American Journal of Engineering Research (AJER)
e-ISSN: 2320-0847 p-ISSN : 2320-0936
Volume-5, Issue-9, pp-93-104
www.ajer.org
Research Paper Open Access

Applying the Scheduler Test Case Technique to Verify Scheduler Implementations in Multi-Processor Time-Triggered Embedded Systems

Mouaaz Nahas[1],Ricardo Bautista-Quintero[2]
[1]*Department of Electrical Engineering, College of Engineering and Islamic Architecture, Umm Al-Qura University, Makkah, Saudi Arabia*
[2]*Department of Mechanical Engineering, Instituto Tecnologico De Culiacan, Sinaloa, Mexico*

ABSTRACT: *We have recently introduced a technique called "scheduler test case" (STC) as a practical means for bridging the gap between scheduling algorithms and scheduler implementations in time-triggered embedded systems. The technique was originally applied to single-processor embedded designs employing "time-triggered co-operative" (TTC) architectures aimed at highly predictable real-time systems. It only makes sense to generalize the STC concept if its applicability in more complicated systems is proven. The study detailed in the present paper explores one way in which the STC technique can be adapted to assess the behaviour of time-triggered scheduling algorithms in multi-processor embedded systems. The particular systems considered are those which employ TTC schedulers (for task scheduling in each microcontroller node) and a "shared-clock" (S-C) scheduling protocol (for message scheduling) when system nodes are connected via Controller Area Network (CAN) protocol. In more detail, the paper proposes a set of generic "scheduler test cases" (STCs) for exploring the impact of various time-triggered S-C scheduler implementations when used on particular hardware. The results of the paper demonstrate that the STC concept developed previously is also useful (and can be adopted) when more sophisticated real-time scheduling algorithms are implemented in low-cost embedded systems.*
Keywords *-Scheduler, Time-Triggered, Shared-Clock, Test Cases, Jitter, Node Failure*

I. INTRODUCTION

There are numerous ways in which we can classify the different architectures employed in embedded computer systems. The architecture which forms the focus of this paper is described as "time triggered" (TT); as opposed to "even-triggered" (ET) [1]. The ideal TT behaviour cannot be easily achieved in practice, since a complete knowledge about the system behaviour (at all time instants) is required. However, approximations of this model have been found to be useful in many practical systems. This model involves a collection of periodic tasks which operate co-operatively (or "non-pre-emptively"). Such a design is referred to as "time-triggered co-operative" (TTC) architecture [2]–[6]. This architecture has sometimes been described as a "cyclic executive" (e.g. [7]–[9]). Unlike time-triggered pre-emptive algorithms – such as "rate monotonic" (RM), for example – systems with TTC architectures have highly predictable timing behaviour (manifested by the very low levels of task jitter)[6], [8]. Moreover, such systems can maintain their low-jitter characteristics even when sophisticated techniques such as "dynamic voltage scaling" (DVS) are employed to reduce system power consumption [10].

Previous work in the area of TT designs has considered the development of both single- and multi-processor systems. For single-processor ones, extensive studies have been carried out in which they explored practical ways to improve the performance of the TT-based systems (especially those which employ TTC architectures (e.g. see [10]–[14]). In the case of multi-processor designs, it has been demonstrated that a "shared-clock" (S-C) message scheduling protocol – used in conjunction with TTC task scheduling algorithm – can provide a simple, flexible and predictable platform for many practical systems [2]. In such distributed designs, the Controller Area Network (CAN) protocol [15] provides high reliability communications at low cost [16]. Many researchers argue that control systems which require a high degree of determinism should be based on modern protocols other than CAN (which is considered an old-fashioned solution). Indeed, where costs are not an issue, or the bandwidth of the CAN network is insufficient, this may be an appropriate solution. However, experience gained with CAN over recent decades means that it is now possible to create extremely reliable networks using this protocol, if care is taken at the design and implementation stages (see [17], [18]).

Please note that CAN is a deep-rooted standard protocol which has been widely used in automotive and other industrial arenas, and as a consequence of its popularity, most modern microcontroller families have members with on-chip hardware support for this protocol. Since CAN is usually viewed as "event-triggered" protocol [19], the use of a S-C architecture along with CAN hardware helps to achieve a TT network operation [2]. For consistency with our previous studies, we will refer to this system as "TTC-SCC" algorithm[20].

The first two TTC-SCC protocols were introduced by Michael Pont in 2001 [2]. Later in 2007 [21], Devaraj Ayavoo developed two other TTC-SCC protocols and carried out a detailed comparison between all four versions of the S-C protocol. In our study published in 2012 [22], we developed a new implementation of the TTC-SCC protocol and show its advantages over the four previously-developed implementations. In [23], we developed a set of mathematical formulas for estimating (calculating) the message latencies between all communicating nodes in all five TTC-SCC scheduler implementations.

While it is generally accepted that there is a 'one-to-many' mapping between scheduling algorithms and scheduler implementations ([7], [24]–[27]), the process of translating between these two system representations has not been widely discussed (see [6], [20], [26]). Some researchers argued that that there is a wide gap between scheduling theory and its implementation in operating systems running on specific hardware platforms, and that such a gap must be bridged to achieve a meaningful validation of real-time applications([24], [28], [29]). To begin to address this issue, we have recently introduced an empirical approach which we referred to as a "scheduler test case" (STC). Such a technique was intended to be a practical means for assessing and comparing the behaviour of a wide range of representative TTC scheduler implementations employed in single-processor embedded designs [26]. The aim with the designed "scheduler test cases" (STCs) was to facilitate empirical 'black-box' comparisons of the different scheduler implementations so that the behaviour of these implementations can be assessed without the need to access (or fully understand) the underlying source code. As clearly pointed out, such a method allowed those implementing the system to gain a better understanding (during system construction, testing or maintenance) of the way in which a given TTC implementation can be expected to behave under a range of both normal and abnormal operating conditions. In the current study, we extend the STC technique developed in [26]to allow an evaluation of more complicated designs based on distributed embedded architectures. The study outlined here proposes an appropriate set of STCs to help assessing and hence distinguishing the behaviour of a representative implementation classes of the TTC-SCC scheduling protocol (as described in [22] and [23]).

It is worth noting that the main focus of this paper is not on the implementation of a TTC scheduler in the individual nodes (which has already been addressed in the previous paper). Instead, we will focus on the S-C scheduling protocol when implemented along with TTC algorithm in order to manage the transmission of data messages between the different nodes in the system and explore the impact of its various implementations. Since such message transmission processes can be described theoretically using mathematical models, the output results from most of the STCs introduced here will have the form of mathematical equations. Please note that as the complexity of the scheduling algorithm – under test – increases, the use of theoretical (as well as empirical) results can help to provide wider information about the real-time behaviour of the system.

Predictability is a key concern in the development process of time-triggered embedded systems. Therefore, it is used here as the main criterion against which each scheduler behaviour is weighed up. To be able to express predictability using quantitative measures, the following three criteria are considered: 1) transmission jitter; 2) message latencies in the communicating nodes; and 3) the ability of the scheduler to detect an unplanned error and begin to handle it (i.e. estimating the node-failure detection time by the network Master). In addition, memory overheads and network utilisation resulted from the implementation of each scheduler are used to allow a practical comparison between the various schedulers being investigated.

The remainder of the paper is organised as follows. Section II provides an overview of the STC concept we previously developed. In Section III, we propose an appropriate set of STCs to use for multi-processor TTC-SCC scheduling protocol. Section IV outlines the methodology used to obtain the results presented in this paper. In Sections V, we provide results from the employment of the STCs in all five implementation models of the TTC-SCC scheduling protocol. In Section VI, a case study is presented in which we perform a quantitative comparison between the compared schedulers in terms of their communication behaviour. Finally, we draw the paper conclusions in Section VII.

II. SCHEDULER TEST CASE (STC) TECHNIQUE

Unlike conventional testing approaches, the STC technique developed in [26] did not aim to check the correct functionality of the application software or evaluate its quality attributes. Instead, it was mainly developed to assess the actual behaviour of the system as a result of employing a particular software implementation of a scheduling algorithm on commercial-off-the-shelf microcontroller hardware. In another word, the technique provides a practical way to ensure that timing predictions made at early stages in the

development process (e.g. at the design stage) are maintained during and after the system implementation process.

Since STC is a testing technique, an appropriate set of test cases must be designed which specify the system inputs, predicted results, and execution conditions. Also, only a selective subset of possible test cases can be used (since comprehensive testing is not viable). As with all testing methods, the feature of the system to be tested must be selected along with the inputs that will execute that feature, and the expected outputs of the test cases must be known in advance. All of these test case elements were considered in the process of developing the STCs applied in [26] and in the present paper.

Moreover, the STC technique requires that the system behaviour must be tested under both normal and abnormal operating conditions. Normal operations refer to the situations where the scheduler operates in an absence of errors, while abnormal operations relate to the occurrence of errors. The error mode in any scheduling algorithm – for which the STCs are developed – must be defined by the developer and it should reflect a commonly encountered problem facing the developers of such an algorithm. For example, in TTC systems, "task overrun" is a major problem which may cause measurable degradation in the system performance or jeopardise the system functionality. Therefore, task overrun was used in our previous study to define the error mode of the TTC scheduler. The same concepts are applied for multi-processor scheduling systems. This is further discussed in the next section.

III. DESIGN OF SCHEDULER TEST CASES (STCS) FOR MULTI-PROCESSOR TT EMBEDDED SYSTEMS

Due to the relative complexity of multi-processor embedded designs, developing test cases for such systems cannot be seen as a trivial process. We therefore suggest that – along with empirical tests – each scheduler implementation will also be tested using test cases that allow evaluating the behaviour of the system by mathematical models. This is further discussed shortly.

In this section, we present the various scheduler test cases used in this study to verify the different implementation options of the TTC-SCC schedulers. In all test cases, we assume that we have one Master and three Slaves connected via CAN fieldbus. Each node executes its tasks using TTC scheduling algorithm, however, only the Master node is driven by periodic interrupts generated from its on-chip timer. The Master Tick message is then used to drive the local time of each of the three Slaves whose timer interrupts are totally disabled. Such a process is illustrated in Fig.1(see [23] for further information about the TTC-SCC scheduler operation).

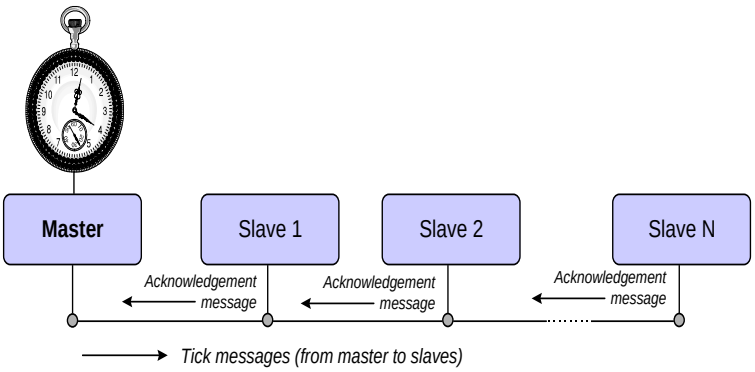

Fig.1: Simple architecture of time-triggered shared-clock scheduler.

3.1. STC A: Jitter Test

STC A is developed to assess the jitter levels in the relative timing of Master and Slave ticks in a TTC-SCC network. Such a transmission jitter is very important and can have a significant impact on the overall predictability of systems employing TTC-SCC protocol. Further details about the sources and consequences of such a jitter, and potential solutions to reduce its magnitude in practical designs are discussed in [4], [17]. Note that the results from STC A are all empirical.

In STC A, the system has one task (`Master_Task_A`) running on the Master node and a corresponding task (`Slave1_Task_A`) running on one Slave node. Given that "`Master_Task_A`" sends random data to "`Slave1_Task_A`" every time it is called, jitter test assesses the variation in the time delay between these two communicating tasks. Please recall that all other Slaves will receive Master data at the same instant over the CAN bus[2]. Also note that the overheads of the Master and the Slave task schedulers do not

introduce any jitter [30] and, hence, the jitter observed is only caused by the communication protocol due to bit-stuffing mechanism [31].

3.2. STC B: Master-to-Slave Message Latency

STC B is developed to assess the communication latency between the Master node and any Slave node in the network. Since such a message latency can be described mathematically, the output results from this STC are in the form of mathematical equations.

In this test case, the message latency between the Master and Slaves in all TTC-SCC protocols is calculated. The STC B evaluates the best-case (minimum) and the worst-case (maximum) message transmission times between the Master and the Slave node.

3.3. STC C: Slave-to-Master Message Latency

STC C is developed to assess the communication latency between any Slave node and the Master node in the network. Results from this STC are also in the form of mathematical equations.

In this test case, the message latency between any Slave and the Master in all TTC-SCC protocols is calculated. The STC C evaluates the best-case (minimum) and the worst-case (maximum) message transmission times between the Slave and the Master node.

3.4. STC D: Slave-to-Slave Message Latency

STC D is developed to assess the communication latency between Slave 'X' and Slave 'Y' in the network. Results from this STC are also in the form of mathematical equations.

In this test case, the message latency between any two Slaves in all TTC-SCC protocols is calculated. The STC D evaluates the best-case (minimum) and the worst-case (maximum) message transmission times between Slave 'X' and Slave 'Y'.

3.5. STC E: Node-Failure Detection Time (NFDT)

As discussed in [26], the STC technique tests the system under both normal and abnormal operating conditions. Having considered that STCs B, C and D assess the behaviour of the TTC-SCCprotocol under normal conditions, STC E is developed to assess the system behaviour when an error takes place. In communication networks, node failure is a common error that can degrade the overall predictability of the system. The node failure describes a situation where one or more nodes do not respond to messages sent from other nodes due to hardware / software error occurred in the receiving node.

The S-C protocol applies several error detection and recovery mechanisms. For example, the Master can easily detect an error on any Slave if no "Ack" message is received from a particular Slave within its sending time interval. Once an error is detected in the S-C network, appropriate handling mechanism(s) can be employed. For example, when a Master detects a failure in one of the Slaves, it can have three options: 1) enter a safe state then shut down the whole network; 2) reset the network; or 3) start a backup Slave (see [2] for further details).The STC E assesses the behaviour of a TTC-SCC protocol when one of the Slaves becomes temporarily out of order. This test case evaluates the worst-case time taken by the network-Master to detect the failure and hence begin to handle it. Results from this STC are also in the form of mathematical equations.

3.6. Memory and Network Utilisation (NU) requirements

Memory requirements are also reported here as a means for distinguishing the various TTC-SCC schedulers. Moreover, in communication network, the message utilisation of the available network bandwidth is a major concern that affects the network efficiency. Therefore, the message bandwidth utilisation in each TTC-SCC scheduling protocol is also estimated.

IV. METHODOLOGY

The methodology used to obtain both the experimental and theoretical results is described in this section.

4.1. Representative Examples of TTC-SCC Implementations

In this study, all five TTC-SCC scheduling protocols developed previously have been considered; i.e. TTC-SCC1 and TTC-SCC2 [2], TTC-SCC3 and TTC-SCC4 [21], and TTC-SCC5 [22]. Such protocols can be viewed as a representative set of the wide range of possible implementation options of the S-C scheduling algorithm for which the test cases are developed.

4.2. Hardware and Software Setup

The empirical measurements in this study (i.e. jitter) were conducted using Phytec boards supporting Infineon C167 microcontrollers. The C167 is a 16-bit microcontroller with a 20 MHz crystal oscillator. The C167 board has additional on-chip support for CAN protocol.

The network nodes (one Master and three Slaves) were connected using a twisted-pair CAN link. The CAN baudrate used was 1 Mbit/sec, and 8-byte "Tick" messages were transmitted, with one byte reserved for the Slave ID, while the remaining data bytes contained random values. The tick interval used was 4 ms.

Note that in [2], a complete set of codes required to implement the TTC-SCC protocol on an 8-bit 8051 microprocessor hardware was provided. For the 16-bit system considered here, the 8051 design was ported to the C16x family. Finally, the codes were compiled using the Keil C166 compiler.

4.3. JitterMeasurements

To make transmission delay measurements, a pin on the Master node was set high (for a short period) at the start of the "`Master_Task A`". Another pin on the Slave (initially high) was set low at the start of the "`Slave1_Task_A`". The signals obtained from these two pins were then AND-ed (using a 74LS08N chip from Texas Instruments) to give a pulse stream with widths that represent the transmission delays. These widths were measured using a National Instruments data acquisition card 'NI PCI-6035E' [32], used in conjunction with appropriate software LabVIEW 7 [33].

The first set of timing results includes: maximum, minimum and average message transmission times. In addition, average jitter and difference jitter were recorded. The difference jitter is obtained by subtracting the best-case (minimum) transmission time from the worst-case (maximum) transmission time from the measurements in the sample set (this jitter is referred to by other authors as absolute jitter: see [34]). The average jitter is represented by the standard deviation in the measure of average message transmission time. Note that there are many other measures that can be used to represent the levels of task jitter, but these measures were felt to be appropriate for this study.

4.4. Message Latency Calculations

Here, we will consider the results presented in our previous publication (i.e. [23]). In summary, the message transmission delays (latencies) were calculated between the start of the tick in which data is generated and the tick in which data is received. For further details, refer to [23].

4.5. Node-Failure Detection Time Calculation

To assess the behaviour of the scheduler in the event of node-failure error, it has been decided to calculate the worst-case time the Master processor would take to detect the failure and begin to react to it (see STC E).

To obtain the worst-case scenario, it is assumed that the Slave fails immediately after it has talked (i.e. sent its Ack message) to the Master. The worst-case node-failure detection time will hence be calculated between this failure time and the start of the tick in which the Master checks the status of this Slave.

4.6. Network Utilisation Tests

Network utilisation in each protocol is also reported in this study. The network utilisation values are represented mathematically as functions of: 1) lengths of the various messages exchanged in the network; and 2) scheduler tick interval. Assuming 1 Mbit/s CAN speed, the message transmission time will be equal to the number of message bits. The network utilisation in each scheduler implementation is presented as the average 'time' bandwidth per tick interval.

4.7. Memory Test

To reflect the scheduler complexity, the CODE (ROM) and DATA (RAM) memory values required to implement each of the compared scheduling protocols are recorded. These values are obtained directly from the ".map" file created when the source code is compiled on the Keil simulator.

V. RESULTS

In this section, the results from the implementation of the STCs in the various TTC-SCC scheduling protocols are presented.

5.1. STC A: Jitter

Table ***1*** shows the results obtained from the STC A (jitter test) implemented with all TTC-SCC scheduling protocols.

Table 1. Task jitter from all TTC-SCC schedulers (all values in μs).

Scheduler	TTC-SCC1	TTC-SCC2	TTC-SCC3	TTC-SCC4	TTC-SCC5
Min transmission time	162.9	163	162.9	99.9	100
Max transmission time	173	173.1	172.9	102	102.2
Average transmission time	166.3	166	166.2	101	101.1
Diff. Jitter	10.1	10.1	10	2.1	2.2
Avg. Jitter	1.5	1.4	1.5	0.6	0.6

It is clear from the results that in TTC-SCC4 and TTC-SCC5 – where Tick messages transmitted from the Master had fixed lengths – the difference jitter was reduced by approximately 80% when compared to the TTC-SCC1, TTC-SCC2 and TTC-SCC3 schedulers. Recall that jitter is a key factor which indicates (manifests) the predictability level of a system.

5.2. STC B, STC C and STC D: Message Latencies

This section presents the results obtained from the STC B, STC C and STC D implemented with all TTC-SCC scheduling protocols. To express the message latencies between any two communicating nodes in the network, the following parameters are defined:

- M: is the Master Tick message length in TTC-SCC1 – TTC-SCC3 schedulers.
- T: is the length of the tick interval.
- TDMA1 – TDMA5: are the Time Division Multiple Access rounds for TTC-SCC1 – TTC-SCC5 schedulers (respectively).
- N: is the number of Slaves in the network.
- D_{XX}: is the distance between successive ticks allocated for a given Slave.
- D_{XY}: is the shortest distance between Ack messages from any two communicating Slaves.
- $D_{Xi\ Yi}$: is the "current" distance between the Ack message of the sending Slave and the Ack message of the receiving Slave.
- $D_{X(i+1)\ Y(i+1)}$: is the "next" distance between the Ack message of the sending Slave and the Ack message of the receiving Slave.
- m: is the number of Slaves replying per tick interval in TTC-SCC3, TTC-SCC4 and TTC-SCC5.
- k: is the total number of ticks in the TDMA round.
- M_T is the Master Tick message length in TTC-SCC4 and TTC-SCC5 schedulers.
- M_D is the Master Data message length in TTC-SCC4 and TTC-SCC5 schedulers.

For a detailed set of graphs showing the communication processes between different nodes and for the derivation of the formulasoutlined below, please refer to [23].

5.2.1. STC B: Master-to-Slave Message Latency

Table ***2*** presents the equations for the best- and the worst-case message latencies between the Master and a given Slave in all TTC-SCC schedulers.

As discussed in [23], TDMA rounds in all schedulers are calculated as follows:

$$TDMA1 = NT \quad (1)$$

$$TDMA2 = (2N - 2)T \quad (2)$$

$$TDMA3 = \frac{TDMA1}{m} = \frac{NT}{m} \quad (3)$$

$$TDMA4 = \frac{(N+1)T}{m} \quad (4)$$

$$TDMA5 = TDMA3 = \frac{TDMA1}{m} = \frac{NT}{m} \quad (5)$$

Table 2: Master-to-Slave latency equations for all TTC-SCC schedulers.

Scheduler	Best-case latency	Worst-case latency
TTC-SCC1	$T + M$	$T + M$
TTC-SCC2	$T + M$	$T + M$
TTC-SCC3	$T + M$	$T + M$
TTC-SCC4	$T + M$	$T + M$

TTC-SCC5	$2T + M_T$	$2T + M_T$

5.2.2. STC C: Slave-to-Master Message Latency

Table *3* presents the equations for the best- and the worst-case message latencies between a given Slave and the Master in all TTC-SCC schedulers.

Table 3: Slave-to-Master latency equations for all TTC-SCC schedulers.

Scheduler	Best-case latency	Worst-case latency
TTC-SCC1	$2T - M$	$TDMA1 + T - M$
TTC-SCC2	$2T - M$	$D_{XX} + T - M$
TTC-SCC3	$2T - M$	$TDMA3 + T - M$
TTC-SCC4	$2T - M$	$TDMA3 + T - M$
TTC-SCC5	$2T - M_T$	$TDMA5 + T - M_T$

5.2.3. STC D: Slave-to-Slave Message Latency

Table ***4*** presents the equations for the best- and the worst-case message latencies between any two Slaves in all TTC-SCC schedulers.

Table 4: Slave-to-Slave latency equations for all TTC-SCC schedulers.

Scheduler	Best-case latency		Worst-case latency	
TTC-SCC1	For $D_{XY} > T$:	$D_{XY} + T$	For $D_{XY} > T$:	$D_{XY} + TDMA1$
	For $D_{XY} = T$:	$2T + TDMA1$	For $D_{XY} = T$:	$T + 2\ TDMA1$
TTC-SCC2	For $D_{Xi\ Yi} > T$:	$D_{Xi\ Yi} + T$	For $D_{X(i+1)\ Y(i+1)} > T$:	$D_{Xi\ Y(i+1)}$
	For $D_{Xi\ Yi} = T$:	$D_{Xi\ Y(i+1)} + T$	For $D_{X(i+1)\ Y(i+1)} = T$:	$D_{Xi\ Y(i+2)}$
TTC-SCC3	$2T$		$TDMA3 + T$	
TTC-SCC4	$2T$		$TDMA3 + T$	
TTC-SCC5	$2T$		$TDMA5 + T$	

5.3. STCE: Node-Failure Detection Time (NFDT)

This section presents the results obtained from the STC E implemented with all TTC-SCC schedulers. Derivation of all formulas is provided in this study.

5.3.1. Nfdt For Ttc-Scc1

In TTC-SCC1, the Master node has to wait for a complete TDMA round before the status of all Slaves can bechecked. UsingFig.2,the worst-case failure detection time for the TTC-SCC1 scheduler is calculated as:

$$NFDT1 = TDMA1 + T - M = (N+1)\ T - M \quad (6)$$

In the example shown in the figure, the Master would take four Tick intervals (i.e. TDMA plus one additional tick) to detect a failure on S1. The situation would become worse ifthe number of Slave nodes *N* increases.

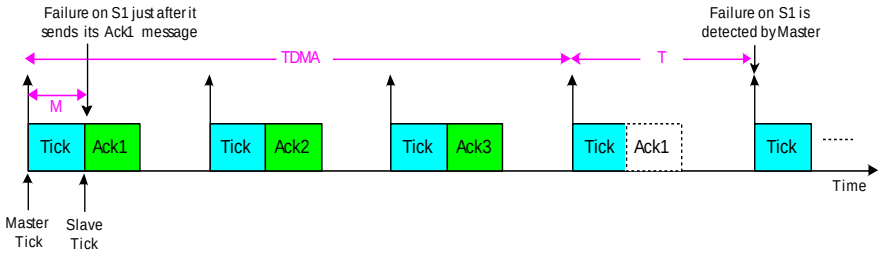

Fig.2: Failure detection time in TTC-SCC1.

5.3.2. Nfdt For Ttc-Scc2

In TTC-SCC2, the Master node has to wait until the status of the Slave is next checked. Using Fig.3, the worst-case failure detection time for the TTC-SCC2 scheduler is calculated as:

$$NFDT2 = D_{XX} + T - M \quad (7)$$

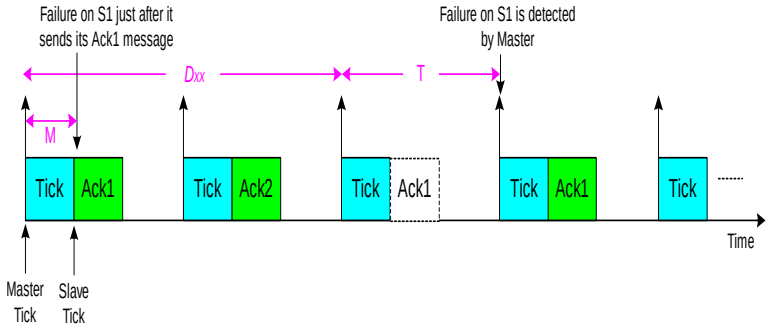

Fig.3: Failure detection time in TTC-SCC2.

In the example shown in the figure, the Master would take approximately three Tick intervals to detect a failure on S1. Failure detection time for a given node in TTC-SCC2 scheduler would depend on the number of Slaves in the network, length of the TDMA round, and the number of ticks – within the TDMA round – used to communicate with that Slave. In some cases, where (for example) *TDMA*2 is very long and the Slave is only checked once per TDMA round, detecting failure in such a Slave can be too long. This can have an adverse impact on the predictability of many networks.

5.3.3. Nfdt For Ttc-Scc3

The TTC-SCC3 allows the Master node to quickly receive Ack messages from the Slaves. For example, Fig.4 illustrates an example where S1 suffers a failure as soon as it has sent its Ack message. It is assumed here that the TDMA round is extended across two tick intervals. As a result, the longest possible time for the Master node to detect a failure on S1 node is calculated as follows:

$$NFDT3 = TDMA3 + T - M = [(N / m) + 1]\ T - M \quad (8)$$

Remember that TDMA here equals to NT / m. When all Slaves are allowed to reply in one tick (i.e. $N = m$), then the worst-case failure detection time becomes equal to $2T - M$.This duration is slightly less than two Tick intervals (which is significantly less than the corresponding time in TTC-SCC1 and TTC-SCC2 for non-trivial networks).

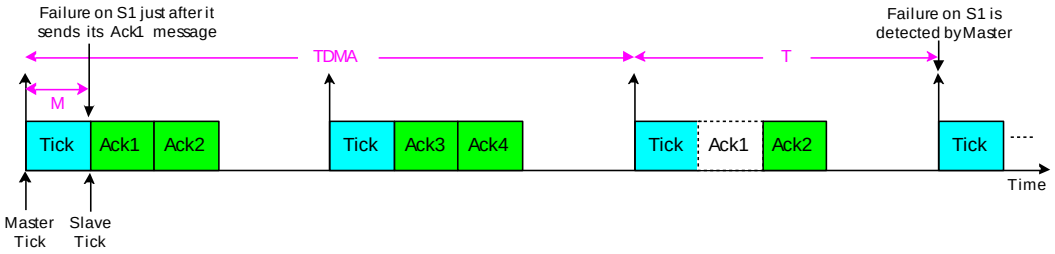

Fig.4: Failure detection time in TTC-SCC3.

5.3.4. Nfdt For Ttc-Scc4

The results here are very similar to those obtained from the TTC-SCC3 scheduler. The only difference is that the Tick message here is extremely short, therefore the worst-case failure detection time for S1 in the example shown inFig.4 is calculated as follows:

$$NFDT4 = TDMA4 + T - M_T = [(N + 1) / (m + 1)]\ T - M_T \quad (9)$$

Recall that N is the original number of Slaves and M_T is the Master Tick message length: this is in order to distinguish it from the ordinary Tick message which contains data in its data field.

5.3.5. Nfdt For Ttc-Scc5

Fig.5 illustrates an example where S1 suffers a failure as soon as it has sent its Ack message. If the TDMA round is extended across two tick intervals, the longest possible time that the Master node takes to detect a failure on the Slave node is calculated as follows:

$$NFDT5 = TDMA5 + T - M_T - M_D = [(N/m) + 1]\ T - M_T - M_D \quad (10)$$

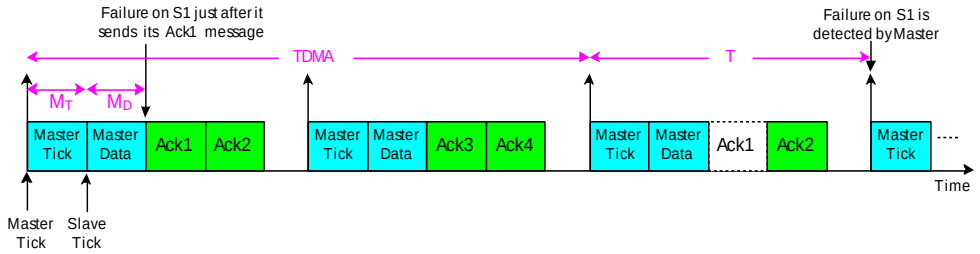

Fig.5: Failure detection time in TTC-SCC5.

As in the TTC-SCC3 scheduler, when all Slaves areallowed to reply in one tick (i.e. $N = m$), then the worst-case failure detection time becomes equal to $2T - M_T - M_D$.

5.4. Network Utilisation (NU)

This section presents mathematical formulas for calculating the network utilisation in all TTC-SCC schedulers. Recall that here we consider the utilisation of the network time-bandwidth.

5.4.1. NU For TTC-SCC1

Assume that all Ack messages are equal in length, and each one is represented by S (due to Slave sending this type of message), then the network utilisation in TTC-SCC1 can be calculated as follows:

$$NU1 = \frac{NM + NS}{TDMA1} = \frac{NM + NS}{NT} = \frac{M + S}{T} \quad (11)$$

Remember that the number of Tick messages in each TDMA round isequal to the number of Slaves. If the length of the Tick message is assumed equal to the length of Ack message, then (11) can be simplified as:

$$NU1 = \frac{2M}{T} \quad (12)$$

5.4.2. NU For TTC-SCC2

Assume that all Ack messages are equal and, where any Ack message is represented by S, k is the total number of ticks in the TDMA round, M is the Master Tick message length, then the network utilisation in TTC-SCC2 can be calculated as follows:

$$NU2 = \frac{k(M + S)}{kT} = \frac{M + S}{T} \quad (13)$$

If the length of the Tick message is assumed equal to the length of Ack message, then (13) can be simplified as:

$$NU2 = \frac{2M}{T} \quad (14)$$

5.4.3. NU for TTC-SCC3

Again, assume that S is the length of any Ack message, then the network utilisation in TTC-SCC3 can be calculated as follows:

$$NU3 = \frac{\left(\frac{N}{m}\right)M + NS}{TDMA3} = \frac{\left(\frac{N}{m}\right)M + NS}{\left(\frac{NT}{m}\right)} = \frac{M + mS}{T} \quad (15)$$

Remember that the number of Tick messages in each TDMA round is equal to the number of tick intervals (which is equal to N / m). If the length of the Tick message is assumed equal to the length of Ack message, then (15) can be simplified as:

$$NU3 = \frac{(1+m)M}{T} \quad (16)$$

5.4.4. NU for TTC-SCC4

Again, assume that S is the length of any Ack message, and M_T is the length of the Master Tick message, then the network utilisation in TTC-SCC4 can be calculated as follows:

$$NU4 = \frac{\left(\frac{N+1}{m}\right)M_T + (N+1)S}{\left(\frac{N+1}{m}\right)T} = \frac{M_T + mS}{T} \quad (17)$$

Remember that the Tick message is sent from a dedicated Master node, and the number of Slaves has increased by one: this is where the term (N+1) comes from. The length of the Master Tick message is assumed shorter than the length of Ack message since it contains no data, so S cannot be substituted by M_T in the equation.

5.4.5. NU for TTC-SCC5

Again, assume that S is the length of any Ack message, then the network utilisation in TTC-SCC5 can be calculated as follows:

$$NU5 = \frac{\left(\frac{N}{m}\right)M_T + \left(\frac{N}{m}\right)M_D + NS}{\left(\frac{NT}{m}\right)} = \frac{M_T + M_D + mS}{T} \quad (18)$$

Remember that in each tick, the Master sends Tick message and Data message. The Master Tick message is assumed shorter that the Data message, since it contains no data. If the length of the Master Data message is assumed equal to the length of Ack message, then (18) can be simplified as:

$$NU5 = \frac{M_T + (1+m)M_D}{T} \quad (19)$$

5.5. Memory Requirements

Table ***5***presents the memory overheads resulted from the implementation of all TTC-SCC scheduling protocols on the used C166 microcontroller hardware.

Table 5: Memory overheads for all TTC-SCC schedulers (all values are in Byte).

Scheduler	Master		Slave	
	ROM	RAM	ROM	RAM
TTC-SCC1	1666	30	1590	108
TTC-SCC2	1710	31	1590	108
TTC-SCC3	1838	33	1722	116
TTC-SCC4	1768	32	1722	116
TTC-SCC5	1884	34	1760	118

The table clearly shows that all Slaves required the same memory overheads in TTC-SCC1 and TTC-SCC2, and in TTC-SCC3 and TTC-SCC4. This is because the Slave codes are identical in these cases. In the Master, it can be seen that the memory overheads increased as the scheduler incorporated more features. For example, TTC-SCC5 scheduler required the largest amount of memory overheads to be implemented on the used hardware platform. However, such increases in memory requirements can still be seen very small (i.e. approximately 12% in the ROM and RAM as compared to the basic TTC-SCC1 scheduler).

VI. CASE STUDY: PRACTICAL COMPARISON BETWEEN TTC-SCC SCHEDULERS

In this section, we seek to provide a practical comparison between the various schedulers considered here using a small case study. This is basically to allow the reader understand how the STC technique helps to provide a meaningful comparison between the behaviour of the various TTC-SCC schedulers when operated at real-time. Thus, we will use the same system described in [23]. Briefly, three Slave nodes are connected up in the network, CAN baudrate is 1 Mbit/s and the tick interval is 4 ms. Assuming “standard” CAN messages (i.e. 11-bit identifier), “Tick” and “Ack” messages send seven “random” data bytes along with the Slave / Group ID byte (except in the Tick-only message which has no data), then the value of M, M_D and S are equal to 135 µs (with the worst-case level of bit-stuffing) and the value for M_T is equal to 47 µs (without data bytes and any bit-stuffing). The configurations of the TTC-SCC schedulers used with this small network are presented inTable 6.

The results obtained from this case study are summarised inTable 7. Again, as in [23], the following abbreviations are used: M-S (Master-to-Slave), S-M (Slave-to-Master), S-S (Slave-to-Slave), NFDT (Node-

failure detection time), NU (Network utilisation), BC (Best-case) and WC (Worst-case).

Table 6: TTC-SCC models used in the case study to allow a comparison between schedulers.

Scheduler	TDMA (Ticks)	TDMA (µs)	Comments
TTC-SCC1	3	12	Three Slaves, where each Slave sends its Ack once in its allocated tick.
TTC-SCC2	4	16	S1 is allocated two ticks to send its Ack message, while S2 and S3 only send their Ack once.
TTC-SCC3	1	4	Here, *m* = 3. All Slaves send their Ack in the same tick
TTC-SCC4	1	4	Here, *m* = 4. The number of Slaves increased by one. Tick message is very short compared to Slaves Ack messages.
TTC-SCC5	1	4	Here, *m* = 3. Tick message is also very short compared to Master Data and Slaves Ack messages.

Table 7: Results from the case study used to compare between TTC-SCC schedulers.

Scheduler	M-S1 Latencies (µs)		S1-M Latencies (µs)		S1-S2 Latencies (µs)		NFDT (S1) (µs)	NU (%)
	BC	WC	BC	WC	BC	WC		
TTC-SCC1	4.135	12.135	7.865	15.865	20	28	15.865	6.75%
TTC-SCC2	4.135	8.135	7.865	11.865	20	24	11.865	6.75%
TTC-SCC3	4.135	4.135	7.865	7.865	8	8	7.865	13.5%
TTC-SCC4	8	8	8	8	8	8	7.953	14.675%
TTC-SCC5	8.047	8.047	7.953	7.953	8	8	7.818	14.675%

The results for message latencies have already been discussed thoroughly in [23]. One key observation is that the TTC-SCC1 results in comparatively long delays especially when worse-case scenarios are considered.However, it maintains high network efficiency (the network utilisation is less than 7%). The use of TTC-SCC2 helps to reduce the worst-case message latencies without compromising the network efficiency.

Moreover, the TTC-SCC3 scheduler provides shorter Master-to-Slave latencies (by 50%) and slightly less network utilisation when compared to the TTC-SCC4 and TTC-SCC5.This of course does not imply that TTC-SCC3 is superior to TTC-SCC4 and TTC-SCC5 since jitter levels induced in TTC-SCC3 are quite high (see

Table ***1*** above). In the same way, one can compare any two schedulers against a chosen criterion based on the results obtained here.Further criteria can also be considered for meaningful comparisons.

VII. CONCLUSIONS

The aim of this paper was to explore the applicability of the STC technique developed previously for single-processor architectures when implementing more complicateddesigns (e.g. systems which are based on multi-processor architectures). The paper proposed a set of scheduler test cases (abbreviated as STCs) to help evaluate the behaviour of multi-processor embedded systems when a representative set of appropriate time-triggered scheduler implementations are employed. The discussions emphasised that the main focus was to assess the predictability behaviour of CAN-based TT systems implemented using TTC-SCC scheduling protocols. The criteria considered in such an evaluation process included: the levels of jitter in the relative timing of Master and Slave ticks, message latencies between any two communicating nodes, node-failure detection time, and practical resource requirements (i.e. memory overheads and network utilisation).

The methodology used to obtain the results from each TTC-SCC implementation when the STCs are applied was outlined. The results suggested that there is no prefectimplementation which can fit all applications. However, according to the features concerned with in this study for multi-processor embedded designs, it can be concluded that the TTC-SCC5 scheduler can be an attractive solution for a wide range of applications due to its low-jitter characteristics, short message latencies and short node-failure detection time along with low resource requirements. Of course, in other occasions, the user may decide to implement any of the four other schedulers (or a combination of them) according to their system-specific requirements.

Overall, the results presented in this paper has practically demonstrated that the use of STC concept is not limited to simple software architectures. Instead, it can easily be adapted to evaluate the implementations of scheduling systems with more complex software architectures such as, and not limited to, the S-C scheduling protocols considered in this study. However, the complexity of the test case design process for a particular system would increase as the system's complexity increases.

VIII. ACKNOWLEDGEMENTS

The work presented in this paper was carried out in the Embedded Systems Laboratory (ESL) at University of Leicester, UK, under the supervision of Professor Michael Pont, to whom authors are thankful.

IX. REFERENCES

[1]. H. Kopetz, Real-Time Systems. Boston, MA: Springer US, 2011.
[2]. M. J. Pont, Patterns for time-triggered embedded systems: building reliable applications with the 8051 family of microcontrollers. Harlow: Addison-Wesley, 2001.
[3]. M. Short, "Analysis and redesign of the 'TTC' and 'TTH' schedulers," Journal of Systems Architecture, vol. 58, no. 1, pp. 38–47, Jan. 2012.
[4]. M. Nahas, M. J. Pont, and M. Short, "Reducing message-length variations in resource-constrained embedded systems implemented using the Controller Area Network (CAN) protocol," Journal of Systems Architecture, vol. 55, no. 5–6, pp. 344–354, May 2009.
[5]. M. A. Hanif, "Design and evaluation of flexible time-triggered task schedulers for dynamic control applications," Thesis, University of Leicester, 2013.
[6]. M. Nahas and A. M. Nahhas, "Ways for Implementing Highly-Predictable Embedded Systems Using Time-Triggered Co-Operative (TTC) Architectures," in Embedded Systems - Theory and Design Methodology, K. Tanaka, Ed. InTech, 2012.
[7]. T. P. Baker and A. Shaw, "The cyclic executive model and Ada," Real-Time Syst, vol. 1, no. 1, pp. 7–25, Jun. 1989.
[8]. C. D. Locke, "Software architecture for hard real-time applications: Cyclic executives vs. fixed priority executives," The Journal of Real-Time Systems, vol. 4, no. 1, pp. 37–53, Mar. 1992.
[9]. G. Langelier, A. Dury, A. Petrenko, S. Ramesh, and T. Assaf, "Building an interactive test development environment for cyclic executive systems," in Industrial Embedded Systems (SIES), 2015 10th IEEE International Symposium on, 2015, pp. 1–9.
[10]. T. Phatrapornnant and M. J. Pont, "Reducing jitter in embedded systems employing a time-triggered software architecture and dynamic voltage scaling," IEEE Transactions on Computers, vol. 55, no. 2, pp. 113–124, Feb. 2006.
[11]. M. Nahas, "Employing Two 'Sandwich Delay' Mechanisms to Enhance Predictability of Embedded Systems Which Use Time-Triggered Co-Operative Architectures," Journal of Software Engineering and Applications, vol. 4, no. 7, pp. 417–425, 2011.
[12]. M. Nahas, "Implementation of highly-predictable time-triggered cooperative scheduler using simple super loop architecture," International Journal of Electrical & Computer Sciences, vol. 11, pp. 33–38, 2011.
[13]. Z. M. Hughes and M. J. Pont, "Reducing the impact of task overruns in resource-constrained embedded systems in which a time-triggered software architecture is employed," Transactions of the Institute of Measurement and Control, vol. 30, no. 5, pp. 427–450, Dec. 2008.
[14]. M. Nahas and R. Bautista-Quintero, "Implementing adaptive time-triggered co-operative scheduling framework for highly-predictable embedded systems," American Journal of Embedded Systems and Applications, vol. 2, no. 4, pp. 38–50, 2014.
[15]. Bosch, CAN Specification Version 2.0. Bosch, 1991.
[16]. M. Farsi and M. B. M. Barbosa, CANopen implementation: applications to industrial networks. Baldock, Hertfordshire, England; Philadelphia, PA: Research Studies Press, 1999.
[17]. M. Nahas, "Applying Eight-to-Eleven Modulation to reduce message-length variations in distributed embedded systems using the Controller Area Network (CAN) protocol," Canadian Journal on Electrical and Electronics Engineering, vol. 2, no. 7, pp. 282–293, 2011.
[18]. M. Short and M. J. Pont, "Fault-Tolerant Time-Triggered Communication Using CAN," IEEE Transactions on Industrial Informatics, vol. 3, no. 2, pp. 131–142, May 2007.
[19]. G. Leen and D. Heffernan, "TTCAN: a new time-triggered controller area network," Microprocessors and Microsystems, vol. 26, no. 2, pp. 77–94, 2002.
[20]. M. Nahas, "Studying the Impact of Scheduler Implementation on Task Jitter in Real-Time Resource-Constrained Embedded Systems," Journal of Embedded Systems, vol. 2, no. 3, pp. 39–52, 2014.
[21]. D. Ayavoo, M. J. Pont, M. Short, and S. Parker, "Two novel shared-clock scheduling algorithms for use with 'Controller Area Network' and related protocols," Microprocessors and Microsystems, vol. 31, no. 5, pp. 326–334, Aug. 2007.
[22]. M. Nahas, "Developing a Novel Shared-Clock Scheduling Protocol for Highly-Predictable Distributed Real-Time Embedded Systems," American Journal of Intelligent Systems, vol. 2, no. 5, pp. 118–128, Dec. 2012.
[23]. M. Nahas, "Estimating Message Latencies in Time-Triggered Shared-Clock Scheduling Protocols Built on CAN Network," Journal of Embedded Systems, vol. 2, no. 1, pp. 1–10, 2014.
[24]. D. I. Katcher, H. Arakawa, and J. K. Strosnider, "Engineering and analysis of fixed priority schedulers," IEEE Transactions on Software Engineering, vol. 19, no. 9, pp. 920–934, Sep. 1993.
[25]. B. Koch, "The Theory of Task Scheduling in Real-Time Systems: Compilation and Systematization of the Main Results," Studies Thesis, University of Hamburg, 1999.
[26]. M. Nahas and R. Bautista-Quintero, "Developing Scheduler Test Cases to Verify Scheduler Implementations in Time-Triggered Embedded Systems," International Journal of Embedded systems and Applications (IJESA), vol. 6, no. 1/2, pp. 1–20, Jun. 2016.
[27]. M. Nahas, "Bridging the gap between scheduling algorithms and scheduler implementations in time-triggered embedded systems.," Thesis, University of Leicester, 2009.
[28]. R. L. Burdett and E. Kozan, "Techniques to effectively buffer schedules in the face of uncertainties," Computers & Industrial Engineering, vol. 87, pp. 16–29, 2015.
[29]. F. Werner, "Scheduling under uncertainty," unpublished document, 2012.
[30]. M. Nahas, M. J. Pont, and A. Jain, "Reducing task jitter in shared-clock embedded systems using CAN," in Proceedings of the UK Embedded Forum, 2004, pp. 184–194.
[31]. T. Nolte, H. Hansson, and C. Norstrom, "Minimizing CAN response-time jitter by message manipulation," in Eighth IEEE Real-Time and Embedded Technology and Applications Symposium, 2002. Proceedings, 2002, pp. 197–206.
[32]. "Multifunction Devices - National Instruments." [Online]. Available: http://www.ni.com/data-acquisition/multifunction/. [Accessed: 01-Jun-2016].
[33]. "LabVIEW System Design Software - National Instruments." [Online]. Available: http://www.ni.com/labview/. [Accessed: 01-Jun-2016].
[34]. G. C. Buttazzo, Hard real-time computing systems: predictable scheduling algorithms and applications. New York: Springer, 2005.