© Universiti Tun Hussein Onn Malaysia Publisher's Office

IJIE

http://penerbit.uthm.edu.my/ojs/index.php/ijie
ISSN : 2229-838X e-ISSN : 2600-7916

The International Journal of Integrated Engineering

An Integration of Open-Source Resources in Distance Teaching for Real-Time Embedded System Using Arduino Microcontroller and Freertos

Kim Seng Chia[1*]

[1]Faculty of Electrical and Electronic Engineering,
Universiti Tun Hussein Onn Malaysia, Parit Raja, Johor, 86400, MALAYSIA

*Corresponding Author

DOI: https://doi.org/10.30880/ijie.2022.14.06.017
Received 10 August 2021; Accepted 7 November 2021; Available online 10 November 2022

Abstract: Distance learning has become a crucial alternative in education system worldwide since the Covid-19 pandemic happened. An affordable and accessible solution is essential to enable a quality remote teaching and learning experience for each student. Real-Time Embedded System (RTES) is about applying real-time system in an embedded system (e.g., microcontroller) in ways that both logical and temporal requirements are fulfilled. A special attention shall be on the temporal response analysis so that students can have a clear distinction between microcontrollers with and without real-time system. Since there is a lack of accessible traceable studies about how to utilize the open-source platform in introducing the key RTES concepts, this study aims to evaluate the feasibility of an integration of open-source resources (i.e., SimSo simulator, Arduino platform and FreeRTOS API) in teaching RTES remotely and to provide real-time learning experience about RTES concepts without additional components or wiring out of school. First, each student was introduced to the ways to integrate existing open-source resources so that each of them could develop RTES in their places using a FreeRTOS compatible microcontroller. RTES concepts of task, hyperperiod, task scheduling, pre-emption, static-priority scheduling algorithms, and mutual exclusion in resource sharing were delivered using the proposed infrastructure. Results show that an obvious context switch could be observed when pre-emption happened. Finding indicates that the proposed integration was useful for students to understand the complex RTES concepts e.g., task scheduling, pre-emption, and mutually exclusion. Since positive responses were received from students in two consecutive semesters, the proposed infrastructure is an affordable and accessible distance learning alternative in assisting students to understand RTES concepts.

Keywords: Real-Time, embedded system, arduino, FreeRTOS, distance learning

1. Introduction

Open-source platforms are getting more attention in various areas, including education. The advantages and disadvantages of Arduino based open source platform in embedded system education were analysed in recent review [1]. The review study concluded Arduino based engineering education is promising to address the current challenges (i.e. student-related, lecture-related, content-related, and instructor-related challenges) faced by embedded engineering education [1]. This is in agreement with [2] who reported that the use of a Arduino Nano Board and USB powered portable instrumentation devices in laboratory exercises positively affected student confidence and improved the retention of students who took the course. Besides, various open source software and cloud platforms were evaluated in teaching Internet of things (IoT) course [3]. Even though various real-time IoT experiments and applications were

**Corresponding author: kschia@uthm.edu.my* developed and explored [3], the focus was on IoT instead of the temporal analysis and key concepts of real-time system that were not provided.

Real-Time Embedded System (RTES) is about applying real-time system in an embedded system (e.g., microcontroller) in ways that both logical and temporal requirements are fulfilled. In most studies, the term of "real-time" was used without reporting the temporal requirements of a system. This kind of "real-time" may refer to the time in users' perspective (e.g. a measured value with respective measurement time in each second [4]) instead of RTES that aims not only to produce a desired output but also to meet its temporal requirements. RTES focuses on both temporal and logical correctness in optimizing the energy efficiency and effectiveness of an embedded system. Thus, various low-cost innovative academic approaches in improving the teaching and learning of real-time embedded system related concepts with simpler infrastructure were studied. For instance, ARM7 microcontroller-based hardware-in-the-loop (HIL) environment was proposed to overcome the space and cost needed to deliver real-time and embedded control course [5]. This is considered bulky and expensive when it is compared with the existing popular open-source platform. Particularly, the uses of Arduino open-source platform in teaching embedded system related courses have been reported in numerous recent works. The benefits of Arduino platform in education were demonstrated by Peter Jamieson in exposing students to various complex and challenging embedded system topics via various Arduino based activities [6]. Nevertheless, challenge for educators is about how to identify what was done by a student as there are plenty of reusable designs in open source community [6]. This may be addressed by a proper instruction that requires students to highlight their contribution against the existing similar works [1]. Despite various Arduino based studies were published, there is a lack of traceable studies about how to further utilize the advantages of open-source platform (i.e., affordable and accessible) to introduce the key RTES concepts to students in terms of hyperperiod, pre-emption, tasks scheduling, and resource sharing. Even though there was a challenge to apply Arduino to teach embedded operating systems [1], an open source real-time operating system of FreeRTOS may address this challenge.

Recently, Hee et. al. reviewed three types of embedded operation systems (OS) i.e. super loop embedded OS, cooperative OS (e.g. Contiki and TinyOS), and real-time OS (e.g. commercial RTOS-μC/OS-III, FreeRTOS, eCos, embOS, Nucleus RTOS, ThreadX) [7]. Among these real-time OS, both FreeRTOS and eCos are distributed under open-source license. Since FreeRTOS real-time kernel or scheduler has a massive ported microcontrollers selection, this can simplify the product development as various complex software modules are available [7]. Other recent review on internet of thing (IoT) OS (e.g. Tiny OS, Contiki OS, FreeRTOS, and RIOT) can be found at [8][9][10]. In general, each OS has its own advantages and limitations. Among these open-source OS, FreeRTOS appears to be the most suitable OS for RTES learning due to its de facto standard and accessible resources. Additionally, FreeRTOS is commonly used RTOS for Unmanned Aerial Vehicles (UAV) (e.g. flapping-wing flying robotic bird [11]) due to its small storage space support for various high real-time requirement embedded platforms [12]. FreeRTOS was also used to enable multi-tasks control system that outperformed traditional control system in designing a motor control driver [13].

Various traceable works that used FreeRTOS in teaching and learning activities are reported as follows. FreeRTOS was studied to enhance students' practical experiences in five key real-time concepts i.e. task management, queue management, interrupt management, resource management, and memory management [14]. However, the design used for users to interact with the microcontroller with FreeRTOS were not reported. This concern has been addressed by recent study that depicted their proposed hardware and software infrastructure that could enhance the learning process of real-time scheduling in undergraduate courses using Arduino, FreeRTOS, and some MATLAB scripts [15]. The proposed Arduino and FreeRTOS in real-time system introductory course was relatively affordable compared to previous related works that used commercial or expensive platforms [15]. Nevertheless, an external RAM (SRAM chip) with SPI interface wiring were needed for data logging in the proposed solution. This may increase financial barrier for implementing a distance learning.

Distance learning is a new norm now since the Covid-19 pandemic happened globally. The awareness and readiness of students have strong influence on the adoption of distance learning [16]. Recent study shows that there is a need to improve the delivery of online teaching and learning in Malaysia e.g. make the learning to be interesting and innovative [17]. Since project-based learning has been reported as an effective way in enhancing the knowledge and capability of students in solving real-world engineering problems using embedded systems design concepts [18], an affordable and accessible solution is urgently needed to enable a quality remote teaching and learning experience for each student. Since a real-time experience about how the tasks (or threads) were scheduled is vital to enhance students learning experiences about important real-time system concepts e.g., pre-emption and task scheduling, there is a need to design and develop an alternative that provides a real-time experience for students with a minimal cost. Thus, this study aims to evaluate the feasibility of an integration of Arduino platform and FreeRTOS in teaching RTES remotely and in providing real-time learning experience about RTES concepts on the fly using a single Arduino microcontroller without additional components or wiring out of school.

2. Methods and Materials

3. Background of the Course

Real-Time Embedded System (RTES) is a final year course that offered by Faculty of Electrical and Electronic Engineering, Universiti Tun Hussein Onn Malaysia to students who enrolled the Bachelor Degree of Electronic Engineering (with Honour) – "*This programme is accredited by the Engineering Accreditation Council (EAC) which comprises five stakeholders namely Board of Engineers Malaysia (BEM), the Institution of Engineers Malaysia (IEM), Industry Employers, Malaysian Qualification Agency (MQA) dan the Public Service Department (JPA).*" Students are required to complete a pre-requisite course of computer programming so that they are ready to learn about real-time concept in embedded system using C programming. This is a three credit hours course (three lecture hours per week) that aims to equip students with the knowledge and ability to solve RTES problems and develop RTES for electronic engineering applications. Four main topics were covered i.e., embedded system, real time operation system, scheduling, and resource sharing. The assessment tools of quiz (15%), assignment (20%), test (15%), and final exam (50%) were used to evaluate the performance of students. At the end of the course, all students were invited to fill-in anonymous feedbacks about their learning experience for continuous quality improvement (CQI).

3.1 Distance Learning Infrastructure

The proposed distance learning infrastructure consists of three main components i.e., an Arduino board that supported FreeRTOS library (www.freertos.org); a computer with Arduino integrated development environment (IDE) (www.arduino.cc); and a USB cable to connect both hardware. Due to the simplicity of the proposed infrastructure, the financial and technical barriers in implementing the proposed infrastructure were minimized. With the assumption that all learners were equipped with a basic internet support and a computer, the additional cost that was needed to implement the proposed infrastructure was only having a microcontroller that is compatible with Arduino platform and FreeRTOS. Since students can reuse their existing compatible microcontrollers from previous courses (e.g., Integrated Design Project) or concurrent courses (e.g., Final Year Project), the cost that is needed to implement the proposed remote learning infrastructure is negligible for these students. The list of required open-source software that students were required to install in their computer is tabulated in Table 1. This software is free and accessible via the given links. Since the proposed distance learning is a combination of open-source resources that are available for each student, students are expected to actively engage their learning in deeper understanding about RTES.

Table 1 - The list of open-source technologies that needed to perform the proposed low-cost portable Learning technology for real-time embedded system learning

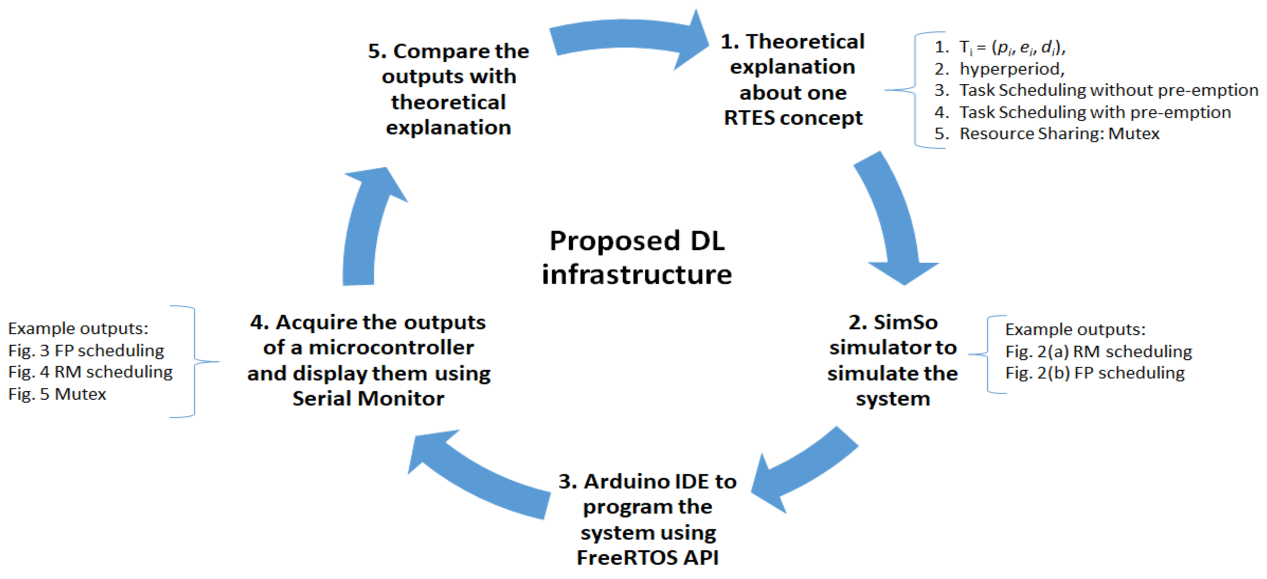
Open technologies	Hardware or Software	Price	Purpose
One Arduino and FreeRTOS compatible microcontroller and a suitable USB cable	Hardware	<20 USD	To execute the C programming to perform a real-time application in an embedded system.
FreeRTOS real-time operating system kernel (https://www.freertos.org/)	Software	Free	To schedule tasks according the assigned priority and period in the microcontroller.
Arduino IDE (https://www.arduino.cc/en/software)	Software	Free	To program a compatible microcontroller; and to acquire the response of the microcontroller via serial communication.
SimSo [19] (http://projects.laas.fr/simso/)	Software	Free	To simulate real-time scheduling

3.2 The Design of the Proposed Distance Learning

First, students were introduced about Arduino platform by having few basic tasks e.g. LED blinking and logic control tasks using the Online Circuits Simulator provided by http://tinkarcad.com so that students had sufficient time to purchase or prepare an Arduino and FreeRTOS compatible microcontroller e.g. UNO and ESP32. After that, an open source real-time scheduling simulator - SimSo 0.7 [19], and the task state transitions diagram [20] were used to visualize the transition of the tasks among the ready, running, and blocked states. In SimSo simulator, worst case execution time (WCET) was used as the execution time model that was the processing load in this study. The scheduler of *simso.schedulers.FP* and *simso.schedulers.RM* were selected for fixed priority (FP) and rate monotonic (RM) scheduling simulation. One processor was used and the assigned priority, period, deadline, and processing load were key-in to the SimSo. In this simulator, the higher the priority level, the larger the priority value. Even though SimSo simulator can be used for multiprocessor real-time systems [21], uniprocessor setting was used for this introductory course. After that, the proposed Arduino-based FreeRTOS distance learning infrastructure was introduced and demonstrated to further enhance students learning experience. A combination of a simulator and a microcontroller pedagogical method is expected to achieved better outcomes than a traditional learning method as that was applied in embedded system learning [22]. Additionally, an assignment was designed to guide students to compare and analyse the outputs of SimSo simulator and the microcontroller, and to relate them to the RTES concepts of task scheduling, pre-emptive multitasking, and resource sharing.

3.3 The Implementation of the Proposed Infrastructure

A RTES consists of multiple tasks that can be defined as $T_i = (p_i, e_i, d_i)$, in which, p_i, e_i, and d_i denote the period, the processing load, and the relative deadline of *i*-th task. Fig. 1 illustrates the implementation of the proposed distance learning infrastructure that was used to enrich the learning experience of students. There were five steps, from introducing the RTES concept theoretically, simulating the system using SimSo simulator, developing the system using the de facto standard of FreeRTOS Application Programming Interface (API), acquiring the outputs of developed system, to comparing the outputs among the theory, simulated, and developed system. These five steps were repeated to introduce different RTES concepts.



1. Theoretical explanation about one RTES concept
1. $T_i = (p_i, e_i, d_i)$,
2. hyperperiod,
3. Task Scheduling without pre-emption
4. Task Scheduling with pre-emption
5. Resource Sharing: Mutex
2. SimSo simulator to simulate the system
Example outputs:
Fig. 2(a) RM scheduling
Fig. 2(b) FP scheduling
3. Arduino IDE to program the system using FreeRTOS API
4. Acquire the outputs of a microcontroller and display them using Serial Monitor
Example outputs:
Fig. 3 FP scheduling
Fig. 4 RM scheduling
Fig. 5 Mutex
5. Compare the outputs with theoretical explanation
Proposed DL infrastructure

Fig. 1 - The implementation of the proposed distance learning (DL) infrastructure for RTES course that enables each student to have first-hand experiences that related to RTES out of school

First, a RTES with three tasks of $T_1 = (10, 1, 7)$, $T_2 = (5, 1, 5)$, and $T_3 = (20, 4, 10)$ was used to study pre-emptive multitasking task scheduling i.e., temporal properties in two hyperperiods with different static priority scheduling algorithms e.g. FP and RM using the SimSo simulator. This RTES is schedulable because more than one feasible schedule are available for this system [23]. In order to produce a real-time learning experience for students, the time values used were in seconds instead of milliseconds. The serial monitor of Arduino IDE with "Show timestamp" function was used to show the real-time responses of the tasks scheduling from the microcontroller. FreeRTOS functions of *xTaskCreate*() and *vTaskDelayUntil*() were used to create the required tasks and to fix the period of each task, respectively. Since the focus was on the real-time properties, a finite loop was used as processing load in the microcontroller. Additionally, the above system was used to study resource sharing in terms of mutual exclusive (Mutex) using *xSemaphoreCreateMutex*(), *xSemaphoreGive*(), and *xSemaphoreTake*(). Based on the time values that appear on the serial monitor of Arduino IDE, Equation (1) and Equation (2) are used to estimate the period and processing time of Task *i*-th at *k* instance when no task is pre-empted. The scheduling sequence between the output of the proposed infrastructure and SimSo simulator was analysed. Each student was required to propose different system, in which, at least one of three tasks was pre-empted once. This unique requirement aims to avoid students recycle previous similar works directly. Equation 3 is used to estimate the deadline in each period in which the release time can be computed using Equation 4.

$$\text{Period of } T_i(k) = \text{Time of } T_i\ \textit{begins}(k+1) - \text{Time of } T_i\ \textit{begins}(k) \tag{1}$$

Processing time of $T_i(k)$ = Time of T_i *end*(k) - Time of T_i *begins*(k) (2)

Deadline of $T_i(k)$ = Release Time of $T_i(k)$ + Assigned relative deadline of T_i (3)

Release Time of $T_i(k)$ = Release Time of $T_i(k\text{-}1)$ + Assigned Period of T_i (4)

In this paper, Arduino NodeMCU ESP32 board (i.e., ESP-WROOM-32 module) was used as an Arduino and FreeRTOS compatible microcontroller. In the Arduino IDE (version 1.8.14), FireBeetle ESP32 mainboard (by DFRobot DFRDuino version 0.0.6) was installed and used to load the compiled code to the microcontroller. Unlike other Arduino FreeRTOS compatible microcontroller (e.g., Arduino UNO), it is unnecessary to include FreeRTOS header file to call FreeRTOS API. Since ESP32 has two processors, xTaskCreatePinnedToCore() was used to ensure all tasks were scheduled in the same processor only. Both SimSo and FreeRTOS have the same priority value and priority level relationship, i.e., the higher the priority level, the larger the priority value. Serial communication with 115200 baud rate was used to acquire the real-time response from the ESP32 board. The start delay of each task was zero as all tasks were created and released without any delay.

3.4 The Evaluation of the Learning Experience

An online survey was used to evaluate students' distance learning experiences in this study. The proposed infrastructure was evaluated in year 2021 and 2020 for session-semester of 2020/2021-2 and 2020/2021-1, respectively. A total of 42 and 25 students enrolled the course in 2020/2021-2 and 2020/2021-1, respectively. All students who enrolled the course were invited to voluntarily complete the online survey form after they completed their final exam at the end of the semester. The survey form was developed using Google Forms. In 2020/2021-1, students were required to provide their email address in the survey form; while in 2020/2021-2, all participates were anonymous. Descriptive statistics were used to analyze the received responses about the usefulness of the proposed infrastructure in terms of learners' perspectives.

4. Results and Discussion

4.1 Simulated Task Scheduling

Fig. 2 illustrates the Gantt charts that were generated using SimSo simulator when RM and FP scheduling algorithms were used. Task 3 was pre-empted when RM scheduling algorithm at time of 5s by Task 2 as Task 2 was assigned a higher priority than Task 3 (Fig. 2 (a)). On the other hand, no pre-emption happened when FP scheduling algorithm was used with the priority level of Task 3 > Task 2 > Task 1 as depicted in Fig. 2 (b). Consequently, different time delay and response time for each task were produced when different scheduling algorithms were used. Since the hyperperiod of this system is 20s, the task scheduling pattern of this system is repeated in each 20s despite different scheduling algorithms were used. This system is schedulable with a total load of 50% that could be computed using a simple test (i.e., total load equals to the summation of the ratio of processing load to the period of all tasks, i.e., 1/10 + 1/5 + 4/20). Both scheduling algorithms are expected to produce a feasible schedule for this RTES as all tasks are completed before their deadlines [23].

4.2 Task Scheduling Without Pre-Emption

Fig. 3 illustrates that the acquired outputs in the serial monitor output of Arduino IDE from the RTES in two hyperperiods when FP scheduling algorithm was used. No pre-emption was observed when FP scheduling (i.e., the priority level of Task 3 > Task 2 > Task 1) was used. The outputs that appear in real-time on the Arduino IDE serial monitor produced on the fly real-time scheduling experience as the outputs were directly acquired from the microcontroller using the USB cable and serial communication. Additionally, the time values could be used to evaluate the temporal properties of the RTES, in which, the processing load (also known as execution time) of a task can be approximated using the difference between the begin time and the end time of the task (Equation 2); and the period of a task can be approximated using the difference between the time begins and the consecutive time begins of the task if the time delay of the task is zero (Equation 1). For instance, the estimated period of Task 3 was 20s, in which, the time begins and the consecutive begin time of Task 3 were 11:56:20.344 and 11:56:40.340, respectively. On the other hand, the estimated processing load of Task 3 was 4s (i.e. the difference between 11:56:24.381 and 11:56:20.344). The expected periods and processing loads were found for the Task 1 and Task 2 using the Equation 1 and Equation 2 in any instance. These outputs and the scheduling sequence are the same as that simulated using SimSo simulator in Fig. 2 in each hyperperiod. This shows that the proposed infrastructure can produce the real-time response in a microcontroller as that simulated in SimSo simulator when no pre-emption happens. Thus, FP scheduling algorithm with a suitable priority assignment can produce a feasible schedule to ensure all tasks meet their deadlines.

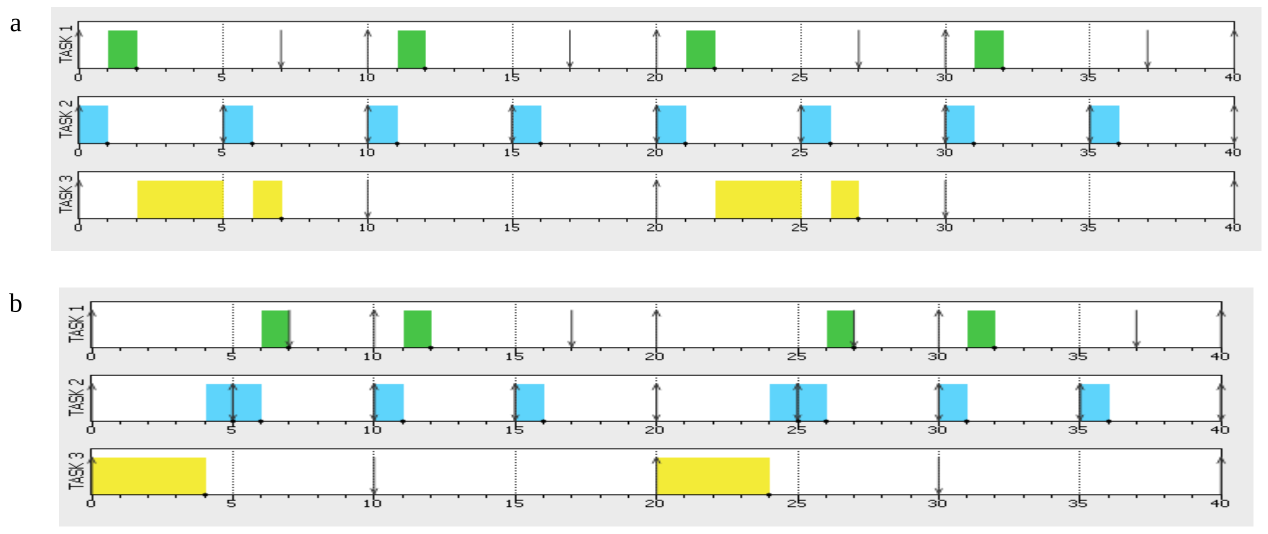
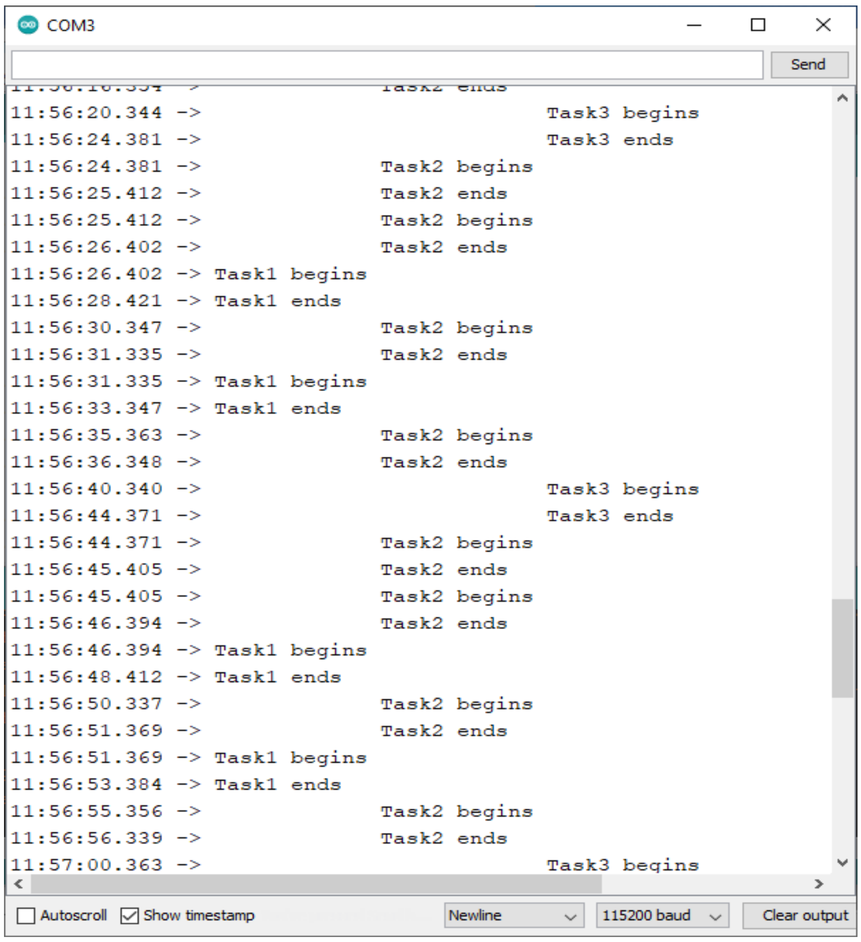


Fig. 2 - The Gantt charts from the RTES (i.e., T_1 = (10, 1, 7), T_2 = (5, 1, 5), and T_3 = (20, 4, 10)) in two hyperperiods when the scheduling algorithms were (a) Rate Motonotic Scheduling algorithm, and; (b) fixed priority scheduling (the priorities of Task 3 > Task 2 > Task 1). The green, blue, and yellow denote the processing time of Task 1, Task 2, and Task 3, respectively

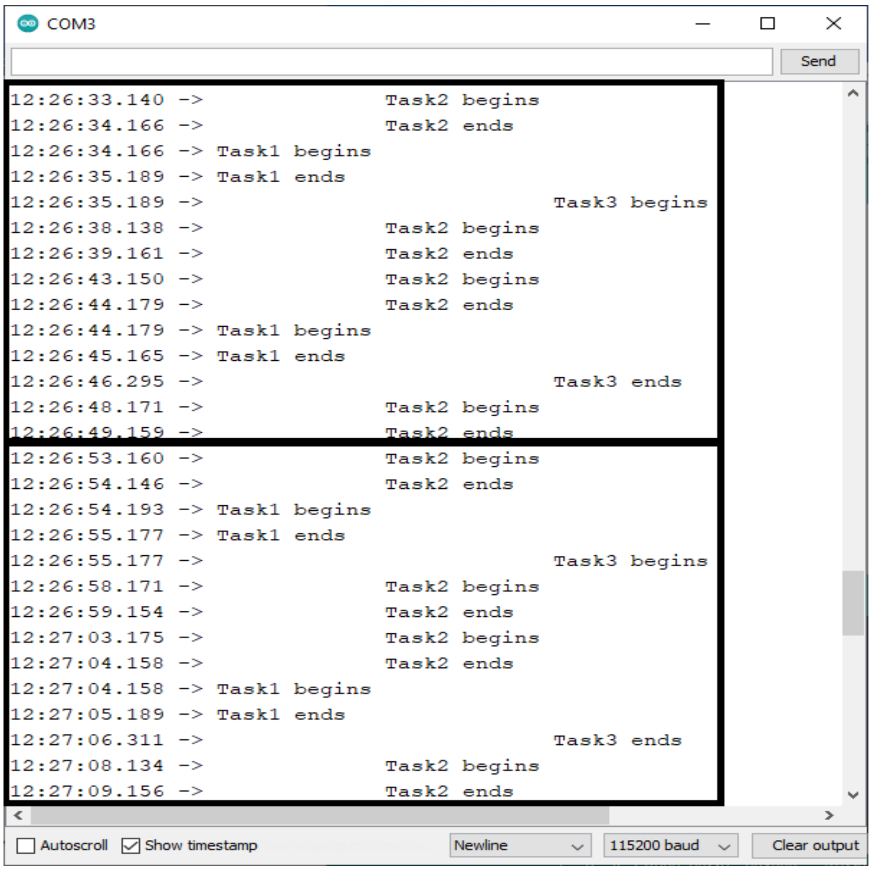


```
11:56:16.394 ->              Task2 ends
11:56:20.344 ->                           Task3 begins
11:56:24.381 ->                           Task3 ends
11:56:24.381 ->              Task2 begins
11:56:25.412 ->              Task2 ends
11:56:25.412 ->              Task2 begins
11:56:26.402 ->              Task2 ends
11:56:26.402 -> Task1 begins
11:56:28.421 -> Task1 ends
11:56:30.347 ->              Task2 begins
11:56:31.335 ->              Task2 ends
11:56:31.335 -> Task1 begins
11:56:33.347 -> Task1 ends
11:56:35.363 ->              Task2 begins
11:56:36.348 ->              Task2 ends
11:56:40.340 ->                           Task3 begins
11:56:44.371 ->                           Task3 ends
11:56:44.371 ->              Task2 begins
11:56:45.405 ->              Task2 ends
11:56:45.405 ->              Task2 begins
11:56:46.394 ->              Task2 ends
11:56:46.394 -> Task1 begins
11:56:48.412 -> Task1 ends
11:56:50.337 ->              Task2 begins
11:56:51.369 ->              Task2 ends
11:56:51.369 -> Task1 begins
11:56:53.384 -> Task1 ends
11:56:55.356 ->              Task2 begins
11:56:56.339 ->              Task2 ends
11:57:00.363 ->                           Task3 begins
```

Fig. 3 - The serial monitor output of Arduino IDE from the RTES (i.e. T_1 = (10, 1, 7), T_2 = (5, 1, 5), and T_3 = (20, 4, 10)) in two hyperperiods when FP scheduling algorithm (i.e. the priorities of Task 3 > Task 2 > Task 1) was used, where the timestamp is in hour:minute:second:millisecond format

4.3 Task Scheduling with Pre-Emption

Fig. 4 depicts the serial monitor output of Arduino IDE from the same RTES (i.e., T1 = (10, 1, 7), T2 = (5, 1, 5), and T3 = (20, 4, 10)) in two hyperperiods when RM scheduling was used. The response time of Task 3 was apparently longer compared to that simulated using SimSo simulator. This is because it was pre-empted by Task 2 twice and Task 1 once when the RTES was tested using the microcontroller. Since the only difference was the priority assignment in the xTaskCreatePinnedToCore() (or xTaskCreate()) when RM instead of FP scheduling algorithm was used, this suggests that the context switches due to pre-emption affected the response time of low priority task (i.e. Task 3 in this case). Consequently, these outputs were different compared to that simulated using SimSo simulator as the overhead of context switches are not considered in the simulator. Nevertheless, the number of each task execution was the same in each hyperperiod as that simulated using SimSo simulator; and the hyperperiod was the correct i.e., 20s that was the difference of 12:26:33.140 and 12:26:53.160, for instance. Additionally, the period of the highest priority task (i.e., Task 2 in this case) was scheduled correctly i.e., Task 2 was started at time of 12:26:33.140, 12:26:38.138, 12:26:43.150, and 12:26:48.171 with a period of approximately 5s. This indicates that FreeRTOS based RTES was able to ensure the task with highest priority to meet its the temporal requirement.



```
12:26:33.140 ->              Task2 begins
12:26:34.166 ->              Task2 ends
12:26:34.166 -> Task1 begins
12:26:35.189 -> Task1 ends
12:26:35.189 ->                           Task3 begins
12:26:38.138 ->              Task2 begins
12:26:39.161 ->              Task2 ends
12:26:43.150 ->              Task2 begins
12:26:44.179 ->              Task2 ends
12:26:44.179 -> Task1 begins
12:26:45.165 -> Task1 ends
12:26:46.295 ->                           Task3 ends
12:26:48.171 ->              Task2 begins
12:26:49.159 ->              Task2 ends
12:26:53.160 ->              Task2 begins
12:26:54.146 ->              Task2 ends
12:26:54.193 -> Task1 begins
12:26:55.177 -> Task1 ends
12:26:55.177 ->                           Task3 begins
12:26:58.171 ->              Task2 begins
12:26:59.154 ->              Task2 ends
12:27:03.175 ->              Task2 begins
12:27:04.158 ->              Task2 ends
12:27:04.158 -> Task1 begins
12:27:05.189 -> Task1 ends
12:27:06.311 ->                           Task3 ends
12:27:08.134 ->              Task2 begins
12:27:09.156 ->              Task2 ends
```

☐ Autoscroll ☑ Show timestamp Newline 115200 baud Clear output

Fig. 4 - The serial monitor output of Arduino IDE from the RTES (i.e. T_1 = (10, 1, 7), T_2 = (5, 1, 5), and T_3 = (20, 4, 10)) in two hyperperiods when RM scheduling was used – a black rectangular line is used to indicate a hyperperiod of the RTES

Table 2 tabulates the transition state and time interval of the RTES system. Task 3 was pre-empted by Task 2 twice and Task 1 once. Consequently, the response time of Task 3 was 11.106s that was longer than that simulated by SimSo simulator as the overhead of context switch was not considered. This is in agreement with a recent work that reported the temporal analysis of FreeRTOS context switch overheads [24]. Particularly, task switching (i.e. vTaskSwitchContext that has three steps of store, select, and load process) is the greatest source of time variation in the FreeRTOS context switch algorithm [24]. This indicates that there is a need to avoid unnecessary pre-emption in a periodic system to avoid unnecessary context switch overhead. In other words, FP scheduling algorithm is better than RM scheduling algorithm in this RTES. Nevertheless, the periods of all tasks (i.e., Task 1, Task 2, and Task 3) and the execution time of both Task 1 and Task 2 were approximately same as that assigned. In other words, the overheads of the pre-emption context switch significantly affected the processing load of the lowest priority task of Task 3. Since the response time of Task 3 was 13s (i.e., 12:26:46.295 – 12:26:33.140) that more than its deadline of 10s, RM scheduling algorithm is not feasible when it was applied to the RTES system using the proposed infrastructure.

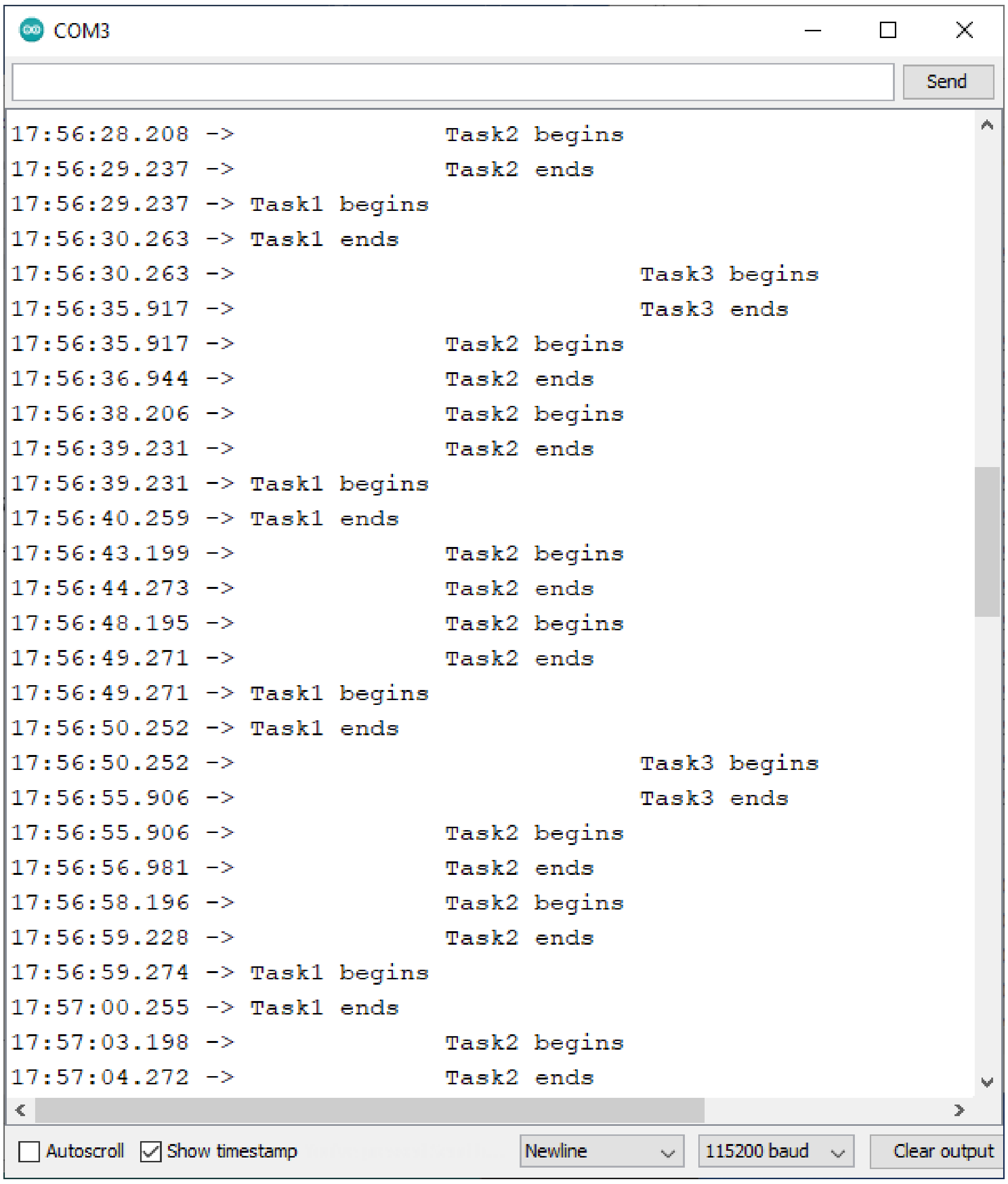
Table 2 - The transition state and time interval of the RTES (i.e., T_1 = (10, 1, 7), T_2 = (5, 1, 5), and T_3 = (20, 4, 10)) when RM scheduling was used

Time (seconds)	Activity	Comment	Time interval	State		
				Task 1	Task 2	Task 3
35.189	Task3 begins	Task 3 is executed.	0	Block	Block	Running
38.138	Task2 begins	Task 3 is pre-empted by Task 2	2.929s	Block	Running	Ready
39.161	Task2 ends	Task 2 is completed and Task 3 is restored.	1.023s	Block	Block	Running
43.150	Task2 begins	Task 3 is pre-empted by Task 2	3.989s	Block	Running	Ready
44.179	Task2 ends Task1 begins	Task 2 is completed. Task 1 is executed immediately. Task 3 is pre-empted by Task 1.	1.029s	Running	Block	Ready
45.165	Task1 ends	Task 1 is completed and Task 3 is restored.	0.986s	Block	Block	Running
46.295	Task3 ends	Task 3 is completed	1.130s	Block	Block	Block
		Total processing time	11.106s	0.986s	2.052s (1.023 + 1.029)	8.048s (2.929 + 3.989 + 1.130)

4.4 Resource Sharing: Mutex

RTES concept of resource sharing is unable to be demonstrated by SimSo simulator. This shows the feasibility of the proposed infrastructure to introduce more RTES topics as a real RTES is used. In this study, a mutual exclusive (Mutex) function was used to avoid Task 3 to be pre-empted by Task 2 when RM scheduling algorithm was used. This could minimize the number of context switch as Task 2 would be in block state until Task 3 was completed, and vice versa, as that illustrated in Fig. 5. Results show that the hyperperiod was the same i.e., 20s from time 17:56:28.208 to 17:56:48.195. Next, due to the use of Mutex that prevents the pre-emption on Task 3, the period of Task 3 was the same, i.e., 20s from time 17:56:30.263 to 17:56:50.252. Nevertheless, the processing load of Task 3 was increased from 4s to 5.65s (i.e., from time 17:56:30.263 to 17:56:35.917). This suggests that the context switch overhead was around 1.65s. This context switch overhead was high because it was one pre-emption that happened on Task 3. Task 3 was pre-empted by Task 2 once when Task 2 was released. Since the Mutex was hold by Task 3, Task 2 went to Block state immediately until Mutex was released by Task 3. Consequently, Task 3 was able to be completed without any further pre-emption as Task 2 could only back to Ready state when Task 3 was completed. The operation was unable to be observed in Fig. 5 as each task only sent a signal when it was started or completed. Nevertheless, by analysing this temporal information, the task scheduling mechanism of a RTES can be deduced.

Since all tasks (i.e., T_1 = (10, 1, 7), T_2 = (5, 1, 5), and T_3 = (20, 4, 10)) were released at the same time in each hyperperiod, all tasks met their relative deadline of 7, 5, and 10s, respectively. For time release of a hyperperiod of 17:56:28.208, the Task 1 deadline were 17:56:35.208 and 17:56:45.208; the Task 2 deadline were 17:56:33.208, 17:56:38.208, 17:56:43.208, 17:56:48.208, and 17:56:53.208; and the Task 3 deadline was 17:56:38.208. These deadlines were calculated using Equation 3. These results show that RM scheduling algorithm coupled with Mutex in minimizing the number of pre-emption is feasible in this RTES as all tasks were completed before their deadlines.



```
17:56:28.208 ->              Task2 begins
17:56:29.237 ->              Task2 ends
17:56:29.237 -> Task1 begins
17:56:30.263 -> Task1 ends
17:56:30.263 ->                           Task3 begins
17:56:35.917 ->                           Task3 ends
17:56:35.917 ->              Task2 begins
17:56:36.944 ->              Task2 ends
17:56:38.206 ->              Task2 begins
17:56:39.231 ->              Task2 ends
17:56:39.231 -> Task1 begins
17:56:40.259 -> Task1 ends
17:56:43.199 ->              Task2 begins
17:56:44.273 ->              Task2 ends
17:56:48.195 ->              Task2 begins
17:56:49.271 ->              Task2 ends
17:56:49.271 -> Task1 begins
17:56:50.252 -> Task1 ends
17:56:50.252 ->                           Task3 begins
17:56:55.906 ->                           Task3 ends
17:56:55.906 ->              Task2 begins
17:56:56.981 ->              Task2 ends
17:56:58.196 ->              Task2 begins
17:56:59.228 ->              Task2 ends
17:56:59.274 -> Task1 begins
17:57:00.255 -> Task1 ends
17:57:03.198 ->              Task2 begins
17:57:04.272 ->              Task2 ends
```

Fig. 5 - The serial monitor output of Arduino IDE from the RTES (i.e. T_1 = (10, 1, 7), T_2 = (5, 1, 5), and T_3 = (20, 4, 10)) in two hyperperiods when RM scheduling and Mutex were used

4.5 Students Feedbacks About Their Learning Experience

In 2020/2021-2, all 42 students were invited to voluntarily provide anonymous feedbacks that were collected and analysed to evaluate the effectiveness of the proposed learning and teaching method. Table 3 tabulates a descriptive analysis of the received feedback. A total of nine responses were received from the 42 invited students (response rate = 21%). A low response rate indicates that the students were not interested in taking survey as they might be overwhelmed by various online surveys and activities during the semester. Nevertheless, this response rate is within a typical online survey response rates of 5 to 30% [25]. A better way to increase the number of responses shall be studied e.g., conducting the survey before their final exam or at the end of the last lecture session. With nine responses (i.e., sample size) from 42 students who enrolled the course (i.e., population size) and the response distribution of 50%, the confidence interval (also known as margin of error) would be 29.31% with a confidence level of 95%. The margin of error was calculated using the sample size calculator at http://www.raosoft.com/samplesize.html. Table 3 shows that 100% of the responses agreed that the use of Arduino platform and FreeRTOS was useful (or very useful) as learning tools or methods in helping them understand the RTES. The score of question 1 (i.e., Assignment with SimSo) was slightly higher than question 2 and 3 (i.e. Assignment with Arduino Platform and FreeRTOS) could be due to the graphical user interface (GUI) platform that is relatively easier to be used and understood compared to the use of C programming, FreeRTOS, and serial communication to display the real-time output. Nevertheless, the same score was achieved to question 4 (i.e., Demo with Arduino Platform) that was about the teaching method that used the proposed infrastructure to demonstrate RTES key concepts. These results are in agreement with another survey that was conducted in 2020/2021-1. A total of 14 responses were received from the 25 invited students (response rate = 56%). Three incomplete responses were excluded in the analysis in Table 4. A relatively more responses were received might be due to the survey form design that required students to provide their email address in 2020/2021-1 instead of anonymous that did 2020/2021-2. With 11 responses from 25 students who enrolled the course and the response distribution of 50%, the confidence interval would be 22.57% with a confidence level of 95%. Slightly higher mapped mean value in 2020/2021-1 might be due to a larger scale of 1-5 that provided a "Neutral" option that contributed a mapped score of 1.8; while "Agree" and "Very agree" options denote mapped scores of 2.4 and 3, respectively. Nevertheless, the proposed learning infrastructure received positive responses in two consecutive semesters, i.e., the "mean – confidence interval" score of each survey question was more than 1.5 with a confidence level of 95% as that tabulated in Table 3 and Table 4. This indicates that the use of proposed learning infrastructure was useful in helping students understand RTES complex concepts.

Table 3 - Descriptive analysis of 9 responses from 42 students who were invited voluntarily participated the survey after they completed the RTES course in 2020/2021-2 with a score of 1 (No useful), 2 (Useful), and 3 (Very useful)

No	Question: Rate the usefulness of the following learning tools/methods in helping understand the Real-Time Embedded System	Score (1-3)			
		Min	Max	Mean ± CI	Standard deviation
1	Assignment with SimSo	2	3	2.56 ± 0.75	0.50
2	Assignment with Arduino Platform	2	3	2.44 ± 0.72	0.50
3	FreeRTOS	2	3	2.33 ± 0.68	0.47
4	Demo with Arduino Platform	2	3	2.56 ± 0.75	0.50

Note: CI denotes confidence interval.

Table 4 - Descriptive analysis of 11 responses that received from 25 students who were invited voluntarily participated the survey after they completed the RTES course in 2020/2021-1 with a score of 1 (Strongly disagree), 2 (Disagree), 3 (Neutral), 4 (Agree) and 5 (Strongly disagree)

No	Question:	Score (1-5)				Mapping to scale (1-3)			
		Min	Max	Mean	Standard deviation	Min	Max	Mean ± CI	Standard deviation
1	SimSo is useful to understand RTES	3	5	4.45	0.87	1.8	3	2.67 ± 0.60	0.49
2	FreeRtos is useful to understand RTES	3	5	4.36	0.83	1.8	3	2.62 ± 0.59	0.49
3	Arduino is useful to understand RTES	3	5	4.18	1.05	1.8	3	2.51 ± 0.57	0.59

Note: CI denotes confidence interval.

5. Conclusion

This study evaluated an alternative way to integrate existing open-source hardware and software platforms that enabled students to have a deeper understanding about RTES when they were required to learn from home. Several temporal analyses were conducted to compare the proposed infrastructure and the simulator. Particularly, the potentials of the existing open-source software and hardware (i.e. an Arduino and FreeRTOS compatible microcontroller and Arduino IDE) were explored to avoid redundant facilities. This is crucial during pandemic period to minimize unnecessary financial burden to learners. By conducting a temporal analysis of the proposed Arduino based FreeRTOS infrastructure, students would have a real-time experience about key RTES concepts, particularly, how RTES guarantees the task with the highest priority to fulfil its timing requirement. The proposed infrastructure can show that the sequence of scheduled tasks in a hyperperiod was the same as that simulated in SimSo simulator when pre-emption did not occur.

When pre-emption happened, on the other hand, the task with lower priority had longer response time due to context switches could be observed using Arduino IDE serial monitor. As a result, the proposed infrastructure produced an output that is different with that simulated in SimSo simulator. Initially, this was confusing for the students. However, eventually, this led students to further understand the abstract concept of context switch overhead. Thus, the optimal algorithm of the RTES that used in this study was FP scheduling algorithm with a suitable priority assignment that avoided unnecessary pre-emption to prevent avoidable context switches when Arduino ESP32 was tested. Additionally, the proposed integration is able to demonstrate complex RTES concept of Mutex as a real RTES is used to show the real-time response. Since positive responses were received in two consecutive semesters, the proposed infrastructure is an affordable and accessible distance learning alternative in assisting students to understand RTES concepts of hyperperiod, task scheduling, fixed priority scheduling, context switch, pre-emption, and Mutex. Nevertheless, a way to teach other RTES complex topics e.g., dynamic scheduling strategy of Earliest Deadline First and Counting Semaphore shall be developed and evaluated using the proposed infrastructure in the near future.

Acknowledgement

Author would like to acknowledge Faculty of Electrical and Electronic Engineering, Universiti Tun Hussein Onn Malaysia for the facilities supports; and students who voluntarily completed the survey.

References

[1] M. El-Abd, “A Review of Embedded Systems Education in the Arduino Age: Lessons Learned and Future Directions,” *International Journal of Engineering Pedagogy*, vol. 7, no. 2, pp. 79–93, May 2017, [Online]. Available: https://www.learntechlib.org/p/207404.

[2] C. Carlson, G. Peterson, and D. Day, “Utilizing Portable Learning Technologies to Improve Student Engagement and Retention,” *IEEE Transactions on Education*, vol. 63, no. 1, pp. 32–38, 2020, doi: 10.1109/TE.2019.2941700.

[3] K. Küçük, C. Bayılmış, and D. L. Msongaleli, “Designing real-time IoT system course: Prototyping with cloud platforms, laboratory experiments and term project,” *The International Journal of Electrical Engineering & Education*, pp. 1–14, Jul. 2019, doi: 10.1177/0020720919862496.

[4] M. F. Roslan, “Internet of Things (IoT)-based Solution for Real-time Monitoring System in High Jump Sport,” *International Journal of Integrated Engineering*, vol. 11, no. 8 SE-Articles, Dec. 2019, [Online]. Available: https://publisher.uthm.edu.my/ojs/index.php/ijie/article/view/3331.

[5] M. Short and C. Cox, “RTE-SIM: A Simple, Low-Cost and Flexible Environment to Support the Teaching of Real-Time and Embedded Control,” *The International Journal of Electrical Engineering & Education*, vol. 48, no. 4, pp. 339–358, 2011, doi: 10.7227/IJEEE.48.4.1.

[6] P. Jamieson, “Arduino for teaching embedded systems. are computer scientists and engineering educators missing the boat?,” *Proc. FECS*, pp. 289–294, 2011, [Online]. Available: http://www.worldcomp-proceedings.com/proc/p2011/FEC3377.pdf.

[7] Y. H. Hee, M. K. Ishak, M. S. Mohd Asaari, and M. T. Abu Seman, “Embedded operating system and industrial applications: a review,” *Bulletin of Electrical Engineering and Informatics*, vol. 10, no. 3, pp. 1687–1700, Jun. 2021, doi: 10.11591/eei.v10i3.2526.

[8] E. A. Shammar and A. T. Zahary, “The Internet of Things (IoT): a survey of techniques, operating systems, and trends,” *Library Hi Tech*, vol. 38, no. 1, pp. 5–66, Jan. 2020, doi: 10.1108/LHT-12-2018-0200.

[9] S. Rounaq and M. Iqbal, “Vision, Challenges and Future Perspectives of Low Constrained Devices IOT Operating Systems: A Systematic Mapping Review,” *European Journal of Engineering and Technology Research*, vol. 5, no. 12, pp. 107–115, 2020.

[10] D. Mocrii, Y. Chen, and P. Musilek, “IoT-based smart homes: A review of system architecture, software, communications, privacy and security,” *Internet of Things*, vol. 1–2, pp. 81–98, 2018, doi: https://doi.org/10.1016/j.iot.2018.08.009.

[11] W. XU, E. PAN, J. LIU, Y. LI, and H. YUAN, “Flight control of a large-scale flapping-wing flying robotic bird: System development and flight experiment,” *Chinese Journal of Aeronautics*, 2021, doi: https://doi.org/10.1016/j.cja.2021.03.009.

[12] Z. ZHENG and G. XIAO, “Evolution analysis of a UAV real-time operating system from a network perspective,” *Chinese Journal of Aeronautics*, vol. 32, no. 1, pp. 176–185, 2019, doi: https://doi.org/10.1016/j.cja.2018.04.011.

[13] X. Guo, “Design of motor control driver based on arm and freertos,” *IET Conference Proceedings*, pp. 897-902(5), Jan. 2021, [Online]. Available: https://digital-library.theiet.org/content/conferences/10.1049/icp.2021.0363.

[14] N. He and H. W. Huang, “Use of FreeRTOS in teaching a real-time embedded systems design course,” *Computers in Education Journal*, vol. 5, no. 4, pp. 18–25, 2014, doi: 10.18260/1-2--23240.

[15] C. Galindo and J. A. Fernandez-Madrigal, “Grounding Concepts and Methods of Real-Time Scheduling in Reality Using Arduino,” *IEEE Transactions on Education*, vol. 63, no. 3, pp. 224–231, 2020, doi: 10.1109/TE.2020.2975352.

[16] A. Qazi *et al.*, “Adaption of distance learning to continue the academic year amid COVID-19 lockdown,” *Children*

and Youth Services Review, vol. 126, p. 106038, 2021, doi: https://doi.org/10.1016/j.childyouth.2021.106038.

[17] M. Selvanathan, N. A. M. Hussin, and N. A. N. Azazi, "Students learning experiences during COVID-19: Work from home period in Malaysian Higher Learning Institutions," *Teaching Public Administration*, p. 0144739420977900, Dec. 2020, doi: 10.1177/0144739420977900.

[18] B. H. Sababha, Y. A. Alqudah, A. Abualbasal, and E. A. Q. Al, "Project-based learning to enhance teaching embedded systems," *Eurasia Journal of Mathematics, Science and Technology Education*, vol. 12, no. 9, pp. 2575–2585, 2016, doi: 10.12973/eurasia.2016.1267a.

[19] M. Chéramy, P.-E. Hladik, and A.-M. Déplanche, "Simso: A simulation tool to evaluate real-time multiprocessor scheduling algorithms," in *5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2014, pp. 6--p.

[20] R. Barry, *Mastering the FreeRTOS ™ Real Time Kernel*. 2016.

[21] F. M. S. Nascimento and G. Lima, "Effectively Scheduling Hard and Soft Real-Time Tasks on Multiprocessors," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021, pp. 210–222, doi: 10.1109/RTAS52030.2021.00025.

[22] Y. Chu and J. H. Park, "Efficient learning modules for embedded system," *The International Journal of Electrical Engineering & Education*, vol. April, 2020, doi: 10.1177/0020720920918153.

[23] J. Wang, "Task Scheduling," in *Real-Time Embedded Systems*, John Wiley & Sons, Ltd, 2017, pp. 53–98.

[24] B. D. Miranda, R. S. de Oliveira, and A. Carminati, "Analysis of FreeRTOS Overheads on Periodic Tasks," in *Anais do XX Workshop em Desempenho de Sistemas Computacionais e de Comunicação*, 2021, pp. 119–130, doi: 10.5753/wperformance.2021.15728.

[25] J. Braithwaite, C. Mayuga, and C. Fraser, "Why they come, what they learn, how they change: Measuring the effectiveness of health and safety training - BESAFE REPORT FOR AKO AOTEAROA," Ako Aotearoa National Centre for Tertiary Teaching Excellence, 2020.