An Energy-Efficient Embedded System Platform for Energy-Critical Real-Time Tasks

Jun Wu *Member, IAENG,* and Jia-Lin Wang

***Abstract*—This paper presents an energy-efficient embedded system platform, called meCreate, to execute energy-critical real-time tasks. In meCreate, we consider a task has multiple versions for different energy criticality levels. At run time, the system will operate at one of the energy criticality levels dynamically according to the remaining capacity of battery. In particular, meCreate will operate at a lower energy criticality level and the corresponding version of each task will be executed when the battery capacity is plentiful. A version for a lower energy criticality level performs relatively complex operations to generate results more accurately, and/or it can shorten the period to obtain results more frequently. When the remaining capacity of battery become insufficient, the system will switch to a higher energy criticality level and the corresponding version of each task will be executed. A version for a higher energy criticality level performs simplified operations and/or a longer period in order to save more energy. We also present an example application, called ecoScout, to demonstrate how to build application upon meCreate. The energy efficiency and the performance were evaluated by a set of experiments based on the example application and randomly generated task sets, for which we have some encouraging results.**

***Index Terms*—Energy criticality, multi-version tasks, embedded system platform, and real-time systems.**

I. Introduction

NOWADAYS, energy efficiency has become one of the most important issues for embedded real-time systems (ERTS) since many of them are powered by batteries. Note that it is also critical for an ERTS powered by mains because energy-efficient design can help to cut down bills as well as to lower down the carbon emissions. In the literature, there are many excellent approaches have been done (comprehensive surveys can be found in [1], [2]) by using voltage scaling techniques. Different from the past work, this paper proposes an innovative task model, called *energy-critical multi-version real-time task model*, which is motivated by Vestal's task model for mixed criticality systems (MCS)[3]. This model considers a set of energy criticality levels $\mathcal{ECL} = \{ECL_1, ECL_2, \cdots, ECL_M\}$, where ECL_1 and ECL_M are the levels for the system with plentiful and insufficient energy, respectively. It assumes that every task τ_i in the system has multiple versions $\tau_i^1, \tau_i^2, \cdots, \tau_i^M$ for every energy criticality level in $\mathcal{ECL}$.

At run time, the system will operate at one of the energy criticality levels $ECL_x \in \mathcal{ECL}$ according to the remaining capacity of battery. When the system is operating at level ECL_x, the corresponding version τ_i^x of a task τ_i will be selected and be executed. Note that the different versions of a task can be designed for obtaining the different degrees of accuracy. In particular, a version for a lower energy criticality level performs relatively complex operations to generate more accurate results. On the contrary, a higher level's version is designed to perform simplified operations to generate less accurate results with less energy consumption.

Based on our proposed task model, an ERTS platform, called *mixed energy-criticality real-time embedded platform* (meCreate), is implemented for energy-constrained embedded real-time systems. Developers can build an application upon meCreate by configuring the system and providing a set of tasks (and their versions for multiple energy criticality levels). In this paper, we also provided an example application, called *ecoScout*, upon meCreate to show the convenience of the development. A set of experiments has been conducted such that the energy efficiency and the performance can be better understood. Note that a previous version of this paper was presented at the 7th IEEE ICASI[4]. This results have been extended in providing more details about the system model and the implementation. Furthermore, more figures and experimental results are also included in this extension.

The paper is structured as follows: Section II summarizes the related work of design tools and platforms for ERTS. Section III defines the system model for energy-constrained embedded real-time systems and our proposed task model. Section IV provides meCreate's architecture and describes how it works. Section V presents an example application, i.e., ecoScout. Section VI presents the experimental results. Section VII provides the conclusion.

II. Related Work

In recent decades, many design tools and platforms have been proposed for ERTS (e.g., [5], [6], [7], [8], [9], [10], [11], [12]). Some of them can be considered as tools for simplifying the design of embedded systems. For instance, Hsu [6] has proposed a visual development tool, called Builder for Embedded SofTware (BEST), such that developers can build embedded applications in a WYSIWYG manner.

Since the popularity of Linux is continuously growing in the field of embedded systems, other approaches (such as [8], [9], [10]) provide a configurable way to build custom Linux operating systems for different hardware architectures so that the time to market can be reduced greatly. Based on those approaches, many applications (e.g., [13], [14], [15]) are capable to be executed on the target system with a custom Linux operating system.

Although many existing approaches are proposed for Linux-based embedded systems, relatively little work is done

Manuscript received March 18, 2022; revised November 6, 2022. This work was supported in part by the National Science and Technology Council of Taiwan under grants 109-2221-E-153-002 and 110-2221-E-153-001-MY3.

Jun Wu is a professor of Department of Computer Science and Information Engineering, National Pingtung University, 900 Pingtung, Taiwan, ROC. (email: junwu@mail.nptu.edu.tw).

Jia-Lin Wang is a graduate student of Department of Computer Science and Information Engineering, National Pingtung University, 900 Pingtung, Taiwan, ROC. (email: garlic8668@gmail.com).

for ERTS[16]. Different from traditional embedded systems, tasks in an ERTS have strict timing constraints. Specifically, the completion time of an ERTS's task must no later than its deadline. Nowadays, there are more and more ERTS have been built to provide the worst-case performance guarantees and to meet tasks' deadlines (e.g., [17], [18], [19]). However, most of them were built without the help from development tools or system platforms. Furthermore, to the best of our knowledge, there is no approach has been done for developing energy-constrained ERTS with the support of our proposed task model.

III. Models

In this section, the system model and energy-critical multi-version real-time task model are presented.

A. System Model

We consider an energy-constrained ERTS has M energy criticality levels $\mathcal{ECL}= \{ECL_1, ECL_2, \cdots, ECL_M\}$. At the run time, the system will operate at one of the energy criticality levels $ECL_j \in \mathcal{ECL}$ according to the remaining capacity of battery. In particular, we assume the system has a set of switching threshold $\mathcal{SWT} = \{SWT_1, SWT_2, \cdots, SWT_M, SWT_{M+1} = 0\}$, where $0 \leq SWT_i \leq 1$ and $SWT_i > SWT_{i+1}$ for $1 \leq i \leq M$. The value of SWT_i is the switching threshold for energy criticality level ECL_i. Let IC and RC be the initial and the remaining capacity of battery. The system will set to be operating at level ECL_i if $SWT_i \geq \frac{RC}{IC} > SWT_{i+1}$, for $1 \leq i < M$. The processor speed is assumed to be varied by the supply voltage, i.e., it can operating at K different speeds $\mathcal{S} = \{s_{min} = s_1, s_2, \cdots, s_K = s_{max}\}$, where s_{min} and s_{max} are the minimum and the maximum processor speed, respectively.

B. Task Model

In this paper, a new task model, called *energy-critical multi-version real-time task model*, is proposed based on the well-known periodic real-time task model[20] and Vestal's task model[3]. Specifically, we consider an application is consists of n periodic real-time tasks $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_n\}$. A task τ_i is defined by its arrival time A_i, periodic $\vec{T_i}$, worst-case computation time $\vec{C_i}$, and relative deadline $\vec{D_i}$, where $\vec{T_i}$, $\vec{C_i}$ and $\vec{D_i}$ are vectors of values for each energy criticality level.

In order to obtain better energy efficiency, each task τ_i is assumed to have multiple versions $\tau_i^1, \tau_i^2, \cdots, \tau_i^M$ for each energy criticality level. The period, worst-case computation time, and relative deadline of τ_i for an energy criticality level ECL_x are defined by $T_i(ECL_x)$, $C_i(ECL_x)$, and $D_i(ECL_x)$, respectively. Note that the different versions of a task reflect the tradeoff between the accuracy of the execution results and the energy consumption for obtaining the results. In order words, the execution results of τ_i^x is more accurate to that of τ_i^y and the energy consumption of τ_i^x is higher than that of τ_i^y if $x < y$. These vectors are given with the following constraints:

$$
\begin{aligned}
x > y \Rightarrow \quad & T_i(ECL_x) \geq T_i(ECL_y) \\
& C_i(ECL_x) \leq C_i(ECL_y) \\
& D_i(ECL_x) \geq D_i(ECL_y)
\end{aligned}
$$

Let ECL^* be the current energy criticality level in the system. Each periodic real-time task τ_i will create the first task instance $\tau_{i,1}$ at its arrival time A_i, and then it create an instance $\tau_{i,j}$ (for $j > 1$) regularly for every period of time $T_i(ECL^*)$. Whenever an instance $\tau_{i,j}$ arrived to the system, it must be executed on the processor for no more than $\frac{C_i(ECL^*)}{s_x}$, where s_x is the processor speed. Note that the values of $\vec{C_i}$ are determined by assuming the processor is operating at s_{max}. Moreover, the following conditions are satisfied:

$$
0 \leq C_i(ECL_x) \leq D_i(ECL_x) \leq T_i(ECL_x),
$$
$$
\forall \tau_i \in \mathcal{T} \text{and } \forall ECL_x \in \mathcal{ECL}.
$$

The current energy criticality level ECL^* is determined according to the current remaining capacity of battery RC and the switching threshold. In particular, the system will be operating at level $ECL^* = ECL_x$ if $SWT_x \geq \frac{RC}{IC} > SWT_{x+1}$. Note that the corresponding version τ_i^x will be selected to be executed when the system is operating at level ECL_x. Consider a temperature sensing task τ_{ts} which is required to be executed in an environmental monitoring system. The system has two energy criticality levels ECL_1 and ECL_2 with the switching threshold $SWT_1 = 1$ and $SWT_2 = 0.5$. Initially, the system starts with plentiful energy (i.e., RC=IC), therefore, the current energy criticality level ECL^* is set to ECL_1 (since $SWT_1 = 1 \geq \frac{RC}{IC} > SWT_2 = 0.5$) and the corresponding version τ_{ts}^1 of τ_{ts} is selected to be executed. Note that τ_{ts}^1 is designed for generating results with higher accuracy. It will gather environmental temperature and return the average of the last 10 gathered temperatures to a backend server. After some time of operation, ECL^* is set to ECL_2 when the remaining capacity of battery becomes insufficient, i.e., $SWT_2 = 0.5 \geq \frac{RC}{IC} > 0$. Therefore, the corresponding version for a higher energy criticality level, i.e., τ_{ts}^2, will be selected. Compared to that of τ_{ts}^1, τ_{ts}^2 performs relatively simplified operations (e.g., it returns the last gathered temperature only) such that the energy consumption is reduced.

The major goal of the scheduling problem is to satisfy tasks' timing constraints (i.e., all task instances have to meet their deadlines). However, our proposed energy-critical multi-version real-time task model is designed to reduce energy consumption while the necessarily accuracy is obtained. Therefore, the following three issues also have to be considered at the run time:

1) switch the energy criticality level ECL^* according to the remaining capacity of battery;
2) select and execute the corresponding task version τ_i^* for ECL^* for every task τ_i;
3) calculate and assign a proper speed for task execution.

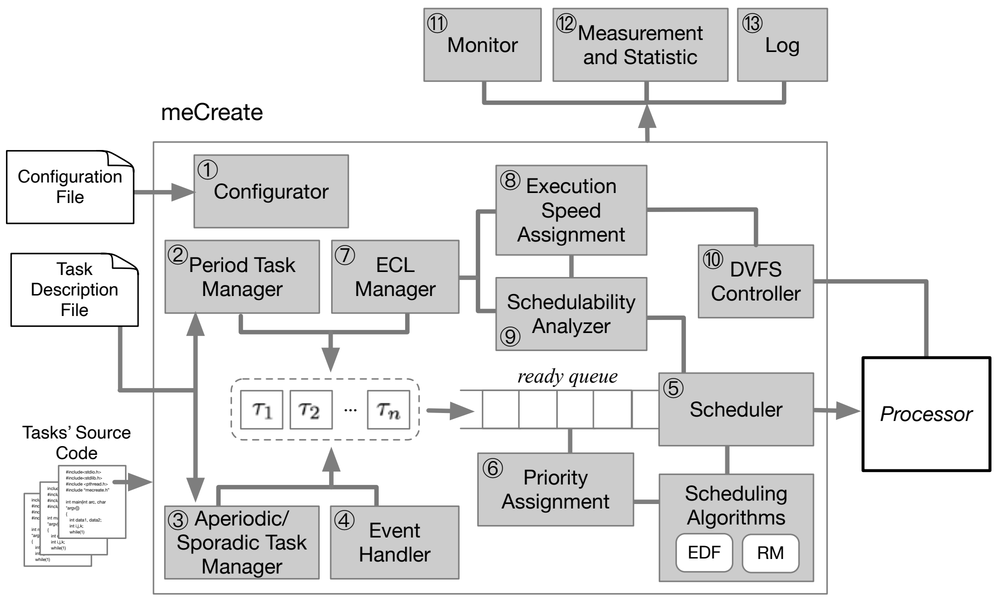


Fig. 1. The architecture of meCreate.

IV. Our proposed energy-constrained ERTS platform

In this section, an ERTS platform, called mixed energy-criticality real-time embedded (meCreate), is proposed to support our proposed energy-critical multi-version real-time task model. meCreate is built based on Linux (kernel 5.10.17) and supports Raspberry Pi 4B presently (the support of more embedded boards is working in progress). By using meCreate, applications for energy-constrained ERTS can be built and deployed conveniently and easily. In order to develop an application upon meCreate, only the following files are needed:

- Configuration file: It is an XML file used to define the options and settings of the target system. Such as the target board (it only supports Raspberry Pi 4 family currently), network settings, energy criticality levels and the switching threshold, task preemptibility, and the scheduling algorithm. Note that meCreate supports two well-known scheduling algorithms: earliest deadline first (EDF)[20] and rate-monotonic (RM)[20] scheduling algorithm.
- Task description file: It is also an XML file for the definition of the task set. Currently, periodic, aperiodic, and sporadic tasks are supported. For a periodic task, we have to define its relative parameters in the description file. The relative parameters including task's arrival time, deadline, period, and computation time. Since each task has multiple versions, the relative parameters of the versions are defined by vectors. For aperiodic and sporadic tasks, the parameters are the same as that of the periodic tasks excepts the period is replaced by the trigger event and the minimum separation time, respectively.
- Tasks' source code: The source code of each task and their multiple versions (corresponding to the energy-critical levels) have to be provided by developers. Note that meCreate supports C and Python programming languages currently.

Fig. 1 shows the architecture and the data/control flows of meCreate. As shown in the figure, it has several modules and tools to help developers to build applications upon meCreate. After developers provided the above files, the Configurator module (labelled by ①) is responsible for setting the target system such that the system can be operated properly according to the content of the configuration file. The Periodic Task Manager module (labelled by ②) will generate the task instances according according to the arrival times and periods defined in the task description file. The Aperiodic/Sporadic Task Manager module (labelled by ③) is cooperative with the Event Handler module (labelled by ④) such that the task instances of aperiodic and sporadic tasks can be generated according to their trigger events and the minimum separation times, respectively.

All generated task instances are scheduled by the Scheduler module (labelled by ⑤) according to the specific task scheduling algorithm. Since both of the supported algorithms (i.e., RM and EDF) are priority-driven scheduling algorithms, the Scheduler module is cooperative with the Priority Assignment module (labelled by ⑥). Note that our implementation of the Priority Assignment module supports both fixed-priority and dynamic-priority scheduling algorithms.

The ECL Manager module (labelled by ⑦) sets ECL^* dynamically by the given switching thresholds. Whenever ECL^* has been switched to a lower energy criticality level, the Execution Speed Assignment module (labelled by ⑧) will cooperative with the Schedulability Analyzer (labelled by ⑨) to calculate tasks' execution speed $s^{'}$. The DVFS Controller module (labelled by ⑩) will set the corresponding execution frequency of the processor after $s^{'}$ is calculated. The calculation of $s^{'}$ for RM and EDF scheduling algorithms are as follows:

- RM:

$$s^{'} = \min_{s_j \in \mathcal{S}} \{ \sum_{\tau_i \in \mathcal{T}} \frac{C_i(ECL^*)/s_j}{T_i(ECL^*)} \leq n(2^{1/n} - 1) \} \quad (1)$$

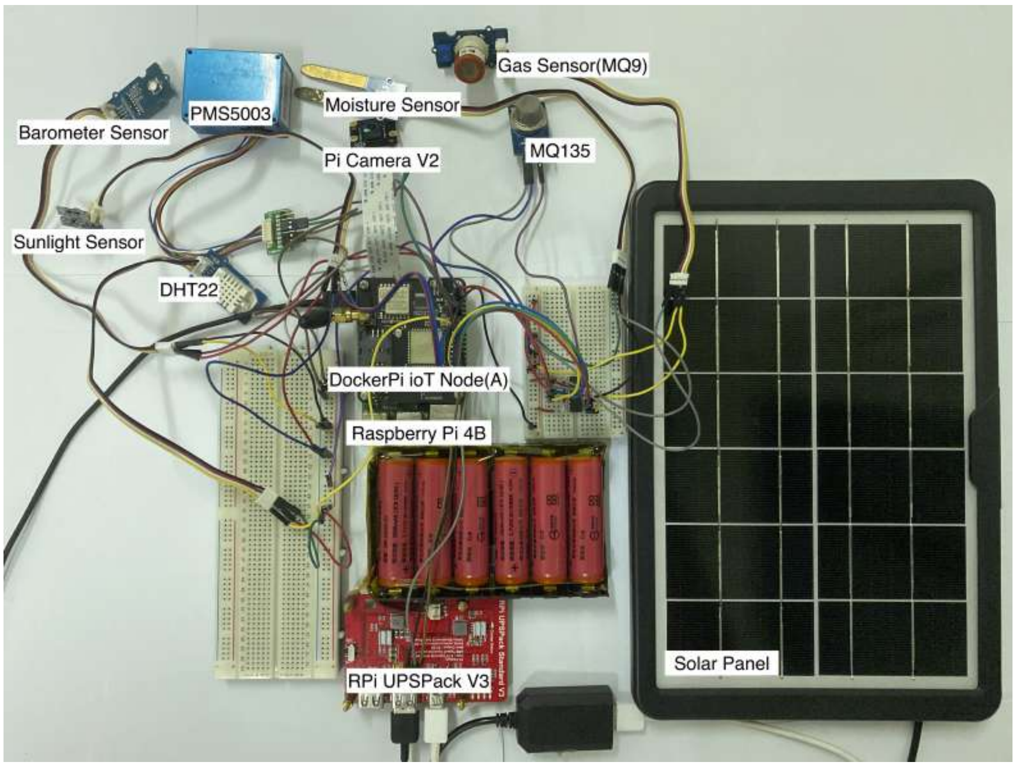


Fig. 2. ecoScout

- EDF:

$$s^{'} = \min_{s_j \in \mathcal{S}} \{ \sum_{\tau_i \in \mathcal{T}} \frac{C_i(ECL^*)/s_j}{T_i(ECL^*)} \leq 1 \} \tag{2}$$

Recall that the current energy criticality level ECL^* is ECL_x if $SWT_x \geq \frac{RC}{IC} > SWT_{x+1}$. As the system continues its operation, the remaining capacity of battery is also going down. The current energy criticality level ECL^* will be switched to a higher level ECL_{x+1} when $\frac{RC}{IC}$ becomes no larger than SWT_{x+1}. Once ECL^* has been changed, the corresponding task version has to be changed. Note that the processor's operating speed also has to be recalculated unless ECL^* is higher than the predefined level. When such a situation occurs, the speed is set to s_{max}.

Moreover, Monitor (labelled by ⑪, Measurement and Statistic (labelled by ⑫), and Log (labelled by ⑬) are tools designed to monitor the on-line usages, to perform measurements and to save statistical data, and to generate log data, respectively.

V. An Example Application: ecoScout

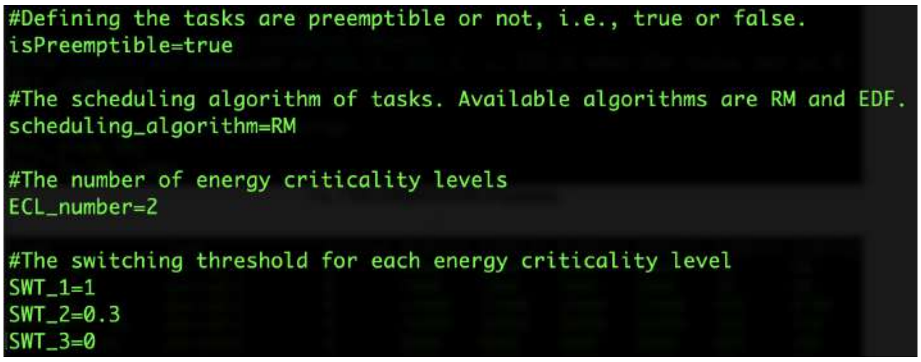
In this section, we present an environmental data collector, called *ecoScout*, to show how to build an application upon meCreate. As shown in Fig. 2, ecoScout is an energy-constrained embedded system since it is powered by battery (a 13200mAh battery pack consists of 6 18600 Li-Ion batteries). However, ecoScout can be also considered an energy-harvesting embedded system because it employed an external photovoltaic module for harvesting energy. The major objective of ecoScout is to collect outdoor environmental eco data. It contains several sensors (e.g., barometer sensor, moisture sensor, air quality sensor (MQ135), gas sensors (MQ9), and sunlight sensors) to collect eco data (e.g., humidity, temperature, air quality, brightness, and soil). Furthermore, ecoScout also has a camera to get environmental photos.

As mentioned in Section IV, developers have to provide (1) configuration file, (2) task description file, and (3) a set of tasks (include multiple versions for different energy criticality levels) in order to build an application upon meCreate. Fig. 3 is the configuration file of ecoScout. It defines that there are two energy criticality levels ECL_1 and ECL_2 in the system (i.e., ECL_number=2), and the switching threshold for the levels ($SWT_1 = 1, SWT_2 = 0.3$ and $SWT_3 = 0$). In other words, ECL_1 is set when the remaining capacity of battery is 100% to 30% of the initial capacity of battery, and ECL_2 is set when the remaining capacity of battery is less than 30%. Fig. 4 is the task description file. As we can observed from the figure, there are several tasks (and their corresponding versions for each energy criticality level) are defined in the file. Based on the configuration file, the task description file, and the tasks, meCreate can generate the application ecoScout. It can be deployed to Raspberry Pi 4B to collect eco data from outdoors. Fig. 5 is the on-line workload monitored by the Monitor module at run time.

VI. Performance Evaluation

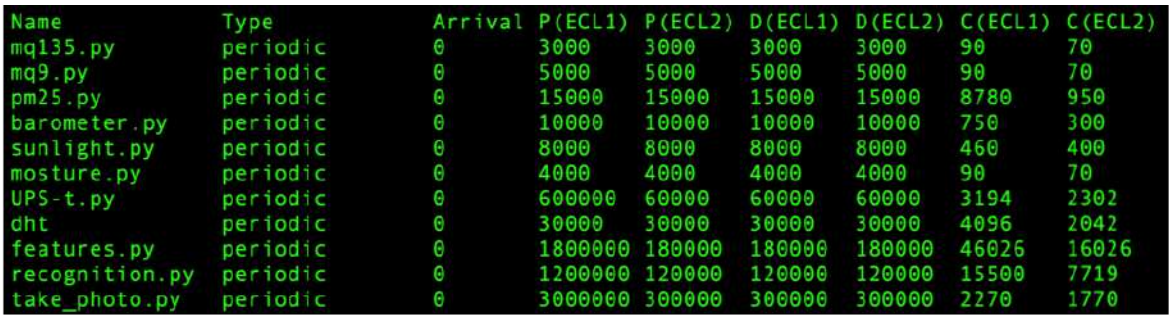
In order to evaluate the performance of meCreate, we have conducted a series of experiments with ecoScout and randomly generated task sets. In the experiments, the energy criticality level ECL^* is set as ECL_1 initially since the battery capacity is plentiful. The ECL^* and the execution speed of tasks will be changed according to the following three strategies:

- FIX: This strategy fixes the energy criticality level as the lowest level (i.e., $ECL^* = ECL_1$) so that higher



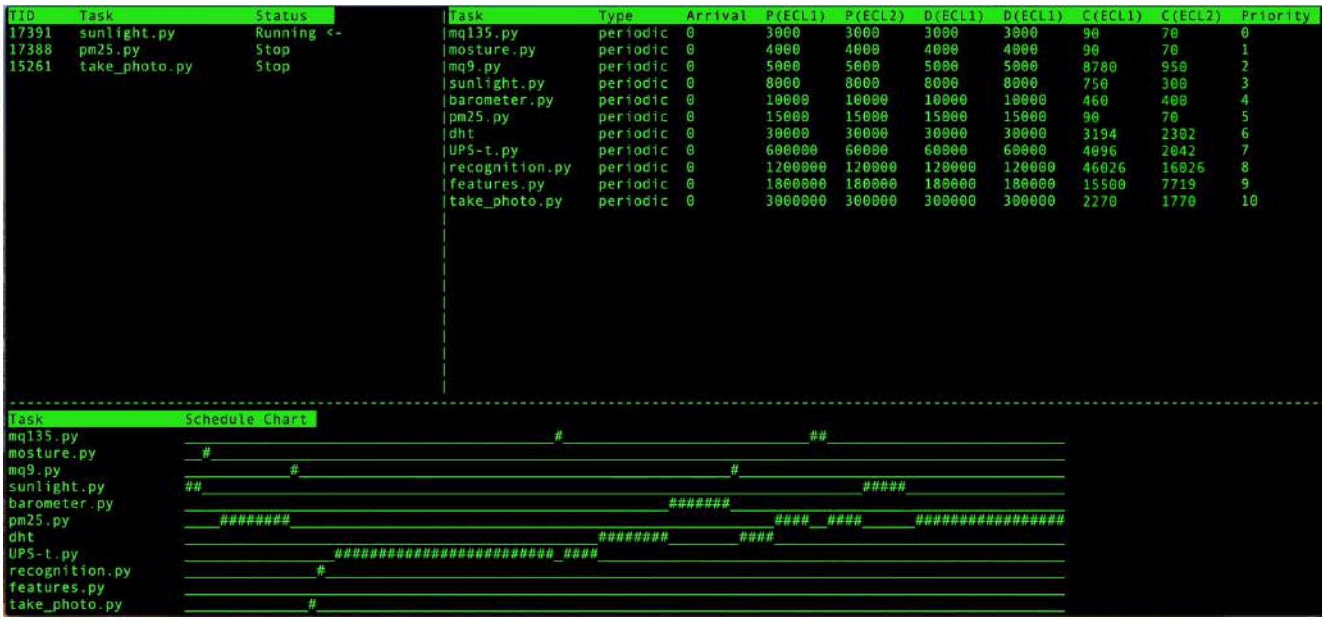
```
#Defining the tasks are preemptible or not, i.e., true or false.
isPreemptible=true

#The scheduling algorithm of tasks. Available algorithms are RM and EDF.
scheduling_algorithm=RM

#The number of energy criticality levels
ECL_number=2

#The switching threshold for each energy criticality level
SWT_1=1
SWT_2=0.3
SWT_3=0
```

Fig. 3. ecoScout: the configuration file.



```
Name            Type        Arrival P(ECL1) P(ECL2) D(ECL1) D(ECL2) C(ECL1) C(ECL2)
mq135.py        periodic    0       3000    3000    3000    3000    90      70
mq9.py          periodic    0       5000    5000    5000    5000    90      70
pm25.py         periodic    0       15000   15000   15000   15000   8780    950
barometer.py    periodic    0       10000   10000   10000   10000   750     300
sunlight.py     periodic    0       8000    8000    8000    8000    460     400
mosture.py      periodic    0       4000    4000    4000    4000    90      70
UPS-t.py        periodic    0       600000  60000   60000   60000   3194    2302
dht             periodic    0       30000   30000   30000   30000   4096    2042
features.py     periodic    0       1800000 180000  180000  180000  46026   16026
recognition.py  periodic    0       1200000 120000  120000  120000  15500   7719
take_photo.py   periodic    0       3000000 300000  300000  300000  2270    1770
```

Fig. 4. ecoScout: the task description file.



```
TID    Task            Status          |Task            Type      Arrival P(ECL1)  P(ECL2) D(ECL1) D(ECL1) C(ECL1) C(ECL2) Priority
17391  sunlight.py     Running <-      |mq135.py        periodic  0       3000     3000    3000    3000    90      70      0
17388  pm25.py         Stop            |mosture.py      periodic  0       4000     4000    4000    4000    90      70      1
15261  take_photo.py   Stop            |mq9.py          periodic  0       5000     5000    5000    5000    8780    950     2
                                       |sunlight.py     periodic  0       8000     8000    8000    8000    750     300     3
                                       |barometer.py    periodic  0       10000    10000   10000   10000   460     400     4
                                       |pm25.py         periodic  0       15000    15000   15000   15000   90      70      5
                                       |dht             periodic  0       30000    30000   30000   30000   3194    2302    6
                                       |UPS-t.py        periodic  0       600000   60000   60000   60000   4096    2042    7
                                       |recognition.py  periodic  0       1200000  120000  120000  120000  46026   16026   8
                                       |features.py     periodic  0       1800000  180000  180000  180000  15500   7719    9
                                       |take_photo.py   periodic  0       3000000  300000  300000  300000  2270    1770    10
                                       |
                                       |
                                       |
                                       |
                                       |
                                       |
                                       |
                                       |
                                       |
                                       |
---------------------------------------------------------------------------------------------------------------------------------
Task                Schedule Chart
mq135.py            __________________________________________#____________________________##___________________________
mosture.py          __#_________________________________________________________________________________________________
mq9.py              ____________#_________________________________________________#_____________________________________
sunlight.py         ##___________________________________________________________________________#####__________________
barometer.py        _______________________________________________________#######______________________________________
pm25.py             ____########_______________________________________________________####__####______#################
dht                 _______________________________________________########________####_________________________________
UPS-t.py            _________________########################_####______________________________________________________
recognition.py      _______________#____________________________________________________________________________________
features.py         ____________________________________________________________________________________________________
take_photo.py       ______________#_____________________________________________________________________________________
```

Fig. 5. ecoScout: On-line monitoring

accurate results can be obtained. Since ECL^* is fixed to ECL_1, the version τ_i^1 will be executed on the processor. Note that all tasks will be executed at speed s_{max}.

- DYN: This strategy switches the energy criticality level dynamically according to the switching thresholds (i.e., switching the value of ECL^* from ECL_1 to ECL_2 when the remaining capacity of battery is less than 30%). Note that the version τ_i^2 will be executed after ECL^* has been changed to ECL_2. Note that all tasks are executed at speed s_{max}.
- DYN+DVFS: This is the default strategy of meCreate. Similar to DYN strategy, this strategy sets ECL^* to ECL_1 initially, and it will be changed to ECL_2 when the remaining capacity of battery is less than 30%. However, this strategy also calculates the execution speed $s^{'} \leq s_{max}$ for task execution according to Equation (2) and (1). After ECL^* has been changed, the corresponding version τ_i^2 of each task τ_i will be executed at the execution speed $s^{'}$.

A. *ecoScout*

In the first part of the experimental results, ecoScout have been modified so that it is capable to work with FIX, DYN, and DYN+DVFS strategies. For each strategy, 10 experi-

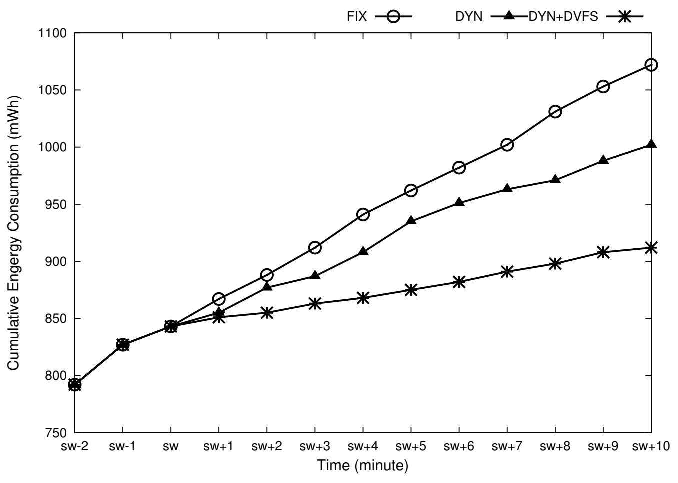


Fig. 6. Cumulative energy consumption of ecoScout.

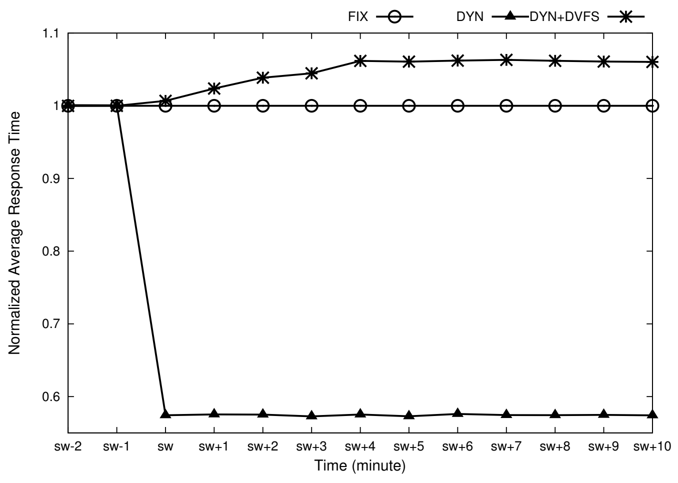


Fig. 8. Normalized average response time of ecoScout.

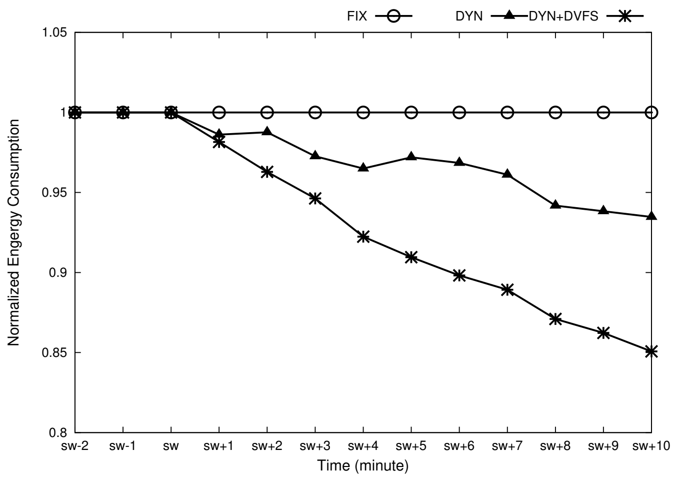


Fig. 7. Normalized energy consumption of ecoScout.

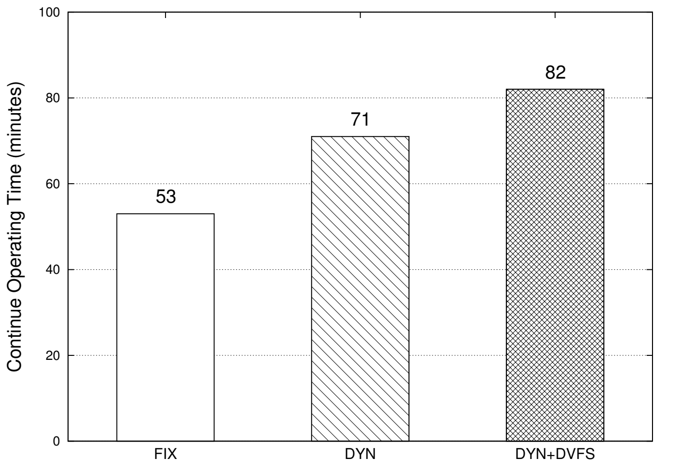


Fig. 9. Continue operating time of ecoScout.

ments have been conducted and the results were averaged. All experiments start with a fully charged battery (with 2200 mAh capacity) until the battery is drained completely. Figures 6 and 7 show the cumulative and the normalized energy consumption (with respect to that of FIX). Note that the behaviors of FIX, DYN, and DYN+DVFS strategies are all the same when the remaining capacity of battery is no less than 30% (i.e., the switching threshold in the experiments). Figures 6 and 7 only show the results in the interval $[sw-2, sw+10]$, where sw is the average time that the remaining capacity of battery is less than 30% (we have observed that sw is 37 minutes).

As shown in Fig. 6 and Fig. 7, DYN+DVFS (which is the default strategy of meCreate) outperforms FIX and DYN greatly. It is obvious that FIX strategy performs worst because it always selects the corresponding version τ_i^1 of each task τ_i and all tasks are executed at speed s_{max}. DYN outperforms FIX because the energy criticality level will be changed from ECL_1 to ECL_2 at the time (i.e., sw) that the remaining capacity of battery is less than 30%. Since the ECL^* has been changed, the comparably lightweight version τ_i^2 of each task τ_i will be selected and to be executed. As a result, DYN outperforms FIX after time sw. DYN+DVFS is the default strategy of meCreate. After ECL^* has changed to ECL_2, DYN+DVFS calculates a proper speed $s^{'}$ to execute tasks such that more energy can be saved.

In addition to the energy efficiency, we also conducted experiments for evaluation of the average response time of tasks so that the performance of meCreate can be better understand. Fig. 8 shows the normalized average response time of tasks (with respect to that of FIX). As we can observed, the performance rank is DYN $>$ FIX $>$ DYN+DVFS. Recall that both DYN and FIX execute tasks at the maximum processor speed s_{max}. After ECL^* has been changed (i.e., from ECL_1 to ECL_2), DYN and FIX execute τ_i^2 and τ_i^1 of every task τ_i, respectively. Since $C(\tau_i^2) \leq C(\tau_i^1)$, the response time of τ_i^2 is no longer than that of τ_i^1.

We also notice that DYN outperforms DYN+DVFS although both of them execute the same version of task τ_i, i.e., τ_i^2. It is because of DYN and DYN+DVFS execute tasks at s_{max} and $s\prime$, respectively, after ECL^* has been changed. Since $s^{'} \leq s_{max}$, DYN+DVFS will postpone the completion time of tasks. Another interesting fact is FIX outperforms DYN+DVFS. Compare with FIX, DYN+DVFS postpones the completion time of tasks as late as possible so that the energy consumption is reduced. Although the late completion causes a performance impact, it is allowed if the completion of a real-time task no later than its deadline. Therefore, the performance impact of DYN+DVFS can be consider an acceptable price to pay for saving more energy.

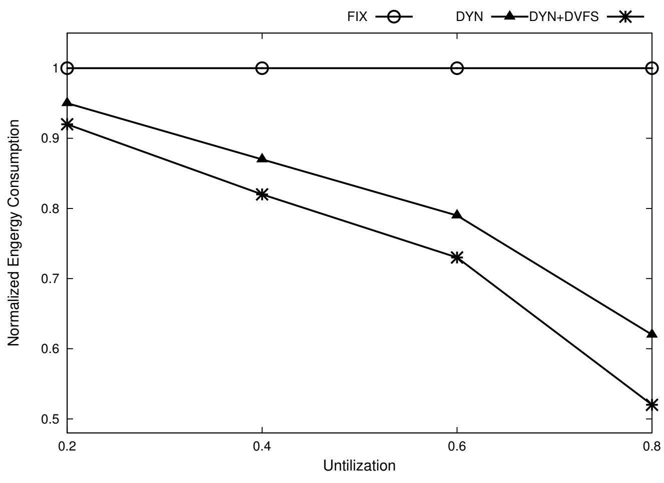


Fig. 10. Normalized average energy consumption of randomly generated task sets.

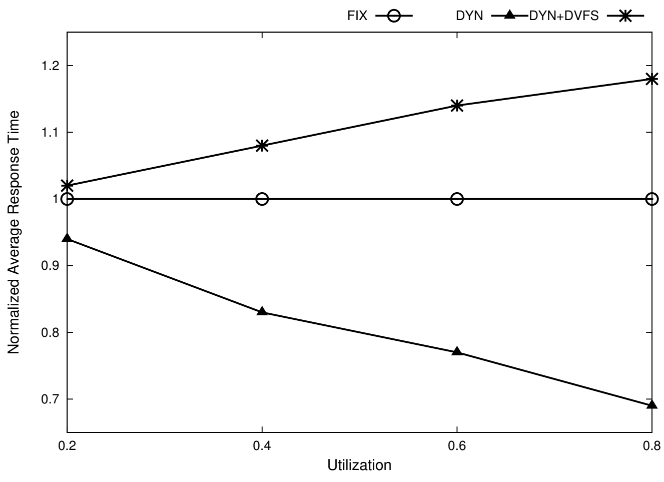


Fig. 11. Normalized average response time of randomly generated task sets.

Finally, we also conducted the long lasting tests for the three strategies FIX, DYN and DYN+DVFS in the experiments. As shown in Fig. 9, ecoScout with the default strategy DYN+DVFS continued to operate for 82 minutes while FIX and DYN are 53 and 71 minutes, respectively.

B. Randomly Generated Task Sets

A testbed has been built upon meCreate for evaluating the performance of randomly generated task sets. The parameters are given in Table I.

TABLE I
THE PARAMETER SETTINGS OF RANDOMLY GENERATED TASK SETS.

Parameters	Values
Utilization of a task set	(0.2 $\sim$ 0.8) stepped by 0.2
Number of tasks of a task set	(30 $\sim$ 300)
Task period*	(2,000 $\sim$ 30,000)ms
Task computation time*	(100 $\sim$ 2,000)ms
Shrink ratio sr	(0.1 $\sim$ 0.5)

*Values for ECL_1.

We generated feasible task sets by setting each task set's utilization from 20% to 80% (stepped by 20%). For each utilization, 10 task sets were generated randomly and their experimental results were averaged. For each task set, we generate 30 to 300 tasks randomly and their period and computation time were selected from the ranges (2,000 $\sim$ 30,000)ms and (100 $\sim$ 2,000)ms randomly. Note that the parameters (i.e., task period and computation time) in Table I are used to generate tasks for ECL_1. For each task τ_i, we generated the corresponding task τ_i^2 for ECL_2 according to the shrink ratio sr (from 0.1 to 0.5 randomly), i.e., the computation time of τ_i^2 is set as $C_i(ECL_2) = C_i(ECL_1) \times sr$. In the experiments, each task set has been executed for more than 60 minutes.

Figures 10 and 11 are the average results in the interval [sw-2, sw+10], where sw is the time that the remaining capacity of battery is less than the threshold (i.e., 30%). In particular, Figure 10 shows the normalized energy consumption of tasks (with respect to that of FIX strategy). The performance rank is DYN+DVFS > DYN > FIX. Figure 11 shows the normalized average response time of tasks (with respect to that of FIX strategy). The performance rank is DYN > FIX > DYN+DVFS. Note that the experimental results in Figures 10 and 11 are consistent with the results of Figures 7 and 8.

VII. CONCLUSION

This paper presents a platform meCreate for developing and executing energy-efficient applications on embedded real-time systems. meCreate is designed to support an innovative task model, called energy-critical multi-version real-time task model, which assumes that there are multiple energy criticality levels in the system and it can be switched dynamically according to the remaining capacity of battery. It also assumes that each task has multiple versions (from higher energy consumption to lower consumption) corresponding to each energy criticality level. Based on an environmental data collector application (i.e., ecoScout), we have demonstrated the procedures for build an application for energy-constrained ERTS upon meCreate. The energy efficiency and the performance have been evaluated for ecoScout and randomly generated task sets, for which we have some encourage results.

REFERENCES

[1] J.-J. Chen and C.-F. Kuo, "Energy-efficient scheduling for real-time systems on dynamic voltage scheduling (dvs) platforms," in *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA2007)*, 2007.
[2] J. Wu, "A Survey of Energy-Efficient Task Synchronization for Real-Time Embedded Systems," in *Proceedings of the 23rd IEEE RTCSA*, Hsinchu, Taiwan, August 16-18 2017.
[3] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, 2007, pp. 239–243.
[4] J. Wu and J.-L. Wang, "A Real-Time Embedded Platform for Mixed Energy-Criticality Systems," in *Proceedings of the 7th IEEE International Conference on Applied System Innovation (IEEE ICASI)*, Chiayi, Taiwan, September 24-25 2021.
[5] W. Haberl, M. Tautschnig, and U. Baumgarten, "Running COLA on Embedded Systems," in *Proceedings of the 2008 International MultiConference of Engineers and Computer Scientists (IMECS)*, Hong Kong, March 19-21 2008, pp. 922–928.
[6] C.-F. Hsu, "A Component-Based Software Development Platform for Rapid Prototyping of Embedded Software," Master Thesis, Department of Computer Science and Information Engineering, National Pingtung University, July 2009.

[7] T. Kamiyama, M. Tamura, T. Soeda, M. Yoo, and T. Yokoyama, "An Embedded Control Software Development Environment with Simulink Models and UML Models," *IAENG International Journal of Computer Science*, vol. 39, no. 3, pp. 261–268, 2012.

[8] Linux Foundation, "The Yocto Project - It's Not an Embedded Linux Distribution, It Creates a Custom One for You," Available on-line at http://www.yoctoproject.org (visited on March 13, 2022.), 2022.

[9] Buildroot Developers, "The Buildroot User Manual (revision 08967921c4)," Available on-line at https://buildroot.org/downloads/manual/manual.pdf (visited on March 13, 2022.), 2022.

[10] OpenWrt, "About the OpenWrt/LEDE Project," Available on-line at https://openwrt.org/about (visited on March 13, 2022.), 2022.

[11] L. Acasandrei and A. Barriga, "SHORES: Software and Hardware Open Repository for Embedded Systems," in *Proceedings of the 2017 World Congress on Engineering and Computer Science (WCES)*, San Francisco, USA, October 25-27 2017, pp. 26–31.

[12] I. Assayad, L. Eljadiri, M. Krichen, A. Zakari, W. Adoni, and T. Nahhal, "A Novel Architecture Prototyping Framework with Generic Properties Verification for Sub-architectures," *Engineering Letters*, vol. 29, no. 2, pp. 634–644, 2021.

[13] H. Khandelwal, P. Mankodi, and R. Prajapati, "Enhancement of Automation Testing System Using Yocto Project," in *Proceedings of the 2017 International Conference of Electronics, Communication and Aerospace Technology (ICECA)*, April 2017, pp. 20–22.

[14] A. Biswas, D. Biswas, S. S. Chauhan, and A. Borwankar, "Smart Home Equipment Control System with Raspberry Pi and Yocto," in *Proceedings of 4th World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, July 27-28 2020, pp. 553–558.

[15] H. Zhen and F. Hui, "Embedded Parking System Based on OpenWrt Database Server," in *Proceedings of 5th International Conference on Instrumentation and Measurement, Computer, Communication and Control (IMCCC)*, September 18-20 2020, pp. 1469–1474.

[16] K. V. Prashanth, P. S. Akram, and T. A. Reddy, "Real-Time Issues in Embedded System Design," in *Proceedings of the 2015 International Conference on Signal Processing and Communication Engineering Systems*, 2015, pp. 167–171.

[17] M. Engin, "Embedded and Real Time System Design: A Case Study Fire Fighting Robot," in *Proceedings of the 5th Mediterranean Conference on Embedded Computing (MECO)*, 2016, pp. 18–21.

[18] S. Cardona and E. Giraldo, "Real-time Multivariable Embedded Control of a Ball and Plate Prototype," *Engineering Letters*, vol. 29, no. 4, pp. 1375–1380, 2021.

[19] L. A. Rios-Norena, J. S. Velez-Ramirez, and E. Giraldo, "Real-Time Optimal Embedded Control of a Double Inverted Pendulum," *IAENG International Journal of Computer Science*, vol. 49, no. 2, pp. 341–348, 2022.

[20] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the Association for Computing Mahinery (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

Jun Wu received a Ph.D. degree in Computer Science and Information Engineering from National Chung Cheng University, Chiayi, Taiwan. He is currently a Professor at Department of Computer Science and Information Engineering, National Pingtung University, Pingtung, Taiwan. He was also a Visiting Researcher at the University of York, a Visiting Scholar at the University of Pittsburgh, and a Visiting Scholar at Academia Sinica. His research interests include: (1) energy-efficient task scheduling and synchronization for real-time embedded systems, and (2) high performance resource management for virtualization platforms. Dr. Wu is a member of IEEE and IAENG.

Jia-Lin Wang received a Master degree in Computer Science and Information Engineering from National Pingtung University, Pingtung, Taiwan, in 2021. His research interests include real-time embedded system platforms and database systems. He is currently an Engineer