted by other tasks. Care must be taken to ensure that both the data state and event sequence used for the test result in the worst-case path through the software; this may be quite difficult to arrange in practice. Measurements will be made on processors which are clocked at frequencies particular to the actual clock device used, their age and operating temperature. Thus, these measurements will in part reflect the speed of the clock device used.

This paper presents the results of an investigation of the influence of system clock frequency uncertainty on the veracity of performance predictions made using standard scheduling analysis. We shall demonstrate that, as a result of variations in clock frequency between clocks, there are circumstances in which real-time performance might be significantly poorer than expected. This is particularly apparent for distributed embedded control systems in which task execution is managed by priority scheduling and tasks communicate by message passing over a network. In this situation, even quite small changes in software execution times could result in disproportionately large variations in response time because of the non-linear dependence of task response time on computation time under priority scheduling.

The remainder of our paper is organised as follows. Section 2 introduces the quartz clock as a means of regulating the clock of a microcontroller and defines the performance of typical devices. Section 3 reports the results of measurements made on a number of typical embedded microcontrollers to determine the range of clock rates experienced in practice. Section 4 outlines the computational model used to represent systems of tasks preemptively scheduled on processors and communication between tasks required to model distributed systems. In Section 5 the implications of clock period variation on scheduling behaviour and task response times are explored. A small empirical study is described in Section 6 as a practical demonstration of the impact of clock speed variation on system performance and to confirm the results of analysis. Section 7 summarises the work and presents some recommendations regarding the handling of clock speed uncertainty in the analysis of distributed real-time systems.

## 2 The Quartz Crystal Clock

Micro-controllers usually incorporate the active part of a clock device - the oscillator-inverter. This must be connected to an external passive component which oscillates at a fixed frequency. The “clock” used to generate timing signals for popular micro-controllers is usually a Quartz (silica - a hexagonal crystalline mineral) device. Such crystals perform a mechanical vibration when stimulated by an electrical current - the piezoelectric effect [3].

The characteristic frequency of vibration of a crystal depends on the direction of cut and on size. Quartz crystals are manufactured to generate timing signals at fixed frequencies and are available with a wide range of nominal frequencies. However, current manufacturing techniques result in devices possessing characteristic frequencies which are scattered about a nominal frequency. Thus, if a typical device is used to control a clock by counting crystal vibrations, the clock time will drift (slew) away from a standard clock. The drift rate, $\rho_i$, for clock $i$ with respect to a standard clock may be defined as [11]:

$$\rho_i = \frac{\lambda_i}{\lambda_0}$$

where $\lambda_0$ is the period of the standard clock and $\lambda_i$ is the period of clock $i$. A slow clock will be characterised by $\rho_i > 1$ and a fast clock by $\rho_i < 1$. We shall drop the $^0$ superscript from $\rho_i^0$ with the understanding that all quoted drift rates are measured relative to a standard clock to which nominal periods and computation times relate. We define limiting values of the drift rate to accommodate the range of drift rates experienced in practice. Thus, $\rho_{\text{min}}$ is a minimal drift rate for the fastest processor, and $\rho_{\text{max}}$ is the maximal drift rate for the slowest processor.

The tolerance of a quartz device is quoted at a standard temperature (the environmental temperature of the device) and is typically $\pm 30$ ppm at a standard temperature of perhaps $25^\circ$C. Such a tolerance permits the frequency of a 10 MHz device to lie within the bounds \([9999700,10000300]\) Hz. Another characteristic of the quartz clock is its sensitivity to temperature. The frequency of a quartz device typically will vary by as much as $\pm 50$ ppm over its operating temperature range (say, $-10^\circ$C to $+60^\circ$C). The so-called “AT” cut crystal offers a flat temperature profile around room temperature and is the most frequently used for microprocessor regulation. Temperature induced frequency changes would be expected to play a more significant role in applications where processors experience a wide variation in thermal conditions. For example, micro-controllers used in automotive or industrial control may experience wide temperature cycling during routine operation. Quartz crystals also suffer a slow drift in their resonance frequency over

\footnote{Maximum drift rates for resonators lie between $\pm 10^3$ ppm to $\pm 10^{-1}$ ppm, depending on quality (and price), where ppm means ‘parts per million’}
time, particularly during the first year or so after manufacture [3]. Performance data characteristic of typical modern Quartz devices are listed in Table 1. In summary, the resonant frequency of a Quartz crystal device is primarily a function of manufacturing variation and temperature but time also plays a part in defining the operating frequency.

3 Clock Period Variations in Practice

In order to assess the range of drift rates exhibited by typical devices and to identify examples of slow and fast clocks, measurements of clock drift rate were made on a number of micro-controllers boards of the same design. For this purpose, M68HC08AZ32 micro-controllers were selected; these were clocked at a nominal frequency of 4.9152 MHz. The M68HC08AZ32 is an 8-bit micro-controller which provides a CAN field bus interface and other parallel/serial I/O and ADC functions. The processor is designed for automobile applications and is typical of modern devices used in embedded systems. Fifteen processors were available for this study; these are identified here by a number between 1 and 15.

The relative clock speed of each crystal/microcontroller was determined by executing a periodic timer interrupt handler on each of two processors drawn from the 15. The interrupt handler simply raised an output signal on a parallel port for the duration of the handler; this signal was observed on an oscilloscope. The oscilloscope was triggered by one of the traces; the two traces were observed to move relative to one another at a rate determined by the relative clock drift rates of the two processors. The processor with the faster clock generated interrupts more quickly and its output trace was generated with a shorter interval. Thus, the high frequency clock trace regressed relative to the lower frequency clock trace. For each pair of processors, the time delay required to result in a displacement between traces of typically 500μs was determined (with an estimated accuracy of ±5%). All measurements were made at a similar room temperature to minimise the temperature effect on clock frequency.

Figure 1 displays the measured clock drift rates for the 15 processors. The data have been plotted relative to the clock rate of processor number 4; thus the clock for processor 4 is shown to have a negligible drift rate ($\rho_4 = 1.0$). It has been assumed that processor 4 exhibited no clock drift (simply because this processor exhibited a clock frequency approximately mid way between the fastest and slowest). The results show what appears to be a random variation in clock speed of about ±20 ppm; this is within the range quoted by manufacturers which is typical of these devices when operated at a fixed temperature. Both temperature variations and aging could significantly increase measured frequency variations between devices. It can be seen that the devices numbered 6 and 9 were at the extremes of those measured and differ in frequency by about 40 ppm (equivalent to about 3.5 s per day).

4 Computational Model and Assumptions

We review the computational model chosen to represent the behaviour of sets of tasks preemptively scheduled on processors and the principal results of scheduling analysis to compute the maximal response times of tasks.

The type of distributed systems we consider are compositions of tasks scheduled on processors and messages scheduled on Controller Area Networks [9]. Both tasks and messages are statically allocated to resources and, in order that they are amenable to a simple scheduling analysis, they are periodic or sporadic (have minimum inter-arrival times).

4.1 Tasks and Scheduling

The notation adopted to define task sets is as follows. Let $P$ be a set of processors, and $n$ be the total number of tasks in the system. Tasks are identified by an integer in the range $1..n$. The task set $T$, is defined as

$$T = \{(T_i, C_i, \pi_i, p_i) \mid 1 \leq i \leq n\}$$

where $T_i$ is the period of task $i$, $C_i$ is the worst-case computation time of the task, $\pi_i$ is its priority and $p_i \in P$ is the processor to which task $i$ is allocated. The set $T_p$ of tasks allocated to the processor $p \in P$ denotes the set

{| Table 1. Typical Quartz Crystal Characteristics |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Temp. stability</td>
<td>±30 ppm</td>
</tr>
<tr>
<td>Aging</td>
<td>&lt; ±5 ppm in 1st year</td>
</tr>
<tr>
<td>Operating temp. range</td>
<td>−10°C to +60°C</td>
</tr>
</tbody>
</table>

Motorola evaluation board: M68HC08AZ32EV

Using crystals: CQ 4.9152

Footnotes: 
3/Motorola evaluation board: M68HC08AZ32EV
3/Using crystals: CQ 4.9152
\{r_i \in T \mid p_r = p\}. Task periods and computation times expressed above are nominal values for which standard clock rates are assumed. Processor \( p \) has an associated clock speed, \( \rho_p \), which dictates the execution rate of all tasks scheduled on that processor and the periods of responses originating on that processor.

Tasks are scheduled according to the fixed-priority preemptive policy in which priorities are totally ordered. A task arrives infinitely often at the start of its period at which point it is logically ready to be scheduled for execution. However, a task may suffer a delay following arrival before it is released and can be entered into a priority ordered run queue (see Fidge [5] for a comprehensive review of the semantics of task scheduling). This delay, called release jitter, is the maximal difference between arrival time and release time; it results from variations in response times of tasks and messages which precede the given task in a distributed transaction. Our model allows tasks to be triggered by clock, interrupt or by the arrival of a message handled by the real-time kernel.

The modelled scheduling and interrupt behaviour matches that supported by popular real-time kernels and processors (e.g., VxWorks [14] and 68000 CPU) and provides timely execution of sporadic, high priority, interrupt-driven tasks and restricted periodic behaviour for other tasks. The time delays associated with context switching are assumed to be included as part of the computation time of tasks. Although a more sophisticated treatment of context switching is possible [2], the analysis reported here will concentrate on the clock speed issue without compromising the generality of our results. The analysis may be generalised to accommodate context switching in a straightforward manner.

### 4.2 Inter-Task Communication

Two mechanisms for inter-task communication are considered - tasks on the same processor communicate with the support of kernel services and tasks on different processors communicate via one or more Controller Area Networks (CAN) [9], a fieldbus widely used in automotive and factory automation applications. Let \( \mathcal{N} \) be a set of networks, and \( m \) be the total number of messages in the system. Messages are identified by an integer in the range \( 1..m \). The message set, \( \mathcal{M} \), is defined as

\[
\mathcal{M} = \{(T_i, S_i, \pi_i, q_i) \mid 1 \leq i \leq m\}
\]

where \( T_i \) is the period of message \( i \) (assuming a nominal clock speed on the processor which transmits the message), \( S_i \) is its length, \( \pi_i \) is its priority and \( q_i \in \mathcal{N} \) is the network to which message \( i \) is allocated. The set \( \mathcal{M}_q \) of messages allocated to the network \( q \in \mathcal{N} \) denotes the set \( \{j \in \mathcal{M} \mid q_j = q\} \). The period of a message is inherited from the task which transmits it. Tasks may communicate with each other in a restricted way by asynchronous message passing. Broadcast communication semantics are assumed: tasks cannot be blocked when they transmit a message and more than one task may receive the same message.

### 5 Schedulability

The calculation of worst-case task response times for fixed priority, preemptive scheduling on uniprocessors is well documented, for example [1, 2, 5, 10]. The basic analysis has been extended to allow distributed systems performance evaluation [12, 13]. Recently, the calculation of best-case response times for distributed systems has also received attention [6, 8] although we shall confine our interest here to worst-case response time calculation. The worst-case local response time, \( r_i \), of a task \( i \) is measured from the point a task arrives at the beginning of its period to the end of its computation. Local response times may be computed using the following formula [5]:

\[
r^*_i = C_i + \sum_{j \in hp(i)} \left[ r_j + J_j \right] C_j
\]

where \( hp(i) \) is the set of tasks with priority higher than task \( i \). Equation 1 is iterated until convergence by replacing \( r_i \) with \( r^*_i \) on each iteration until \( |r_i - r^*_i| \leq \epsilon \) where \( \epsilon \) is a convergence criterion, typically 1 \( \mu s \). An initial response time of \( r_i = C_i \) may be assumed. We have excluded from the analysis the time delays associated with context switching in the real-time kernel. Task jitter, \( J_j \), of task \( j \) is the variation in arrival time of a task resulting from variations in scheduler performance and interference to predecessor tasks and messages in a transaction. Schedulability of a task is determined by comparing its computed response time with its period (or deadline if this is shorter).

#### 5.1 Nominal Task Periods and Computation Times

We shall examine the impact of clock period variation on task response times by way of a simple example. Figure 2 illustrates a small distributed system composed of

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Priority</th>
<th>Period ( T ) [ms]</th>
<th>Computation ( C ) [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>p.t1</td>
<td>0</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>q.t1</td>
<td>0</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>q.t2</td>
<td>1</td>
<td>100</td>
<td>9</td>
</tr>
</tbody>
</table>
two processors, \( p \) and \( q \), and a network, \( net \). A single task on \( p \) called \( p.t1 \) executes periodically and transmits a message on a network. When the message \( net.m1 \) is received on processor \( q \) it causes the arrival of the task \( q.t1 \). The sequence \( p.t1 \rightarrow net.m1 \rightarrow q.t1 \) is termed a distributed transaction. Task \( q.t2 \) is the lower priority periodic task on processor \( q \) and may be preempted by the task \( q.t1 \). Table 2 specifies the behaviour of three periodic tasks preemptively scheduled on a two processors.

![Figure 3. Worst-case schedule for task \( q.t2 \) - Nominal Clock rates](image1)

Figure 3 illustrates the execution behaviour of the task set on processor \( q \) assuming a worst-case condition for the lower priority task in which both tasks arrive at the same instant - the critical instant. The higher priority task will always complete in 1 ms because it cannot be preempted and will be executed immediately on arrival. Task \( q.t2 \) will be released at \( t = 1 \) ms resulting in a response time of 10 ms; no other task phasing can result in a longer response time. However, task \( q.t2 \) can be executed without preemption in 9 ms if it arrives at the instant task \( q.t1 \) completes its computation. In this simple example it is assumed that tasks arrive periodically, suffer no release time jitter and that context switching overheads are negligible.

It is instructive to examine the response time bounds of task \( q.t2 \) assuming that its computation time may be varied. Figure 4 shows both the minimal and maximal response times of task \( q.t2 \) as functions of its computation time. The maximal response time of the task is never less than 1 ms since it can be preempted by task \( q.t1 \). With a computation time of \( C_{q,t2} < 9 \) ms, the minimal response time of task \( q.t2 \) is simply its computation time. With a computation time of 9 ms, the response time may lie in the range \([0,11]\) ms. This behaviour repeats for a computation time of 18 ms. Note that \( r_{q,t2} \) expressed as a function of \( C_{q,t2} \) is piecewise linear with discontinuities at particular values of \( C_{t2} \). It is at these discontinuities that scheduling analysis is vulnerable if clock periods differ from the nominal rates. We shall now examine the impact of clock drift on scheduling analysis for distributed systems.

### 5.2 Processor clock frequency variations

The effect of clock period uncertainty on the response times of tasks in distributed systems depends on the method of triggering tasks. In general, distributed systems will include tasks which are triggered by local clocks and others which are triggered externally. The execution rate of tasks located on a range of different processing hardware will vary as a function of the local clock frequency. Referring to our example distributed system (Figure 2), we shall initially assume that the clock on processor \( p \) is accurate (\( \rho_p = 1.0 \)) and that the clock on processor \( q \) is slow by, a much exaggerated, 10% (\( \rho_q = 1.1 \)). The period of a distributed transaction is influenced by the clock speed of the processor on which the initial task is scheduled. Even if the frequency of the transaction \( p.t1 \rightarrow net.m1 \rightarrow q.t1 \) is accurate because processor \( p \) has an accurate clock, the schedulability of tasks on processor \( q \) may be reduced if processor \( q \) is slow.

Figure 5 illustrates the schedule for the two task system under the conditions that task execution times are extended by 10% as a result of the low frequency of the local clock on \( q \). However, tasks arrive periodically at the nominal rate because task \( t1 \) is part of a distributed transaction which begins on another processor with an accurate clock. Notice that the second arrival of task \( t1 \) is at 10 ms because tasks periods are not extended in this example. In the worst case, task \( t2 \) now suffers two preemptions from task \( t1 \); the worst-case response time of task \( t2 \) is \( r_{t2} = 12.1 \) ms (cf., \( r_{t2} = 11 \) ms if both periods and computation times vary). Clearly, if the arrival periods of tasks can be subject to variations in frequency of different clocks, task \( t1 \) could arrive at a higher frequency than the
nominal rate; this could further increase the worst-case interference to lower priority tasks. Thus, a task suffers the worst-case preemption if higher priority tasks arrive at a frequency higher than the nominal frequency and the preempted task executes on a processor with a slow clock. In summary, there are two mechanisms which can cause increased task interference as a result of clock rate error:

1. If a distributed transaction with a nominal period visits a processor with a low clock speed, increased preemption to lower priority tasks will take place;

2. If a distributed transaction begins on a processor with high clock speed, all tasks and messages on that transaction can cause increased preemption to lower priority tasks and messages on other processors with nominal or lower clock speeds.

These two sources of error are cumulative and can conspire to cause significant error in the analysis of distributed systems.

If task arrival periods and task computation times are functions of different clock drift rates, worst-case response times of tasks may be evaluated by modifying Equation 1 to replace computation time, \( C_i \), by \( C_i \rho_{\text{max}} \), where \( \rho_{\text{max}} \) is the maximal drift rate (of the slowest processor clock likely to be experienced in practice). Task periods are also a function of clock rate, so the task period and jitter, \( T_j \) and \( J_j \), are replaced by \( T_j \rho_p \) and \( J_j \rho_p \) respectively, where \( \rho_p \) is either the minimal or maximal clock drift rate. The maximal clock drift rate is selected if task \( \tau_j \) is part of a transaction which originates on the same processor as task \( \tau_i \), since the period and jitter of \( \tau_j \) will be influenced by the local clock rate. Conversely, if the transaction involving \( \tau_j \) originates on another processor, the period and jitter may be shorter than the nominal values and the minimal clock drift rate is selected. These changes are reflected in Equation 2.

\[
r_i^* = \rho_{\text{max}} \left\{ C_i + \sum_{j \in \text{tp}(i)} \left[ \frac{r_i + J_j \rho_p}{T_j \rho_p} \right] C_j \right\} \tag{2}
\]

where

\[
\rho_p = \begin{cases} 
\rho_{\text{max}} & \text{if the transaction involving } \tau_j \text{ originates on } p_i; \\
\rho_{\text{min}} & \text{if the transaction involving } \tau_j \text{ does not originate on } p_i.
\end{cases}
\]

In general, this calculation of response time in the presence of clock speed error will be pessimistic since not every part of each distributed transaction will necessarily be executed on processors with extreme clock rates. Thus, a portion of the accumulated jitter on a distributed transaction may be less than \( J_j \rho_{\text{min}} \) and response times will be over-estimated because of this.

The 10% clock drift employed in this example is about three orders of magnitude greater than typical drift rates of current processor clocks. However, the principle relating clock drift to scheduling behaviour is established correctly in the example. In practice, even clock drifts of only a few ppm can be responsible for large differences in response time because of the non-linear nature of the response time function.

### 6 Empirical Study

As a practical demonstration of the importance of clock rate error in the analysis of distributed systems, and as a means of verifying the proposed analysis, a simple empirical study was undertaken. The example system was implemented using two 68HC08 micro-controllers of the type discussed in section 3. The example chosen was of the small distributed system depicted in Figure 2 composed of two processors, \( p \) and \( q \), and a CAN network, \( \text{net} \). A periodic distributed transaction begins on processor \( p \) with the execution of task \( p.t1 \) which, on its completion, transmits a single message, \( \text{net}.m1 \). The arrival of the message releases the task \( q.t1 \) and the distributed transaction ends with the completion of task \( q.t1 \). A second task of lower priority, \( q.t2 \), executes on processor \( q \); this may be preempted by the arrival of task \( q.t1 \). The nominal periods and computation times of the system components are listed in Table 3. The message \( \text{net}.m1 \) contained a single byte of data and was transmitted at a speed of 10 kbps. The data content of the message did not change between transmissions; consequently its transmission time of about 6.3 ms never varied. The distributed transaction starting with task \( p.t1 \) arrived with a nominal period of 1s. Thus the task \( q.t1 \) also arrived with the same nominal period. The computation time \( C_{q,t2} \) was chosen to be as long as possible without incurring a second preemption (i.e., \( C_{q,t2} = T_{q.t1} - C_{q,t1} \)), this behaviour is depicted in the timing diagram of Figure 6. It can be seen that task \( q.t2 \) was designed to complete just before the second arrival of task \( q.t2 \). It is assumed for the sake of this analysis that task \( q.t2 \) meets its timing obligation if the system is implemented on processors which are clocked at the nominal rate (i.e., with little or no clock drift).
Table 3. Nominal task/message parameters for the simple distributed system

<table>
<thead>
<tr>
<th>Task</th>
<th>Nominal Period, T</th>
<th>Nominal Computation, C</th>
</tr>
</thead>
<tbody>
<tr>
<td>q t1</td>
<td>1000 ms</td>
<td>70 ms</td>
</tr>
<tr>
<td>q t2</td>
<td>5000 ms</td>
<td>930 ms</td>
</tr>
</tbody>
</table>

6.1 System 1

The example system was first executed using the 68HC08 micro-controllers numbered 4 and 7 (see Figure 1); Table 4 specifies the task allocation adopted. These processor/clock combinations exhibited clock periods which differed by less than 1 μs per s. In order to create the critical conditions which resulted in worst-case responses of task q t2, both tasks on processor q were released together. The computation time of task q t2 was adjusted to avoid a second preemption by task q t1. The scheduling of task q t2 was studied to confirm that only one preemption by task q t1 took place. No appreciable difference in schedulability was observed when the task allocations on the micro-controllers were reversed. This established a base case with nominal clock rates to allow an examination of the variation of clock rates on response times to be studied. The response time for task q t2 with nominal clock rates is denoted by $r_{q,t2}^{nom}$ in the results.

6.2 System 2

In this configuration, the distributed transaction began on a slow processor (number 6) and continued on a fast processor (number 9). Thus, the period of the distributed transaction was longer than the nominal period (of 1 s) resulting in increased interference to task q t2. In addition, the clock on processor q was faster than nominal. These two factors should each result in increased schedulability on processor q. The analysis indicated by Equation 2 was performed using the clock drift rates measured for the two processors (see Figure 1) to determine the response time of task q t2 for this allocation. The slower processor, p, operated with a clock drift of $\rho_{min} = 0.99998$ and the faster processor, q, operated with a clock drift rate of $\rho_{max} = 1.000016$. The change in response time for task q t2 between the systems 1 and 2 was also determined empirically. The increased schedulability on processor q was confirmed both by the empirical study and by analysis as indicated in the results in Table 4. The response time of task q t2 is approximately 20 μs shorter than the nominal case, $r_{q,t2}^{nom}$, and could increase by approximately 40 μs before a second preemption by task q t1 would take place.

6.3 System 3

For the final test, tasks were assigned to processors in reverse order to experiment 2 as indicated in Table 4. In this configuration, the distributed transaction began on a fast processor and continued on a slow processor. Thus, the period of the distributed transaction was shorter than the nominal period (of 1 s) resulting in increased interference to task q t2 and reduced schedulability on processor q. The slower processor, q, operated with a clock drift of $\rho_{min} = 0.99998$ and the faster processor, p, operated with a clock drift rate of $\rho_{max} = 1.000016$. The response times determined by both empirical and analytical means show that task q t2 suffered a second preemption by task q t1 in this system configuration, i.e., the response time $r_{q,t2}$ included a second instance of the computation time $C_{q,t1}$.

6.4 Summary of Results

The small differences between the analytical and empirical values of response times for systems 2 and 3 are within the expected measurement error bounds. We have chosen quite long periods for tasks in the experimental system in order to generate easily measurable errors in response times. However, the magnitude of any analytical error resulting from clock drift uncertainty will very much depend on the computation times of higher priority tasks and not simply on periods. The results confirm that system performance may under some circumstances depend on the random variation in clock drift rates between processors of the same class, and that system responses may miss their deadlines because of this. Furthermore, it is possible to predict the effects of clock drift on performance if individual clock drift rates are available, otherwise minimal and maximal bounds may be assumed. The potential for error in applying standard scheduling analysis in computing response times of distributed transactions has been demonstrated.

7 Conclusions & Recommendations

The performance of distributed embedded systems may vary simply as a result of the random allocation of clock crystals when using otherwise identical processors. The response times of tasks in a multitasking preemptive environment are not linearly related to their computation times except for the highest priority task; all tasks other than the highest priority task will suffer execution time delays which will exhibit discontinuities. The errors in computed response times introduced by uncertainties in clock rate can be significant in a priority preemptive scheduling environment if response times happen to be close to critical points. Low utilisation alone is no guarantee that clock speed variations will not compromise scheduling predictions. It is the presence of critical points which matters and not the load on the processor. The following points summarise the results and offer some recommendations for coping with clock rate error in schedulability analysis:

1. Uniprocessor scheduling results are little affected by clock drift errors unless the periods of tasks are determined by clocks external to the processor.
Table 4. Three experimental systems

<table>
<thead>
<tr>
<th>System number</th>
<th>Task Allocation</th>
<th>Empirical ( r_{\text{emp}} - r_{\text{nom}} )</th>
<th>Analytical ( r_{\text{ana}} - r_{\text{nom}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{p.t1} on processor #4 (nom) {q.t1, q.t2} on processor #7 (nom)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>{p.t1} on processor #6 (slow) {q.t1, q.t2} on processor #9 (fast)</td>
<td>-22\mu s</td>
<td>-20\mu s</td>
</tr>
<tr>
<td>3</td>
<td>{p.t1} on processor #9 (fast) {q.t1, q.t2} on processor #6 (slow)</td>
<td>+70013\mu s</td>
<td>+70017\mu s</td>
</tr>
</tbody>
</table>

2. The analysis of distributed systems is particularly vulnerable to clock period uncertainty.

3. The risk of analytical error is present even when processors are lightly loaded, although the magnitude of any error is likely to be greater at high utilisation.

4. Schedulability analysis can be extended to account for worst-case clock rate variation; necessarily, this widens the bounds that can be placed on response times.

5. If task computation time bounds are determined empirically, care must be taken to ensure that the clock frequency of the test processor is close to the nominal rate.

6. Long computation times and periods exacerbate the problem since errors associated with clock skew accumulate. Thus, the analysis of low frequency systems is more vulnerable to clock rate uncertainty.

A programme of further work includes an investigation of temperature effects on schedulability, the computation of lower bounds on response times in a similar way to [8] and a study of how network clock speed (and therefore message transmission times) are influenced by clock rate drift.

References


