Spectrum of Engineering Sciences
ISSN (e) 3007-3138 (p) 3007-312X Volume 3, Issue 7, 2025

TASK SYNCHRONIZATION IN WIRELESS DISTRIBUTED NETWORKS USING A SHARED-CLOCK ARCHITECTURE

Muhammad Amir[1], Bilal Ur Rehman[*2], Kifayat Ullah[3], Muhammad Farooq[4], Humayun Shahid[5], Muhammad Iftikhar Khan[6]

[1, *2,3,4,6]*Department of Electrical Engineering, University of Engineering and Technology, Peshawar, KPK, Pakistan.*
[5]*Department of Telecommunication Engineering, University of Engineering & Technology, Taxila, Pakistan*

DOI: https://doi. org/10.5281/zenodo.16568400

Keywords
Shared-Clock, Wireless, Task Synchronization

Article History
Received on 29 April 2025
Accepted on 14 July 2025
Published on 29 July 2025

Corresponding Author: *
Bilal Ur Rehman

Abstract
Safety-critical embedded systems should exhibit robustness and fault tolerance. Shared-clock (S-C) scheduling architectures have been developed to encapsulate the above qualities in resource-constrained distributed systems which employ a time-triggered (TT) architecture. Previous work in this area has targeted 'wired' communication mediums (e. g. local connections based on CAN). In this paper we begin to consider whether S-C architecture can be adapted for use in wireless environments. This work is at an early stage and the main aims of the paper are: (1) to explain the operation of wired S-C designs; (2) to summarize the problems which are faced when attempting to use a S-C algorithm in a wireless environment; (3) to present some initial suggestions for ways in which a S-C algorithm might be adapted for use in a wireless environment; (4) to describe the design of a testbed which will be used to support further research in this area.

INTRODUCTION

Shared-clock scheduling schemes [1-3] have been developed to address three major problems faced during the development of multiprocessor distributed applications.
Those three problems are as follows:
1. Clock Synchronization among nodes in the system.
2. Error-free data communication between nodes.
3. Detection of errors on nodes in the network.

Figure 1 shows a multiprocessor arrangement in which we can see one master node connected to N slave nodes through a certain arrangement either locally, through UART RS232/RS485 or through a CAN bus) [1]. In this arrangement the master node has an accurate clock which may (if required) take the form of a temperature compensated crystal oscillator (TCXO) or oven-controlled crystal oscillator (OCXO). The use of a TXCO or OCXO [4-5] is important in this case as we need a stable frequency to run the master node. A single timer is responsible for running the task scheduler on the master node. Apart from running its own scheduler the master node also generates "tick" messages. These tick messages are used to (i) drive the schedulers on the slave nodes, and (ii) transfer data to the slave nodes. The slave nodes in this arrangement have their own (low-cost) oscillators, and their own schedulers: however, schedulers on the slave are only updated when a tick message is received. The tick messages include a slave ID: only the slave with this ID will send back an acknowledgement message to the master. Using unique slave IDs ensures that although all the slaves share a common communication channel only one slave will send an acknowledge message.

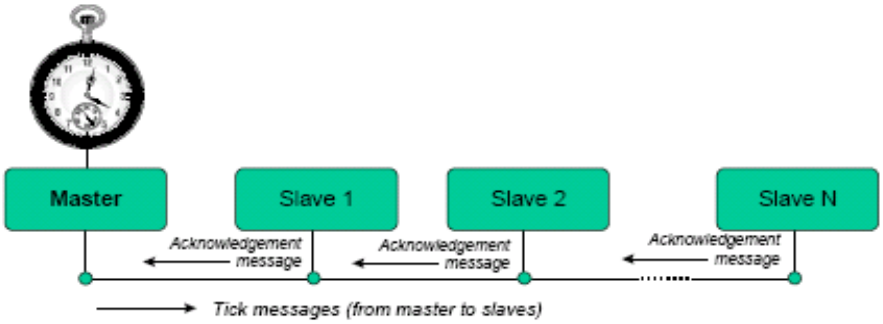

Figure 1: Shared Clock Scheduling Scheme

The scheme outlined above (and summarized in Figure 2) ensures that tasks running on the master and slave nodes are in sync. This arrangement requires transparent communication of tick messages from the master node and acknowledgement messages from slaves in order to work. The algorithm (implicitly) assumes that – under normal circumstances – all messages transmitted by the master will be received by all slaves and that all messages sent by individual slaves will be received by the master. If messages are lost then (in a basic wired design) the network will simply "fail silently". It is clear that attempting to implement such an algorithm in a wireless environment will immediately present a number of challenges. These issues are discussed in Section 2.

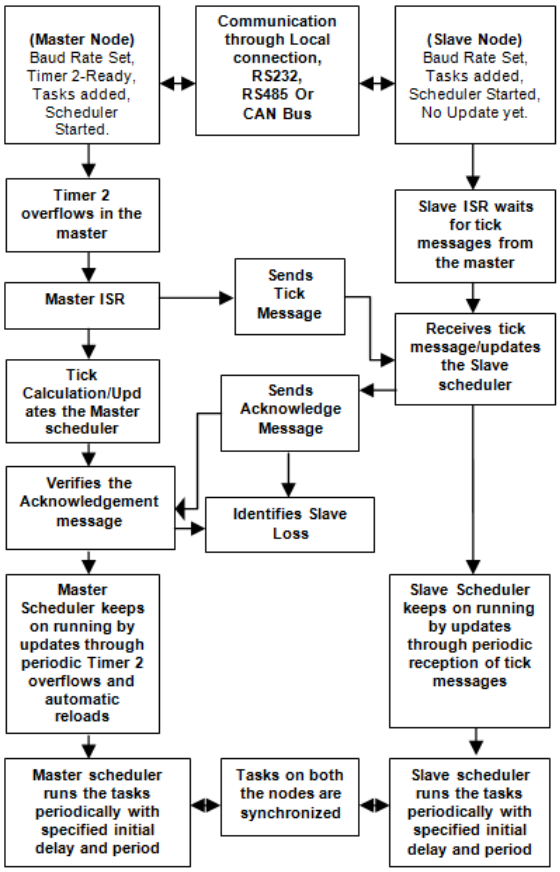

Figure 2: Shared Clock Scheduling Scheme block diagram

Issues with wireless implementation

Shared-clock scheduling scheme as described works on the mutual transmission of tick messages from the master node and in response acknowledgement messages from the addressed slave nodes. Even while working on a wired medium there is a possibility that a particular slave node may not receive a tick message in turn this will cause an imbalance of scheduler tick calculations between the master and that slave node. On the other hand when the master does not receive an acknowledgement message from that particular slave the master will try to either switch to a backup slave, enter a safe state or shutdown the entire network. Non reception of acknowledgment message can be a cause of slave node failure after last acknowledgement was sent by that particular slave. The master node in this case will notice the missing/failed slave after some time known as failure detection time [6]. Now this kind of situation hampers the normal working of such scheme. For the working of such scheduling scheme it is immensely important that the transmission/reception of messages is error free and ideally instantaneous without any delay. Before implementing on a wireless platform in its original form it must be kept in perspective that wireless media is prone to adverse external factors. These external factors will seriously impinge on the performance of the system. We review such factors in this section.

Electromagnetic Interference (EMI)

Wireless communication systems experience today unprecedented growth. The number of systems increases almost exponentially. In view of very limited spectrum available and large concentration of system over limited space (for example, in dense urban or indoor environments), the potential for mutual interference is tremendous [7]. The EMI effect is the major cause of disruption in wireless data transfer between multiple nodes in a distributed system. EMI or RFI may be broadly categorized into two types; narrowband and broadband. Narrowband interference usually arises from intentional transmissions such as radio and TV stations, pager transmitters, cell phones, etc. Broadband interference usually comes from incidental radio frequency emitters. These include electric power transmission lines, electric motors, thermostats, bug zappers, etc. Anywhere electrical power is being turned off and on rapidly is a potential source. Included in this category are computers and other digital equipment as well as televisions. The rich harmonic content of these devices means that they can interfere over a very broad spectrum. The coupling arrangement of interference to a wireless system can be shown with three different components.

- Source of Interference

(Electrical and Electronic devices / other transmitters and radiating systems) The source may be something more exotic such as a lightning strike, electrostatic discharge (ESD)

- Coupling path

(Air for wireless communication)

- Receptor/Sink

(The communication channel itself plus all the receivers)

In another way we can regard noise as unwanted (and usually uncontrollable) electrical signals interfering with the desired signal. Unwanted signals arise from a variety of sources, both natural and artificial. Artificial sources include noise from automobile ignition circuits, commutator sparking in electric motors, 60-cycle hum, and signals from other communication systems. Numerous artificial sources result from harmonics of the natural frequency. For example, spark plugs firing in car engines have a frequency on the order of thousands of rpm; consequently, although the fundamental frequency is less than 1 kHz, the energy emitted by this excitation is so strong that high-order harmonics can cause significant interference in radio systems. Natural sources of noise include circuit noise, atmospheric disturbances and extraterrestrial radiation [8]. The above-mentioned phenomena can introduce certain impairments which can cause the data transmitted wirelessly to be either corrupted or lost. Regardless of the source, interference prevention is and has been of a key importance in wireless systems. In the case of wireless implementation of TTC-SCSS, interference avoidance will have a prime importance. Because if the data sent from the master node towards the slave nodes is changed or lost due to the effect of discrete or continuous EMI, then it is going

to be difficult to operate the system in a proper way. If the designed system is prone to continuous residual EMI then it is going to be very difficult to maintain the integrity of the system throughout. Measures have to be taken in this regard to either reduce the effects of interference in the vicinity of the system or changes had to be done to the system structure to make it less responsive or ignore the sources of interference.

Propagation Delay

To reduce the impact of propagation delay on system performance, it is usual to include a guard-time between time slots. No signal is transmitted during this guard time, and as a result, much of the harmful effect of propagation delay is avoided [9]. In our case propagation delay is going to be the time taken by the tick message to travel from the master node and reach a slave node plus the time taken by the slave acknowledgment message to travel from that particular slave node and reach the master node considering the processing times of the message on both nodes as negligible. Propagation delay can cause a problem when it is greater than the tick interval. In a case like this the master node will not receive the acknowledge message and thus will hamper the scheduler operation. It is a remote possibility that such a scenario will occur but immense care should be taken during the topology design of such system. For that purpose the slave nodes should be kept in such an acceptable proximity in order to maintain delay free topology and keep the transmission and reception times well inbounds of the tick intervals.

Multi-path Fading

In wireless communications [10], the presence of reflectors in the environment surrounding a transmitter and receiver create multiple paths that a transmitted signal can traverse. As a result, the receiver sees the superposition of multiple copies of the transmitted signal, each traversing a different path. Each signal copy will experience differences in attenuation, delay and phase shift while traveling from the source to the receiver. This can result in either constructive or destructive interference, amplifying or attenuating the signal power seen at the receiver. Strong destructive interference is frequently referred to as a deep fade and may result in temporary failure of communication due to a severe drop in the channel signal-to-noise ratio. A common example of multi-path fading is the experience of stopping at a traffic light and hearing an FM broadcast degenerate into static, while the signal is re-acquired if the vehicle moves only a fraction of a meter. The loss of the broadcast is caused by the vehicle stopping at a point where the signal experienced severe destructive interference. Cellular phones can also exhibit similar momentary fades. In implementing a shared clock scheme on wireless platform these factors should be kept in mind in order to prevent interferences bad for the system performance and keep the signal strength in operating range.

Non-line-of-sight propagation

When only scattered waves arrive at the terminal, the signal envelope shows fast fading represented by a Rayleigh distribution against time. Such class of propagation channels is called non-line-of-sight (NLOS) [11]. Non-line-of-sight (NLOS) or near-line-of-sight is a term used to describe radio transmission across a path that is partially obstructed, usually by a physical object in the Fresnel zone. Many types of radio transmissions depend, to varying degrees, on line of sight between the transmitter and receiver. Obstacles that commonly cause NLOS conditions include buildings, trees, hills, mountains, and, in some cases, high voltage electric power lines. Some of these obstructions reflect certain radio frequencies, while some simply absorb or garble the signals; but, in either case, they limit the use of many types of radio transmissions, including most of those used for Wi-Fi. The acronym NLOS has become more popular in the context of wireless local area networks (WLANs) such as WiFi and WiMax because the capability of such links to provide a reasonable level of NLOS coverage greatly improves their marketability and versatility in the typical urban environments in which they are most frequently used. The influence of a visual obstruction on a NLOS link may be anything from negligible to complete suppression. An example might apply to a LOS path between a television broadcast antenna and a roof mounted receiving antenna. If a cloud passed between the antennas the link could actually

become NLOS but the quality of the radio channel could be virtually unaffected. If, instead, a large building was constructed in the path making it NLOS, the channel may be impossible to receive. NLOS links may either be simplex (transmission is in one direction only), duplex (transmission is in both directions simultaneously) or half-duplex (transmission is possible in both directions but not simultaneously). Under normal conditions all radio links including NLOS are reciprocal which means that the effects of the propagation conditions on the radio channel are identical whether it operates in simplex, duplex or half-duplex [11].

If the above-mentioned adverse effects on the wireless connectivity scheme are not taken into account and we implement the shared-clock system in its original form, the system performance will be extremely poor. So it is of the utmost importance that we first make the system able to deal with any unintended development and then implement it on a wireless platform. The following sections specifically deal with such occurrences.

Things that could go wrong

Shared clock architecture in a wired environment relies on a periodic exchange of messages between a designated master node and its subservient slave nodes. The reliability of the scheme depends upon the highest degree of communicational integrity. On a wired medium the performance of the system is high but even still the periodicity of massage exchange can get compromised, making the system's behavior erratic. Now in order to implement the scheme wirelessly enhances the probability of communicational errors, as of now the communication channel is more exposed to certain impairments (Noise, EMI etc). Here we list some of the most important things to take care of in order to keep the system's integrity intact while the nodes communicate wirelessly.

List of Factors:

- Master node fails.
- A single Slave node fails.
- Multiple Slaves fail.
- Slave node does not receive a tick message from Master node.
- Master node does not receive an acknowledgment message from Slave node.

Countering the above factors in a wireless environment will definitely increase the reliability and continuous performance of the system. We will discuss some of the ways in which these factors can be dealt with one by one.

Master Node fails

If for any reason the master node in the network fails, for the sake of redundancy it is feasible to keep a backup master. In the event of master node failure the backup master should take over, reset the network, establish communication with the slaves and start running its own and slave schedulers as before. If we do not want to reset the entire network and do not want to lose time while doing that, another alternative that we can use is to run the network with two running master nodes with independent power supplies but same oscillator module. In this case if one of the two fails the other keeps on running the system and there is going to be no need of resetting the system or re-establishment of communication with the slave nodes. This will impinge on the power consumption of the system though but for a safety critical application it should be acceptable.

A single Slave Node fails

If for any reason a single slave node in the network fails, for the sake of redundancy it is feasible to keep a backup slave. The backup slave should be switched ON which will keep on running the system.

Multiple Slaves fail

The procedure used for the failure of a single slave can be applied for a multiple slave failure or if not possible we may have to reconstitute the network and jump-start it again. But it is highly unlikely that a situation will arise that multiple slaves will fail simultaneously and create such a chaotic scenario.

Slave node does not receive a tick message from master node

In our design we use two different interrupt service routines (ISRs) on the slave nodes. One is triggered by the UART interrupt, which is always used in

normal operation. The second ISR is triggered by the timer 2 overflow on the slave. As long as the communication is fine and no tick messages are lost the slave is triggered by the UART interrupts through the tick messages sent by the master node and which in turn updates the slave schedulers for task performance. The timer 2 triggered ISR remains dormant at this time. The timer 2 on the slave node is setup (stopped and loaded with appropriate value for overflow) at the start inside the UART interrupted ISR and before leaving the UART interrupted ISR the timer is started. So if the slave does not get triggered by the arrival of a tick message through the UART, the previous tick would have started the timer on the slave and will automatically update the slave scheduler as both ISRs use the same scheduler update function. If after a lost tick another tick arrives this will make the UART triggered ISR to again stop the timer, load it again with the appropriate value, update the scheduler and before leaving the ISR will start the timer in case another tick goes missing. If a succession of tick messages get lost and are not received by the slave, the timer 2 ISR will take over and automatically update the slave scheduler until the communication is restored or until the master resets the network.

Master node does not receive an ack-message from a slave node

If the master node does not receive an acknowledgement from a slave it sends the next tick message instead of showing a network error. If the problem with the reception is brief and it goes away the master node keeps on working as normal and shows no error. But if the problem persists for more than 100 ticks or any specified number of ticks and no acknowledgement message returns back then the master node shows network error and takes appropriate action.

In the next section we give the schematic of the testbed that is used for simulating a S-C wireless environment. This testbed is designed as such that all the slave nodes have individual CAN connections with the master node, so it is very easy to inject controlled periodic faults into each CAN cable.

Schematics of wireless testbed

As first steps we have tested the software guidelines given in this paper for the wireless implementation of shared-clock scheme on RS-232 and RS-485 protocol based wired networks, these guidelines are applicable to any communication protocol used for shared-clock architecture. As we know an RS-232 protocol supports serial communication between only two nodes having single UARTS (one master and one slave). Communication between one master and several slaves can be achieved using RS-232 protocol, but for that we should either use microcontrollers with multiple UARTS or use external UARTS with single UART controllers.

The same case is with RS-485 protocol, this protocol supports a multi-drop system means more than 2 nodes (one master/several slaves) individually having one UART each. But the UART based RS-232 and RS-485 protocols didn't gave us a feeling of a wired type wireless layout. So we came up with the design schematic of a wired system testbed using shared-clock architecture that feels more like a wireless implementation schematic will behave and look. The system consists of 5 nodes (1 master node and 4 slave nodes. CAN protocol [12] is used for communication between the master and slave nodes. The topology of the network is kept as Star as shown in Figure 3. Even though the system is still wired, in this star topology the master node behaves exactly like a wireless transceiver beacon that transmits or broadcasts tick-messages intended for a particular slave or intended for all slaves. This beacon also receives acknowledgment messages from the slaves through their transmissions.

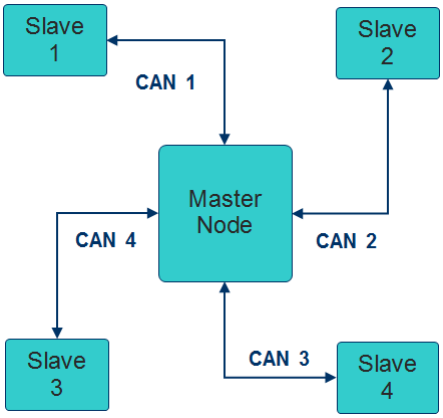

Figure 3: Testbed schematic in star topology connected via CAN

The master node used here is a Philips LPC-2294 microcontroller [13] on an Olimex LPC-E2294 rev. B development board [14]. The LPC-2294 has a support for 4 CAN interfaces on the board. This means that the master node can communicate with 4 different slaves simultaneously and does not need any external peripherals for adding more interfaces as such in the case of testbeds discussed earlier. This approach reduces the code size and ultimately reduces CPU power consumption which is vital for such resource constrained environments. 4 slaves are connected to the master node through 4 CAN interfaces. The slaves are Philips LPC-2129's microcontrollers [13] on Olimex LPC-P212X-B development boards [14], each of whom can support 2 CAN interfaces (we use only one of them). The idea here is to simulate the scenarios that emanate in a wireless environment using a wired setup and then apply the developed guidelines completely on a wireless platform.

In this paper the ideas and software guidelines we tested are given in various listings as a part of the ongoing research. The software guidelines adopted for wireless transition listed in this paper are for the RS-485 protocol-based system that we tested and are given in this paper for one reason only, to make the reader easily understand what we are trying to achieve. In Section 5 we will elaborate on tackling potential message losses in the system in either direction and provide evasive software strategies to avoid system failure.

Tackling Message Loss

Master to Slave

The description in sub section 3. 4 gives the remedy for tick messages lost when sent by the master node. The listed code sample (Listing 1) and block diagram (Figure 4) enhances the understanding further and shows how a slave node will cope with such a situation and automatically update its scheduler in the absence of tick messages.

```
/*-----------------------------------------*-
Main Slave ISR (triggered through UART interrupts)
-*-----------------------------------------*/
void SCU_B_SLAVE_Update(void)
interrupt INTERRUPT_UART_Rx_Tx
{
tByte Index;

if (RI == 1) // Must check the Receive Interrupt flag.
{
// Timer 2 has'nt been started in the first place

TR2 = 0;// Stop Timer 2
ET2 = 0;// Timer 2 interrupt is disabled
TH2 = 0xEC; // load timer 2 high byte
TL2 = 0x78; // load timer 2 low byte

// Default
Network_error_pin = NO_NETWORK_ERROR;

// Two-byte messages are sent (Ack) and received
(Tick)
```

```
// it takes two scheduler ticks to process each message
// Keep track of the current byte
if (Message_byte_G == 0)
 {
 Message_byte_G = 1;
 }
else
 {
Message_byte_G = 0;
 }
// Check tick data - send ack if necessary
// 'START' message will only be sent after a 'time out'

if (SCU_B_SLAVE_Process_Tick_Message() == SLAVE_ID)
 {
SCU_B_SLAVE_Send_Ack_Message_To_Master();

// Feed the watchdog ONLY when a
// *relevant* message is received
// (noise on the bus, etc, will not stop the
// watchdog. . . . . )
// START messages will NOT refresh the slave
// Must talk to every slave at regular intervals
SCU_B_SLAVE_Watchdog_Refresh();
}
// NOTE: calculations are in *TICKS* (not milliseconds)
for (Index = 0; Index < SCH_MAX_TASKS; Index++)
{
// Check if there is a task at this location
if (SCH_tasks_G[Index]. pTask)
 {
 if (SCH_tasks_G[Index]. Delay == 0)
{
// The task is due to run (Set the run flag)
SCH_tasks_G[Index]. RunMe = 1;
if (SCH_tasks_G[Index]. Period)
 {
 // Schedule periodic tasks to run again
 SCH_tasks_G[Index]. Delay =
 SCH_tasks_G[Index]. Period;
 }
}
 else
{
// Not yet ready to run
 // just decrement the delay
SCH_tasks_G[Index]. Delay -= 1;
}
 }
}
 RI = 0;// Reset the Receive Interrupt flag
 }
else
 {

// ISR call was triggered by Transmit Interrupt flag,
// after last character was sent

// RS485_Tx_Enable flag is reset here
RS485_Tx_Enable = 0;

TI = 0;// Must clear the Transmit Interrupt flag

}
TR2 = 1; // Start Timer 2
ET2 = 1; // Timer 2 interrupt is enabled
}
```

Listing 1: Code for main ISR assembly on any Slave in the network

The code in listing 1 is a scheduler update function which increments the tick count on the slave when a tick message has been received through the UART. In our design listing 1 is considered as the main ISR on the slave node. The main ISR is repeatedly called when the communication between master and slave is fine and no tick messages are lost. The main ISR embeds initialization controls of a second ISR that is also used on the slave node in conjunction with the main ISR. As clear from listing 1, when the main ISR is triggered by a UART interrupt through the arrival of a tick message, a timer (timer 2) on the slave node is stopped (Which was not even started at the power up time of the network) and loaded with an appropriate value exactly equal to the tick interval used in the system.

A tick interval is the time between two consecutive tick messages sent by the master node. The main ISR then goes forth and checks the contents of the message, sends an acknowledgment message to the master node, updates the slave scheduler by

incrementing the tick count on the slave (essential for synchronized task performance) and at the end starts the timer and enables its overflow interrupt. Now as evident the timer 2 on the slave is being initialized a step back from the main tick interval. The arrival of the next tick interval will stop the timer again and do all the process once more and repeatedly as long as the communication between the master and slave is fine and no tick message is lost. The timer 2 overflow triggers an alternate ISR, which we call here the dormant ISR, as it is always dormant and not used when the communication is fine. In an event when a tick message is lost, the main ISR cannot get triggered through the UART. In previous designs if an event like this happened, in such a situation the slave node was not able to reply with an acknowledgement message as it didn't received any tick message. So the master node after not receiving a reply from the slave would reset the system immediately.

This behavior is not acceptable if the system is working in wireless environment, as we know the probability of massage loss in wireless arena is much higher, in such a situation the system will keep on resetting all the time and prove catastrophically impaired for the performance of a safety critical application. The dormant ISR is used here in order to make the operation smooth and flexible. So when a tick message is lost the slave's main ISR can't get triggered through UART. But timer 2 on the slave was started through the last received tick message. When timer 2 overflows it generates an interrupt and triggers the dormant ISR. The dormant ISR is used for the automatic update of the slave scheduler. Listing 2 specifies the code for the dormant ISR. In our present design the dormant ISR does not sends an acknowledgment message as it does not know about the contents of the tick message. If successive tick messages are lost the slave scheduler gets automatically updated through timer 2 overflow interrupts. When communication gets better after the loss of some successive or single tick messages, the arrival of the next tick message will trigger the main ISR on the slave through the UART, again stopping the timer, taking the ISR it was using back into dormant state, keep on updating the scheduler and sending acknowledgment messages through the main ISR as long as successive tick messages from the master are received by the slave without loss. The probability of task de-synchronization among the master and slave(s) for a 1 to 100 tick messages lost is near to zero. Figure 4 gives a block diagram explanation of the procedure we discussed above. For more than 100 tick messages lost and how the master node should react if the slave is running on automatic updates through the dormant ISR and do not receive any acknowledgments, the system adapts another mechanism that is explained in the next sub-section.

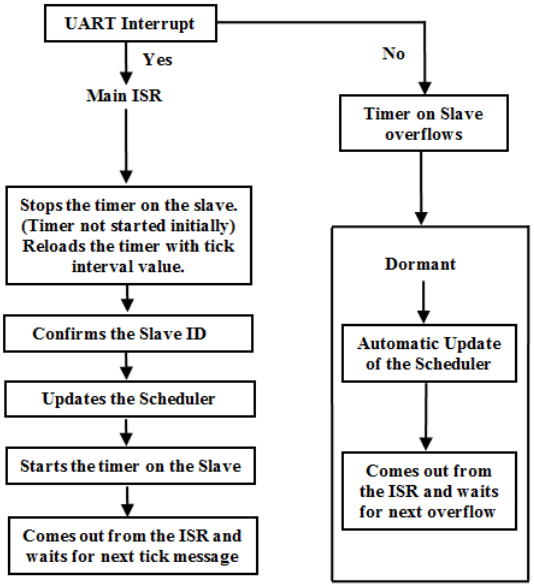

Figure 4: Active and Dormant ISR functionality

```
/*-----------------------------------------------------*-
Dormant Slave ISR (triggered through timer 2 interrupts)
-*-----------------------------------------------------*/
void SCU_B_SLAVE_Update_T2(void)
interrupt INTERRUPT_Timer_2_Overflow
 {
tByte Index;

if (RI == 0)// Must check the Receive Interrupt flag
 {

TF2 = 0;// Must manually clear timer 2 overflow
//interrupt flag
 // Default
Network_error_pin = NETWORK_ERROR;
SCU_B_SLAVE_Watchdog_Refresh(); //refresh the watchdog

// NOTE: calculations are in *TICKS* (not milliseconds)
for (Index = 0; Index < SCH_MAX_TASKS; Index++)
{
// Check if there is a task at this location
if (SCH_tasks_G[Index]. pTask)
 {
 if (SCH_tasks_G[Index]. Delay == 0)
{
 // The task is due to run (Set the task run flag)
SCH_tasks_G[Index]. RunMe = 1;
if (SCH_tasks_G[Index]. Period)
 {
 // Schedule periodic tasks to run again
 SCH_tasks_G[Index]. Delay =
 SCH_tasks_G[Index]. Period;
 }
}
 else
{
// Not yet ready to run the task:
// just decrement the task delay
SCH_tasks_G[Index]. Delay -= 1;
}
}
}
}
```

Listing 2: Code for dormant ISR assembly on any Slave in the network

Slave to Master

On a wireless platform there also arises a possibility that some of the acknowledgment messages from the slave(s) will not reach the master node and are going to be lost. The reason for that can be a slave loss or interference. If a slave is lost the master will try to switch over to a backup slave, if a backup slave is not available the master node will shut down the network for maintenance. But if message loss is due to interference or some other phenomena impinging on the communication channel, in our design the master node has a window of 100 tick elapses. What this means is that when the master node sends a tick message it waits for the acknowledgment to that message from a particular slave. If the acknowledgment message does not arrive, the master node increments the tick elapse counter by 1 and sends another tick message. If a slave node replies to the second tick message the master node decrements the tick elapse counter to zero. There are two reasons why an acknowledgment message does not arrive on the master node. First it is possible that a slave node didn't receive a particular tick message from the master node. Secondly due to interference it was lost inside the communication channel. The tick elapse counter on the master node allows 100 transmissions of tick messages for 100 successive lapses of acknowledgment messages. After the overflow of the tick elapse counter the master node takes evasive action by either switching to backup slave or shutting down the system for maintenance. If the topology of the system is kept symmetrical and slaves are operational there is a high probability that the communication is going to restore inside the time taken by 100 tick elapses. Time calculation for 100 tick elapses can be done through a simple equation given as follows,

$$T = ti \times 100 \qquad (1)$$

Were

T = time taken by 100 tick elapses.

ti = tick interval.

So if tick interval is 1ms, 100 tick elapses will take 0. 1 seconds and if tick interval is kept at 5ms, 100 tick

elapses will take half a second (0. 5 seconds). Figure 4 here gives a block diagram explanation of the preceding sub-sections and code listing 3 gives code example for sub section 5. 2.

```
/*-----------------------------------------*-
Main Master ISR (triggered by timer 2 overflows)
-*-----------------------------------------*/
void SCU_B_MASTER_Update_T2(void)
interrupt INTERRUPT_Timer_2_Overflow
{
tByte Task_index;
tByte Previous_slave_index;
static tByte tick_elapse = 0;

TF2 = 0; // Must manually clear timer 2 overflow
 //interrupt flag

// Refresh the watchdog
SCU_B_MASTER_Watchdog_Refresh();

// Default
// Network_error_pin = NO_NETWORK_ERROR;
(for our design)

// Keep track of the current slave
// FIRST VALUE IS 0
Previous_slave_index = Current_slave_index_G;

// Assume 2-byte messages sent and received
// it takes two ticks to deliver each message

if (Message_byte_G == 0)
{
 Message_byte_G = 1;
}
else
{
 Message_byte_G = 0;

if (++Current_slave_index_G >=
NUMBER_OF_SLAVES)
{
 Current_slave_index_G = 0;
}
}

// Check that the appropriate slave responded to the
// previous message: if it did, store the data sent by
// this slave)
if
(SCU_B_MASTER_Process_Ack(Previous_slave_ind
ex) ==
 RETURN_ERROR)
{
tick_elapse++;
Network_error_pin = NETWORK_ERROR;

// If we have lost contact with a slave, we attempt to
// switch to a backup device (if one is available) as we
reset
// the network.  We do not do this every tick (or the
network will
// be constantly reset).  Choose a value of
SLAVE_RESET_INTERVAL
// to say 5 seconds

if ((++Slave_reset_attempts_G[Previous_slave_index]
> = SLAVE_RESET_INTERVAL) && (tick_elapse
== 100))
{
tick_elapse = 0;
SCU_B_MASTER_Reset_the_Network();
}
}
else
{
//Do nothing
}

// Send 'tick' message to all connected slaves
// (sends one data byte to the current slave)
SCU_B_MASTER_Send_Tick_Message(Current_sla
ve_index_G);

// NOTE: calculations are in *TICKS* (not
milliseconds)

for(Task_index=0;Task_index<SCH_MAX_TASKS;T
ask_index++)
{
// Check if there is a task at this location
if (SCH_tasks_G[Task_index]. pTask)
{
 if (SCH_tasks_G[Task_index]. Delay == 0)
{
```

```
// The task is due to run, Increment the task run flag
SCH_tasks_G[Task_index]. RunMe += 1;
if (SCH_tasks_G[Task_index]. Period)
 {
 // Schedule periodic tasks to run again
 SCH_tasks_G[Task_index]. Delay=
 SCH_tasks_G[Task_index]. Period;
 }
}
 else
{
// Not yet ready to run: just decrement the task delay
SCH_tasks_G[Task_index]. Delay -= 1;
}
 }
}
 }
```

Listing 3: Code for Master node handling tick elapses

After elaborating on the functionality of guidelines we have used for making the operation of a shared-clock scheduling scheme in wireless environment more flexible, in the next section we will discuss the simulation of fault-injection system for such a system using shared-clock architecture while working in wired environment. We will now discuss injection of periodic faults using the communication lines of the testbed considered in section 4 and also briefly elaborate its necessity in our simulation of faults.

Fault injection

As the guidelines are tested on wired platforms such as RS232/RS485 and CAN (Controller area network), a fault injection system should be also one that can inject faults like severing the network cables for a specified amount of time or periodically open and close all the communication lines of the network by the use of a TTC interrupt scheduler. The system we used for this purpose is shown in Figure 5. The DG417 is a monolithic CMOS analogue switch designed to provide high performance switching [15].

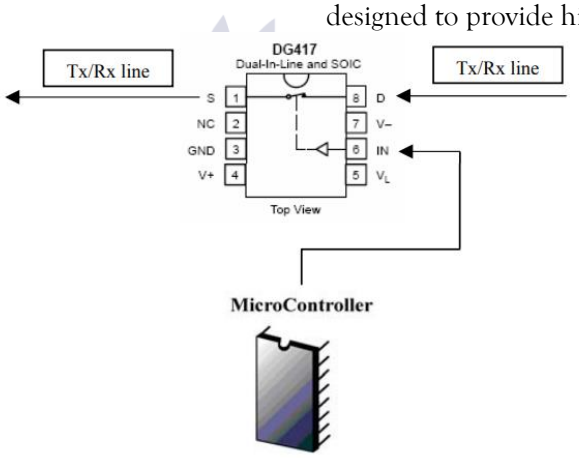

Figure 5: Fault injection in the Communication lines

The DG417 series is ideally suited for portable and battery powered industrial and military applications requiring high performance and efficient use of board space. The role of the microcontroller unit in this system is evident from Figure 5. We simulate wireless environment through wired implementation because of the complexity of injecting controlled faults in wireless environment. The fault-injector is an Olimex LPC-P212X-B development board [14] running an interrupt scheduler. The external interrupts are generated by the master node on a GPIO port pin, which are used to interrupt the fault-injector on a particular GPIO port pin. These external interrupts are used to run the scheduler on

the fault-injector in synch with the master node's scheduler. The scheduler on the fault-injector then generates hi/low voltages (0 t0 3. 28 volts) on a particular GPIO port pin with a certain delay and period. These alternating voltages are applied to the CMOS analogue switches injected in each of the 4 CAN cables (CAN High and CAN Low) in order to make them open and close according to the task scheduled in the fault-injector scheduler. Results from a 3-day run are presented in section 7 as follows.

Results

Figure 6 below shows a timing sequence for fault injection setup on the fault-injector microcontroller.

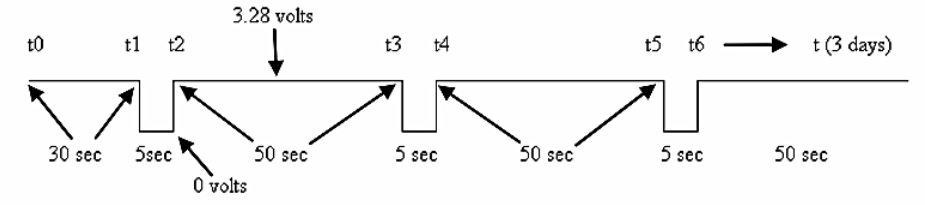

Figure 6: Scheduler timing sequence for Fault Injection

As we injected one fault inside each set of 55 seconds, so the number of faults injected into the system during a 3-day run are calculated as follows,
259200 – 35 = 259165 sec
(259165 sec is the time of 3 days minus the initial delay + first fault duration)
Number of faults without the first fault = 259165 / 55 = 4712 faults

Here t0.... t1 (30 sec) shows the initial delay before the first fault is injected, t1.... t2 (5 sec) shows fault duration of first fault, t2.... t3 shows the time gap between first and second fault. After that we kept the same sequence as evident from Figure 6 for 3 consecutive days. By removing the initial delay and first fault duration (t0.... t2) from the equation, it is evident that one fault was injected in each set of 55 seconds (t2.... t4), (t4.... t6) ...and so on for 3 days (72 hours = 4320 minutes = 259200 seconds). The number of faults injected and other specs are given in table 1.

So Total number of faults injected = 4712 + first fault = 4713 faults
Even after injecting the above number of faults into each communication channel of the system at a time the system was able to recover instantly after 100 msec (one tick interval) each time and kept the overall tasks running on the system in synch.

Detail	Amount
Days	3
Seconds	259200
Tick-interval	100 msec (Overall testbed)
Initial delay	30 sec
Fault duration	5 sec
Fault-sample time	1 fault every 55 sec
Total Faults	4713

Table 1: Specifications of a 3-day experimental run

8 Conclusions

In this paper, we have given and discussed some major guidelines for using shared-clock scheduling schemes on wireless platforms. We compared the original form implementation of the shared-clock scheme with the modified form and came to the conclusion that it is not feasible to implement the scheme using the original format. We made the scheme flexible to cope with the unavoidable impingement of external factors which will produce

communicational impairments and degrade the performance of the system. We have shown the behavior of the system to the introduction of faults. The guidelines are handled inside the software so it makes the scheme very cost effective to be implemented on wireless platforms. Wireless technology is also cost effective as it removes all the cabling and cuts down maintenance costs for fixing and replacing aging cable networks.

REFERENCES

M. J. Pont, Patterns for Time-triggered Embedded Systems: Building Reliable Applications with the 8051 Family of Microcontrollers, Addison Wesley/ACM Press, Harlow, 2001.

M. J. Pont, Supporting the development of time-triggered co-operatively scheduled (TTCS) embedded software using design patterns, Informatica 27 (1) (2003) 81–88.

M. J. Pont, M. P. Banner, designing embedded systems using patterns: A case study, Journal of Systems and Software 71 (3) (2004) 201–213.

An Analysis of Frequency Stability for TCXO Fujii, S.; Uchida, H. 29th Annual Symposium on Frequency Control. 1975 Volume, Issue, 1975 Page(s): 294 - 299

The modern OCXO quartz oscillators requirements and parameters, Weiss, K. ; Gniewinska, B. ; Nafalski, L. Laser and Fiber-Optical Networks Modeling, 2004. Proceedings of LFNM 2004. 6th International Conference on Laser and Fiber-Optical Networks Modeling, 2004. Volume, Issue, 6-9 Sept. 2004 Page(s): 209 - 212

D. Ayavoo, M. J. Pont, M. J. Short, S. Parker, Two novel shared-clock scheduling algorithms for use with 'Controller Area Network' and related protocols. Microprocessors & Microsystems, Volume 31, Issue 5 (August 2007), Pages 326- 334, 2007.

EMC/EMI analysis in wireless communication networks
Loyka, S. Electromagnetic Compatibility, 2001. EMC. 2001 IEEE International Symposium on Volume 1, Issue, 2001 Page(s):100 - 105 vol. 1.

Simon Haykin and Michael Moher, Modern Wireless Communications, Pearson Prentice Hall, Pearson Education, Inc. Upper Saddle River, NJ 07458, 2005.

T. S. Rappaport, Wireless Communications: Principles and practice, Second Edition, Prentice Hall.

Kapil Chawla, Xiaoxin Qiu. Quasi-Static Resource Allocation with Interference Avoidance for Fixed Wireless Systems. IEEE Journal on selected areas in communications, vol 17, No. 3, March 1999.

Giuseppe Ferrara, Maurizio Migliaccio, Antonio Sorrentino. Characterization of GSM Non-Line-of-Sight Propagation Channels Generated in a Reverberating Chamber by Using Bit Error Rates. IEEE Transactions on Electromagnetic Compatibility, Vol. 49, NO. 3, August 2007.

R. G. Bosch, CAN specification version 2. 0: Robert Bosch GmbH, Postfach 50, D-7000 Stuttgart 1, Germany, 1991.

NXP semiconductors. http://www. nxp. com.

http://www. olimex. com/dev/index. html

http://www. vishay. com.