# 1 System Design

## 1.1 List of main structures and operations:

### 1.1.1 Sender task:

It sends a message containing current time in system ticks to the receiver. The message is either blocked or sent depending on the queue cells. If the queue is full the message is blocked and if there is at least an empty cell in the queue, the message is sent successfully. This task can only be executed when it takes the sender semaphore which is released through the call back function of the sender timer.

### 1.1.2 Receiver task:

It receives a message containing current time in system ticks from the sender. It checks whether the queue is empty or not, so if the queue is empty the receiver tasks sleeps and if the queue is not empty the task receives the message at the front of the queue and print it on the screen. When the received messages =500 the Init function is called. This task can only be executed when it takes the receiver semaphore which is released through the call back function of the receiver timer.

### 1.1.3 Sender Timer:

It is a periodic timer whose call back function is to release the receiver semaphore which enables the receiver task to take the semaphore and do its job. The period of this timer changes every time we call the Init function. The different periods are {100, 140, 180, 220, 260, 300} represented in mSec. Every time the receiver task received 500 messages the period changed to the next value.

### 1.1.4 Receiver Timer:

It is a periodic timer of a certain period (200mSec) whose call back function is to release the receiver semaphore which enables the receiver task to take the semaphore and do its job.

### 1.1.5 Sender semaphore:

The sender task is to take this semaphore to perform its task but the semaphore is released in the call back function of the sender timer. So this semaphore is released every time the sender timer fires.

### 1.1.6 Receiver semaphore:

The receiver task is to take this semaphore to perform its task but the semaphore is released in the call back function of the receiver timer. So this semaphore is released every time the sender timer fires (every 200mSec).

### 1.1.7 The Queue:

There is only one queue to which the sender task sends messages and from which the receiver task receives messages. The size of this queue affects the number of blocked and successfully sent messages.

### 1.1.8 Init function:

This function prints the number of blocked and successfully sent messages then it resets them. It also resets the queue and makes it empty. It changes the period of the sender timer every time we call it through the array {100, 140, 180, 220, 260, 300} represented in mSec. When all values in the array are used, it deletes the timers and prints a message “Game Over”.

### 1.1.9 Main function:

In this function we call the Init function for the first time. We also create semaphores, timers, tasks and the queue. Then we call the scheduler to start executing.

## 1.2 Pseudo-code for the design and main structures of our code:

```
void SenderTask (void *p)
{
xSemaphoreTake(SendSemaphore)
instrucyions of the Sender
}
--------------------------------------
void SendAutoReloadTimerCallback()
{
xSemaphoreGive(SendSemaphore);
}
--------------------------------------
void Init ()
{
//Print number of successfully sent
messages and number of blocked message
then reset them (not for first call)
// Clears the queue
// Reset the sender timer period Tsender
   to the next value
//destroy timers and stop execution after
   using all values of Tsender
}
```

```
void ReceiverTask (void *p)
{
xSemaphoreTake(ReceiveSemaphore)
instrucyions of the Receiver
}
 ------------------------------------
void ReceiveAutoReloadTimerCallback()
{
xSemaphoreGive(ReceiveSemaphore);
}
 ------------------------------------
int main()
{
// creating semaphores
// creating queue and defining its size
Init();//calling Init for the first time
//creating periodic timers
//creating sender and receiver tasks
vTaskStartScheduler();
return 0;
}
```

## 1.3 Design of structures and Flow of the operations:

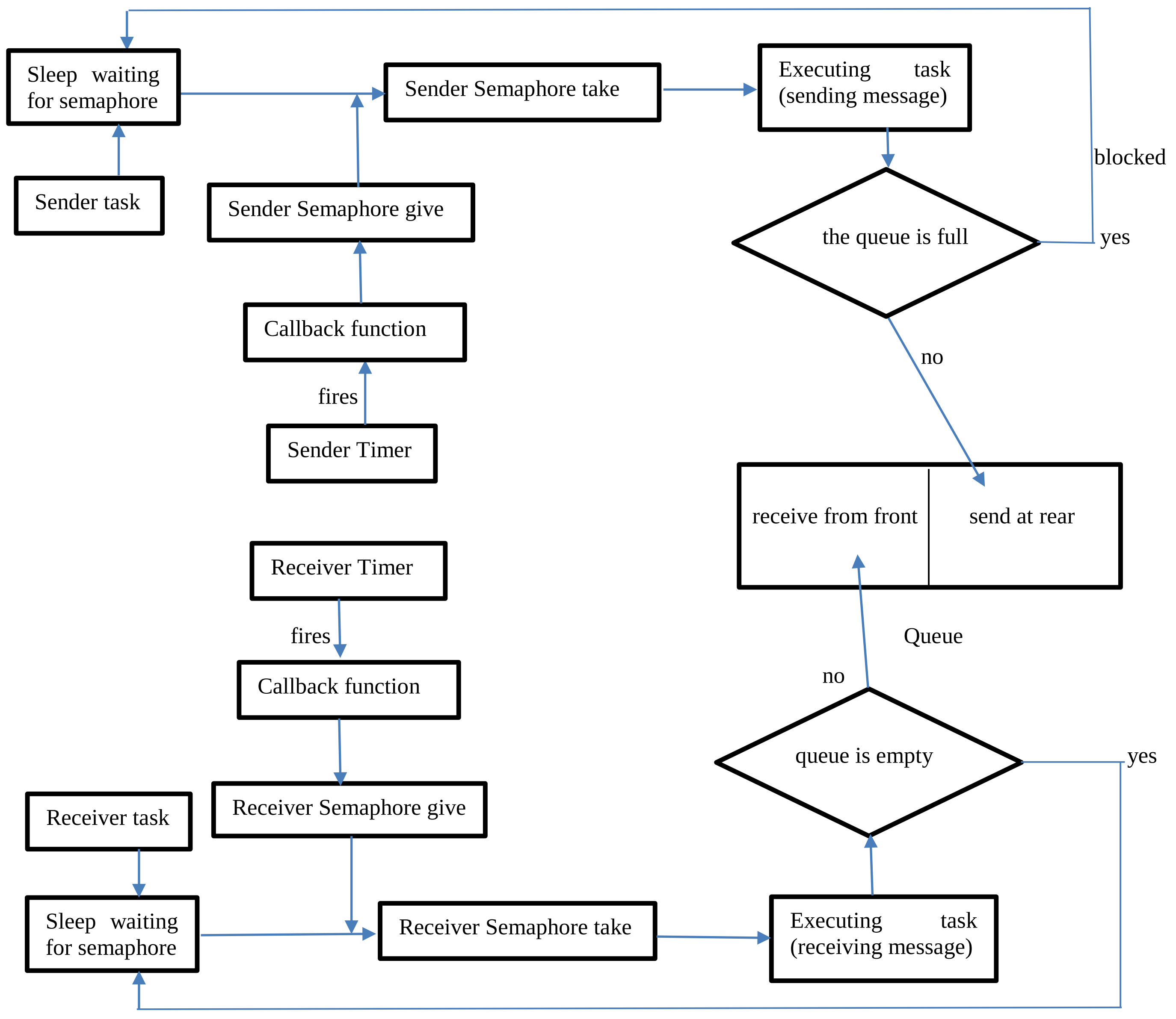
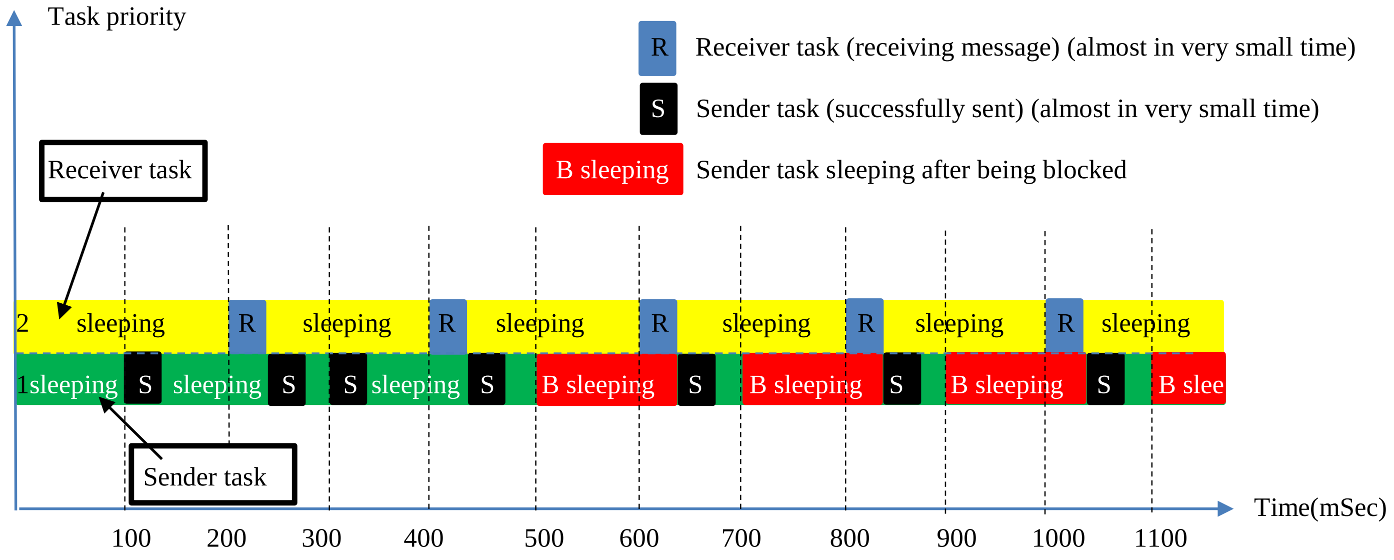


Figure1: Design of structures and Flow of the operations

## 1.4 Analysis of design:

Analysing our design makes us expect the folloing time diagram for the output at the first 1000mSec for queue of size=2:

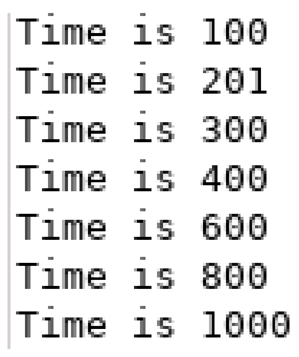


Task priority
R Receiver task (receiving message) (almost in very small time)
S Sender task (successfully sent) (almost in very small time)
B sleeping Sender task sleeping after being blocked
Receiver task
Sender task
2 sleeping R sleeping R sleeping R sleeping R sleeping R sleeping
1sleeping S sleeping S S sleeping S B sleeping S B sleeping S B sleeping S B slee
Time(mSec)
100 200 300 400 500 600 700 800 900 1000 1100

Figure2: Timing diagram of the first 1000mSec

Note that the width of R and S is not scaled on the timing diagram as the actual width (if scaled) should be much smaller than this width but we used it to visualize executing tasks.

Messages at First 1000mSec of the queue of size=2 to verify our analysis:



```
Time is 100
Time is 201
Time is 300
Time is 400
Time is 600
Time is 800
Time is 1000
```

Figure3:Messages at the first 1000mSec

As shown in figure3:

-The messages at time=500,700,900 mSec are blocked as the queue is full at these times so the sender task cannot send messages.

-The receiver task receives messages every 200mSec with higher priority than the sender task.

-The sender task sends message every 100mSec with lower priority than the receiver task.

## 2 Results and Discussion

### 2.1 Results in case that size of the queue=2:

| Tsender | successful | blocked | all sent |
|---|---|---|---|
| 100 | 501 | 498 | 999 |
| 140 | 501 | 214 | 715 |
| 180 | 501 | 54 | 555 |
| 220 | 500 | 0 | 500 |
| 260 | 500 | 0 | 500 |
| 300 | 500 | 0 | 500 |

Table1: Number of messages at queue of size=2

-The number of total sent messages as function of sender timer period:

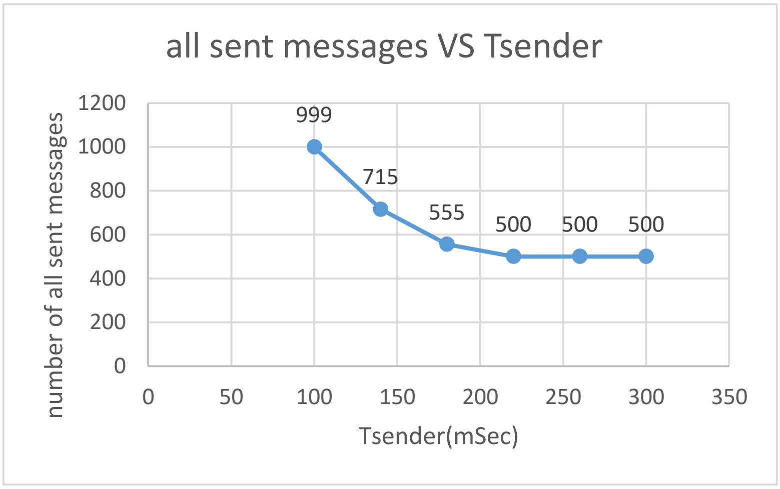


Figure4: total sent messages as function of Tsinder

The gap between the number of sent and received messages in the period is a result of the blocked messages which are blocked because the sender timer has a smaller period than the receiver timer so the messages accumulate in the queue, so when the queue is full, other messages from the sender task are blocked until the receiver task receive a message and the queue has empty space.

That makes sense as when the sender timer period gets larger, the gap between the number of sent and received messages gets smaller until it reaches 0 when the receiver timer has a smaller period than the sender timer as the queue is not full anymore because the receiver task receives messages with a higher rate than the rate of sending messages by the sender.

-The number of blocked messages as function of Tsender:

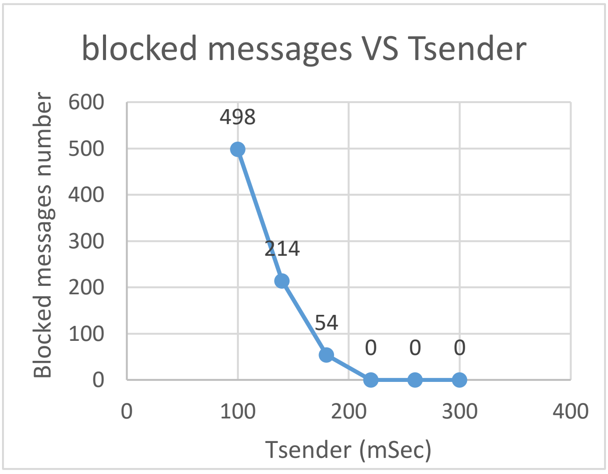


Figure5: blocked messages as function of Tsinder

-The number of successfully sent messages as function of Tsender

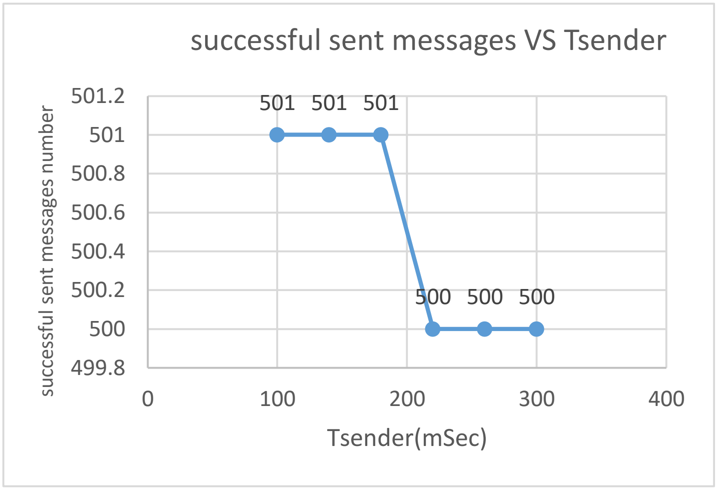


Figure6: successful sent messages as function of Tsinder

## 2.2 Results in case that size of the queue=20:

| Tsender | successful | blocked | all sent |
|---|---|---|---|
| 100 | 519 | 480 | 999 |
| 140 | 519 | 196 | 715 |
| 180 | 519 | 36 | 555 |
| 220 | 500 | 0 | 500 |
| 260 | 500 | 0 | 500 |
| 300 | 500 | 0 | 500 |

Table2: Number of messages at queue of size=20

-The number of total sent messages as function of sender timer period:

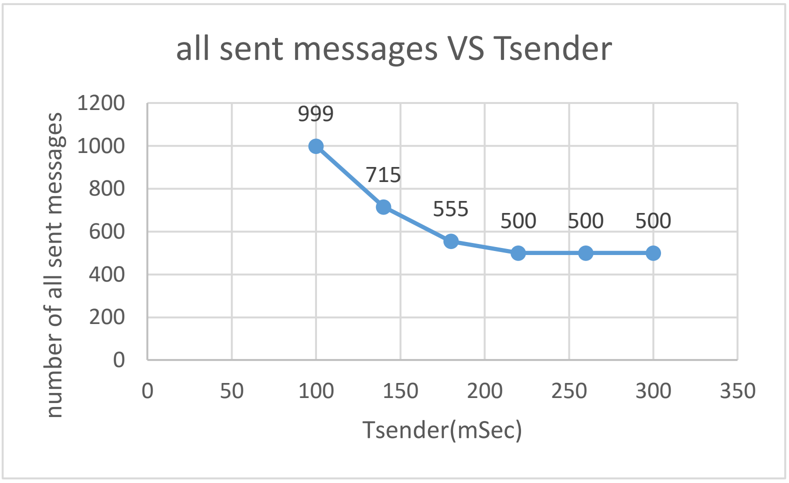


Figure7: total sent messages as function of Tsinder

-The number of blocked messages as function of Tsender:

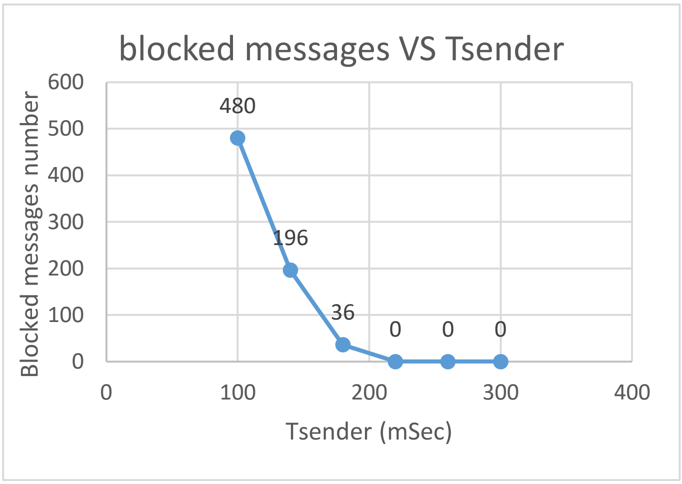


Figure8: blocked messages as function of Tsinder

-The number of successfully sent messages as function of Tsender

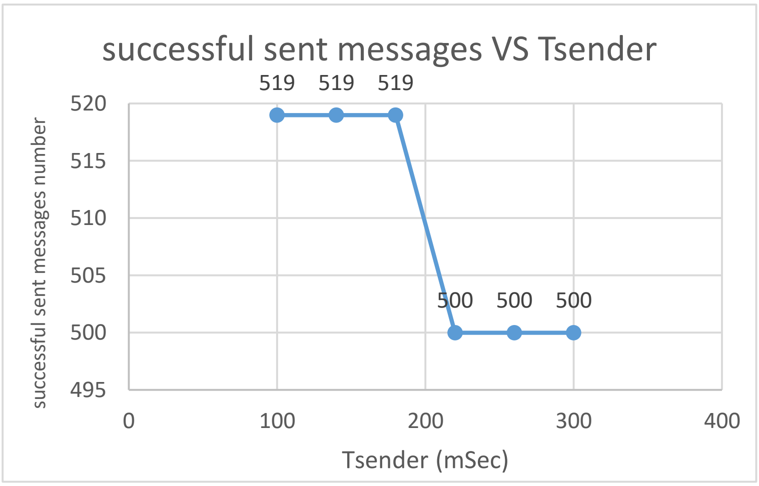


Figure9: successful sent messages as function of Tsinder

When the size of the queue increases the blocked messages decreases and the successfully sent messages increases because the messages' accumulation takes a larger time to make the queue full so the messages during this time are not blocked as the queue still has empty spaces.

## References

[1] Reference Manual for FreeRTOS version 9.0.0 issue 2. Real Time Engineers Ltd. 2016.