electronics

MDPI

Article

Task Scheduling to Constrain Peak Current Consumption in Wearable Healthcare Sensors

Robert Simon Sherratt [1,*], Balazs Janko [1], Terence Hui [2], William S. Harwin [1], Nilanjan Dey [3], Daniel Díaz-Sánchez [4], Jin Wang [5] and Fuqian Shi [6]

1 Department of Biomedical Engineering, University of Reading, Reading RG6 6AY, UK
2 IO Senses Ltd, Reading RG1 3BJ, UK
3 Department of Information Technology, Techno India College of Technology, West Bengal 740000, India
4 Department of Telematic Engineering, Universidad Carlos III de Madrid, Leganés, 28911 Madrid, Spain
5 School of Computer & Communication Engineering, Changsha University of Science & Technology, Changsha 410114, China
6 College of Information and Engineering, Wenzhou Medical University, Wenzhou 325035, China
* Correspondence: r.s.sherratt@reading.ac.uk; Tel.: +44-118-378-8588

Received: 30 May 2019; Accepted: 12 July 2019; Published: 15 July 2019

Abstract: Small embedded systems, in our case wearable healthcare devices, have significant engineering challenges to reduce their power consumption for longer battery life, while at the same time supporting ever-increasing processing requirements for more intelligent applications. Research has primarily focused on achieving lower power operation through hardware designs and intelligent methods of scheduling software tasks, all with the objective of minimizing the overall consumed electrical power. However, such an approach inevitably creates points in time where software tasks and peripherals coincide to draw large peaks of electrical current, creating short-term electrical stress for the battery and power regulators, and adding to electromagnetic interference emissions. This position paper proposes that the power profile of an embedded device using a real-time operating system (RTOS) will significantly benefit if the task scheduler is modified to be informed of the electrical current profile required for each task. This enables the task scheduler to schedule tasks that require large amounts of current to be spread over time, thus constraining the peak current that the system will draw. We propose a solution to inform the task scheduler of a tasks' power profile, and we discuss our application scenario, which clearly benefited from the proposal.

Keywords: wearable; low-power; embedded; task scheduler; healthcare

1. Introduction

The use of personalized healthcare devices, from wearable fitness trackers to smartphone apps, has been of growing significance. We are now seeing artificial intelligence (AI)-based intelligent devices address the daily challenges of those living with chronic diseases, including Alzheimer's, cerebral palsy, chronic obstructive pulmonary disease (COPD), diabetes, hypertension, multiple sclerosis, obesity and Parkinson's disease. One of the main challenges has been to embed enough computing power into a wearable device while keeping the device lightweight. This places demanding constraints on the battery weight and size, so power management is of utmost importance.

Power efficiency has been a long-running research challenge, and is often addressed though low-power designs enabling a device to run for a long time between charges. This is particularly important for devices aimed at addressing the needs of people living with dementia who may not remember to charge the device.

To enable battery-powered devices to run for a long time between charges, research has focused on hardware solutions, software solutions, and hybrids of the two. Achieving a low-power solution is

often seen as a balance between reducing battery current by placing the central processing unit (CPU) into a low-power idle state (i.e., sleep mode) versus the need for a CPU to consume battery current to execute task code at full clock speed. While there are numerous smart devices on the market based on common operating systems, these generic and ubiquitous devices are designed to support many features, and as such, they have a relatively heavy use of power. On the other hand, research has indicated that low-power embedded solutions can come from devices that are based on a real-time operating system (RTOS) that has been targeted to support the resource-constrained embedded device with their application solely focused on the use case.

An important element of an RTOS is the task scheduler (TS), which is responsible for deciding when software tasks are to be executed (preemptively), and is typically based on an assigned task priority [1]. In addition, power management is also an important design enabler for an RTOS. Modern RTOS systems address this objective by only waking up the CPU when processing is required, and enter a low-power idle state when all tasks have completed or are waiting. While dynamic voltage frequency scaling (DVFS) and dynamic power management (DPM) are important and topical research issues [2–4], our focus in this paper is concerned with small embedded devices that typically can only be either running or idle, and only use one core voltage. It has been also shown that when the TS places a system into idle, the battery current is reduced, enabling certain battery chemistries to recover some charge [5], potentially up to 20%. Therefore, such duty cycling is easily enabled using RTOS mechanisms.

Li et al. [6] presented a dynamic task scheduling algorithm focusing on energy reduction. The work considered the dynamic scheduling of algorithms to obtain a global minimization of power in a cloud computing platform.

Liu et al. [7] considered a delay optimal task scheduling technique on a mobile device to minimize the power of a local CPU load or to offload a task to a cloud edge server. Computation tasks were scheduled based on the queueing state of a task buffer, the execution state of the CPU, and the state of the transmission unit. Their algorithm searched for an optimal solution that minimizes the global power consumption, but of course, the search itself consumed power.

Ghofrane et al. [8] used a neural network (NN) to obtain a global minimized power schedule, while Li and Wu [9] presented an ant colony optimized (ACO) search method for their task scheduler. While both approaches produce global power savings, the application of such schemes are directed toward higher-end computing systems rather than wearable healthcare devices due to the considerable power needed to perform the optimizing search.

Nguyen et al. [10] presented interesting work by considering various algorithms for task scheduling for the Internet of Things (IoT) by considering completion time and operating costs. While the work itself was not concerned with power scheduling, it is possible that the concepts presented may also be used for power management.

Ahmad et al. [11] presented a formal method for the evaluation of real-time task scheduling in mission critical systems. Such research will be useful to medical IoT systems, but they did not consider power management.

Zagan et al. [12] presented scheduling of tasks via the use of a hardware architecture as opposed to a software-based RTOS.

Singh et al. [13] considered task scheduling for data centers by considering process time flow in order to minimize the overall energy consumed by the data center. The work presented an interesting power model, but the focus of the work was to minimize overall energy, not constrain peak current.

Ahmad et al. [14] noted that as small embedded devices have limited resources (power and memory), then an RTOS should consider the required tasks to produce minimum CPU power that still gives correct operation. The work presented a resource-aware approach to minimize tasks based on device profiles. A power profile for each device is an interesting idea, but again, the focus of the work was the global minimization of CPU power, not to constrain peak current.

Chowdhury and Chakrabarti [15] presented interesting work by considering power profiles for battery-operated systems in order to maximize the residual charge left in the battery. Their system used dynamic voltage scaling (DVS), which is currently only present on high-end CPUs, and not applicable in our case with the current technology level. Again, this work was useful to create overall power savings, but did not constrain peak current demands.

It is clear from the literature that where electrical power is concerned, the previous research objectives have been to schedule tasks to globally minimize the consumed power, not to constrain peak power. To the authors' knowledge, no work has considered task scheduling for the context of constraining peak electrical current. This position paper proposes that low-power embedded systems, particularly wearable healthcare devices where battery operation is paramount, can significantly benefit when the software task scheduler embedded in the operating system is informed of the current drain for each task, so that higher-current tasks can be separated and executed over time in order to guarantee a maximum peak current drain. Constraining the peak current aids in the design of the internal regulator and battery management circuits. It also enables energy harvesting systems and brownout recovery due to lower peaks of current in the device's startup condition. In this paper, we present how the need for such a system was realized through our real-life application scenario, and then present our general proposal to address the issue.

The paper is organized as follows: Section 2 discusses the development of the concept of the need for power-aware schedulers to reduce peak power. Section 3 presents our proposed solution, Section 4 presents a discussion, and Section 5 presents our conclusions.

2. Application Scenario

The position statement from this paper is derived from experience in creating the wearable device used in the SPHERE (https://www.irc-sphere.ac.uk) project. SPHERE was a large interdisciplinary research project to create technology and algorithms for monitoring people for residential healthcare with the objective of enabling people to live in their home for longer. The SPHERE architecture integrates environmental, video, and wearable sensors into a smart hub and backbone network [16] enabling intelligent processing and data-driven decisions from the fusion of the sensor data and data mining.

The SPHERE wearable [16] was specifically designed for low-power operation to enable pervasive monitoring; otherwise, behaviors could be influenced. It was based on a Texas Instruments CC2650 (Dallas, TX, USA) [17] system on chip (SoC) supporting either Bluetooth 4.2 [18] (often called Bluetooth low energy (BLE) or Bluetooth smart) or 802.15.4 [19]. The SoC contains an ARM Cortex-M0 CPU dedicated to controlling just the radio interface and an ARM Cortex-M3 CPU for hosting the application code. The SPHERE wearable also contained a FLASH memory device [20] for local permanent storage, two low-power accelerometers [21] allowing for gyro-free 3D spatial measurements [22], and an inertial measurement unit (IMU) [23]. The CC2650 does have an internal DC–DC convertor in the SoC, allowing the SoC to be powered from 1.8 V–3.8 V. However, a high performance and efficient external buck convertor [24] was used to step down the voltage from a 100-mA lithium polymer (LiPo) battery (typically 4.2 V to 3.6 V) to 1.8 V for the SoC core. The ability to wirelessly charge the LiPo battery via Qi was also included. The wearable was programed using an RTOS [25].

The SPHERE wearable was configured to use the BLE radio in a non-connected mode [26]. The wearable application packaged the sensed movement, battery voltage, and SoC core temperature measurements into a non-connectable BLE advertisement packet (ADV), which was detected by passive BLE hubs spread across the residence. In BLE, devices advertise their presence by transmitting on the advertising channels (37, 38, and 39) in sequence [26]. Room hubs capture the encapsulated data and forward the data to be fused with the video data. Typically, the wearable lasted for four weeks on one charge when transmitting one BLE ADV packet per second, while variants of more frequent transmissions were also obtained. The SPHERE wearable in the above configuration worked well when deployed.

Then, the SPHERE wearable was required to be used in a different set of experiments on monitoring people with Parkinson's disease in their own home continually for three days, with the aim of capturing movement events called near-falls [27], which are signature movements that occur prior to an actual fall. This application required the IMU to continually sample movement at 60 Hz and store the data to the local FLASH for download three days later. An interrupt was generated by the IMU 60 times a second to request that the CPU read the IMU's data. As the battery current when writing a single page of data to the FLASH was 12.4 mA (SoC core current plus FLASH page write current), then the movement data was not in this case streamed over BLE in order to save power, enabling three-day operation.

The battery current profile when reading data from the IMU and writing the data to a FLASH page is given in Figure 1 over the whole IMU interrupt service routine (ISR). In Figure 1, point (a) indicates that before the IMU interrupt, the CPU is in its idle state; therefore, the 1.2-mA wearable device battery current is dominated by the IMU, which is always on. When the interrupt from the IMU occurs, the SoC changes from its low-frequency (LF) clock to the high-frequency (HF) clock at point (b). Once the HF clock is running, the CPU is fully operational to execute code using the HF clock with the battery current increasing to 5.2 mA. The data in the IMU is read via the SoC SPI bus at point (c), post-processed at point (d), and then, the data is written to local FLASH memory at point (e), resulting in a significant peak of battery current (12.4 mA). Then, the SoC returns back to idle while the IMU is still running at point (f).

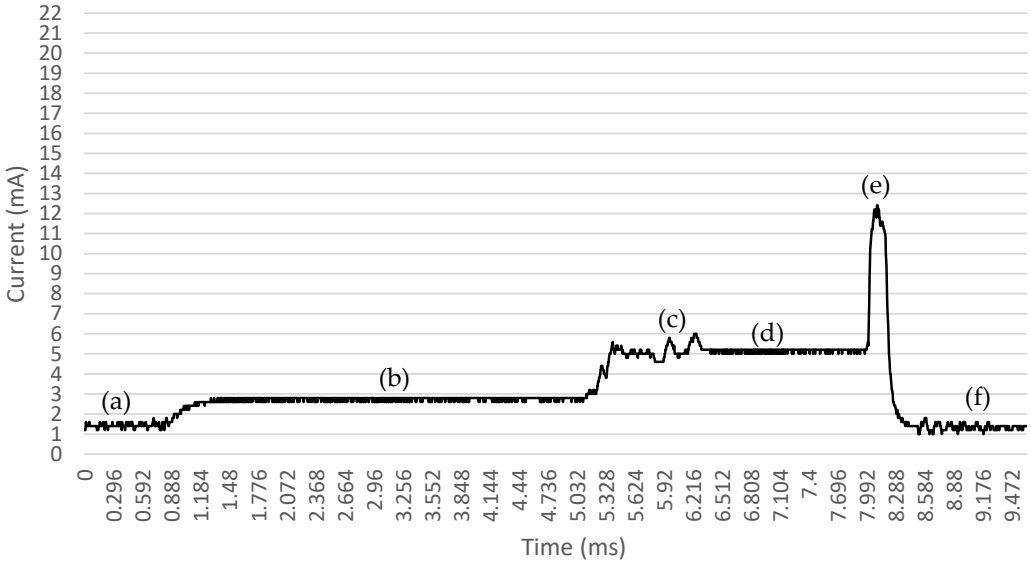

Figure 1. Current profile of the wearable over the inertial measurement unit (IMU) interrupt service routine (ISR). (a) is the idle current with IMU always on (1.2 mA); point (b) is where the system on chip (SoC) wakes up to process the ISR (2.8 mA). Then, the SoC enables the high-frequency clock; point (c) is where the SoC reads the IMU data over the SPI bus, and point (d) is where the IMU data is post-processed (5.2 mA); point (e) is where the IMU data is stored in the FLASH memory using a page write process (12.4 mA); and point (f) is where SoC then returns to idle with just the IMU running.

In our near-fall detection application, a connectable BLE mode was used to enable a smartphone/tablet to control the wearable over BLE via an app enabling the user to start/stop the logging, and to download the movement data. Therefore, the wearable advertised its presence using a connectable BLE ADV process once per second, requiring the transmissions of three peaks from 10.5 mA to 12 mA, as shown in Figure 2, over point (g). A user can connect to the device to start the logging process, and then disconnect, leaving the device logging to the FLASH memory over three days while the device periodically advertises its presence for a host to connect to the device.

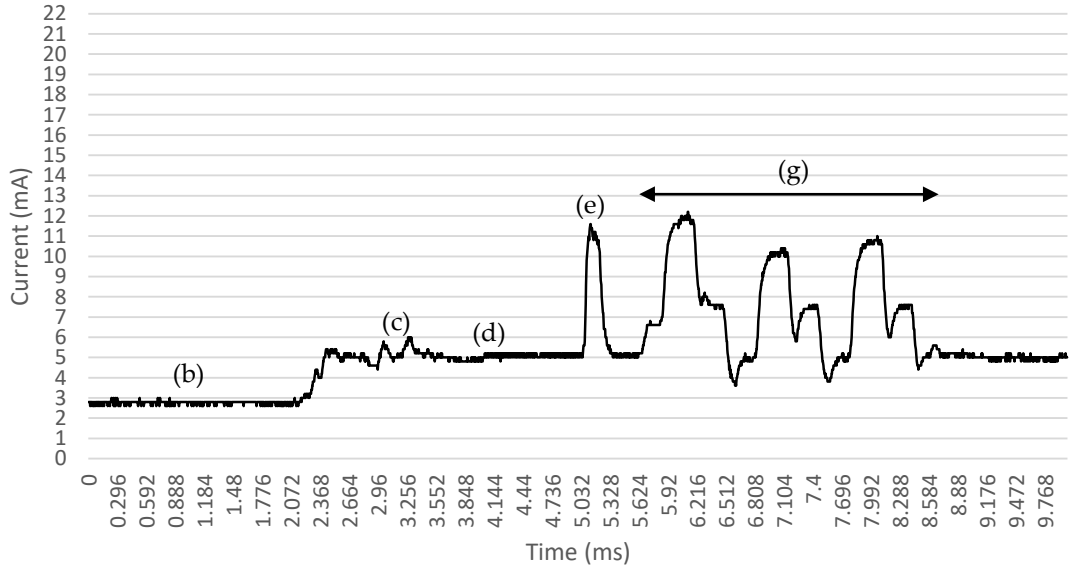

Figure 2. Current profile of just the latter part of the IMU ISR followed by a Bluetooth low energy (BLE) advertisement packet (ADV) process. Point (g) is the three transmissions on each of the three BLE ADV frequencies. Of note is that the FLASH write process point (e) and the BLE ADV process point (g) are separated in time, extending the current drain into that of period (f), as depicted in Figure 1.

However, it was found that the device would brownout after 24 h of successful operation. After investigations, we realized that the ADV transmission process was asynchronous to the operation of the FLASH page write process. When both the BLE ADV and the FLASH write processes occurred at the same time, then the current drawn from the Buck convertor contained the SoC core current, the current to write to the FLASH, and the current required for the BLE ADV transmission, as can be seen in Figure 3, point (h). By using the external buck to drive the SoC core at its required 1.8 V in order to minimize power losses, then the consequence was that when a large instantaneous current is drawn by the wearable, then the Buck convertor took time to respond, and the SoC core voltage dropped, causing the brownout.

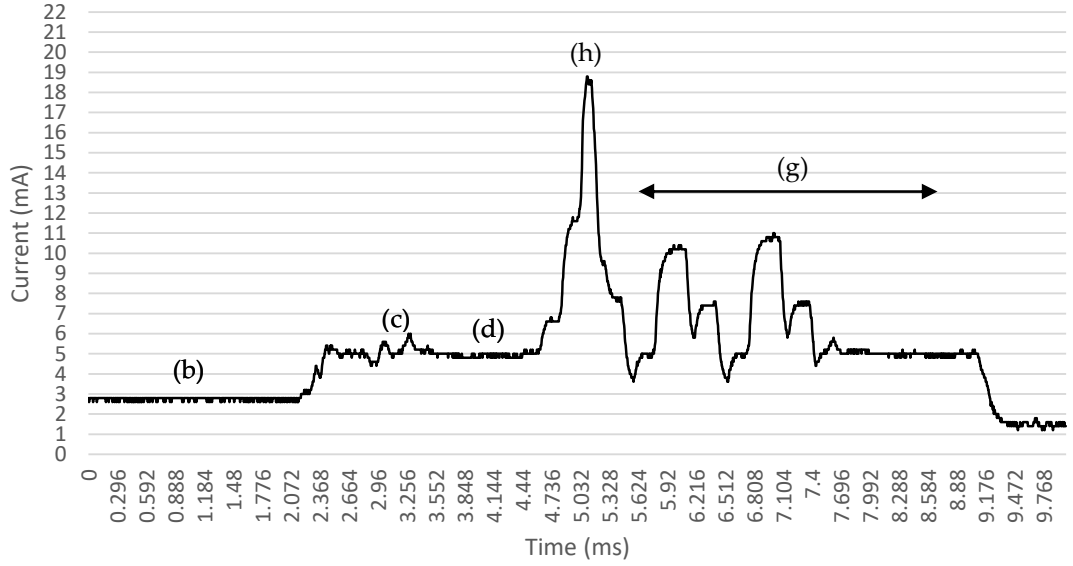

Figure 3. Current profile of just the latter part of the IMU ISR where the BLE ADV process occurs over the same time as the first BLE ADV transmission. Point (h) is now the summation of the SoC current, the FLASH write current, and the BLE ADV transmit current (18.8 mA).

Application Scenario Solution

The issue of the brownout, in this case, was simply solved by synchronizing the ADV transmission process with the FLASH write process, such that the ADV transmission occurred spaced in-between successive FLASH write processes, allowing the buck time to recover. Thus, it removed the large peak of current by spreading the current drawn from the buck over time.

It was these experiments and solutions that have led to our position in this paper of proposing task schedulers in embedded systems to be aware of the current (power) requirements for each task, and for the scheduler to schedule tasks based not only on task priority, but also to separate in time tasks that have large peak currents, thus constraining the maximum current drawn at any time. In effect, this process associated a peripheral's electrical current profile with a given software task. Such an approach does not itself save any power; rather, it constrains the peak current drawn.

3. Proposed Modifications to RTOS Task Parameters

From the literature, it was clear that while there are many works that have focused on the creation of task schedulers to schedule tasks with the objective of globally minimizing electrical power, there are no works that have focused the task scheduler to constrain peak electrical current. The placement of such a constraint would not only simplify the power delivery circuits of wearable devices, but also have advantages in EMI reduction.

While Ahmad et al. [14] proposed that an RTOS should consider the required tasks to produce minimum global CPU power via a resource-aware approach based on device profiles, we propose that a scheduler can be informed of the required current consumption for each task when the task is instantiated to enable the task scheduler to then separate higher current tasks over time.

The design and modification of operating system task schedulers is outside of the scope of this work. Rather, we propose the position that there are significant advantages to be gained in the power delivery of wearable devices if task schedulers were made aware of the current required for each task. In order to create such a task scheduler, then the task scheduler would need to be informed of an expected electrical current profile for each software task. Clearly, a software task itself requires current for the CPU to perform the task, but may also use a peripheral device (e.g., the case of a FLASH memory page write) and the associated current to perform that task. In the case where the required peripheral current may be of relatively short duration to the overall software task, then the task can be split into multiple concurrent tasks, as was the application scenario presented in Section 2.

In order to create software tasks, each task needs to be instantiated in runtime though the scheduler. The typical task instantiation code is given in Figure 4. It can be seen in Figure 4 that a structure taskParams is created to contain the task parameters required, including the task stack TaskStack (of size TASK_STACK_SIZE) and task priority. Then, the task scheduler can construct the new task in the scheduler in runtime using the information in taskParams and the address of the task.

```
Task_Struct Task;
Char TaskStack[TASK_STACK_SIZE];
Task_Params taskParams;

// Configure task
Task_Params_init(&taskParams);
taskParams.stack = TaskStack;
taskParams.stackSize = TASK_STACK_SIZE;
taskParams.priority = 1;

Task_construct(&Task, taskFxn, &taskParams, NULL);
```

Figure 4. Typical method in a real-time operating system (RTOS) to instantiate task taskFxn with the task parameters configured in a structure.

As parameters are used to define the task, then an additional parameter, which was in this paper called the additional_current, may be added to the structure to pass the expected additional current (of defined units) that needs to be taken into consideration when the task runs (due to the peripheral or radio transceiver action), as shown in Figure 5. In this way, tasks can then be scheduled with the full knowledge of the required current that will be drawn when that task is ran, thus allowing for the scheduling of tasks to constrain the peak current. The FLASH write task sends a command to the FLASH device to start its internal process of transferring the page memory into the FLASH memory, and it is when this transfer process occurs that the FLASH draws a large current. Hence, the FLASH write task has been deliberately kept to a short duration so that we can approximate the FLASH write task to have constant current (the FLASH write current) across the whole of the task. A compile-time directive can be used to indicate the acceptable peak current, or target constrained peak current for a given system, with the scheduler arranging the tasks to meet the target constrained peak current. In the case of the scheduler being unable to schedule the tasks given the peak current constraint, then a runtime error can be generated in the same way as a RTOS running out of memory space to manage new instantiated tasks.

```
Task_Struct Task;
Char TaskStack[TASK_STACK_SIZE];
Task_Params taskParams;

// Configure task
Task_Params_init(&taskParams);
taskParams.stack = TaskStack;
taskParams.stackSize = TASK_STACK_SIZE;
taskParams.priority = 1;
taskParams.additional_current = 12;

Task_construct(&Task, taskFxn, &taskParams, NULL);
```

Figure 5. Modified method to instantiate task taskFxn with the addition of new parameter additional_current to inform the task scheduler of the required additional current (in this case 12 mA) that will be drawn when that task runs.

In the specific case of Figure 3, where the FLASH write current occurred at the same time as the BLE ADV transmission, we applied the concept of the scheduler to delay the FLASH write process, and the results are presented in Figure 6. As can be seen in Figure 6, the FLASH write process, point (e), has been delayed in time to separate its current from the BLE ADV transmission, point (g), thus successfully constraining the peak current drawn from the battery.

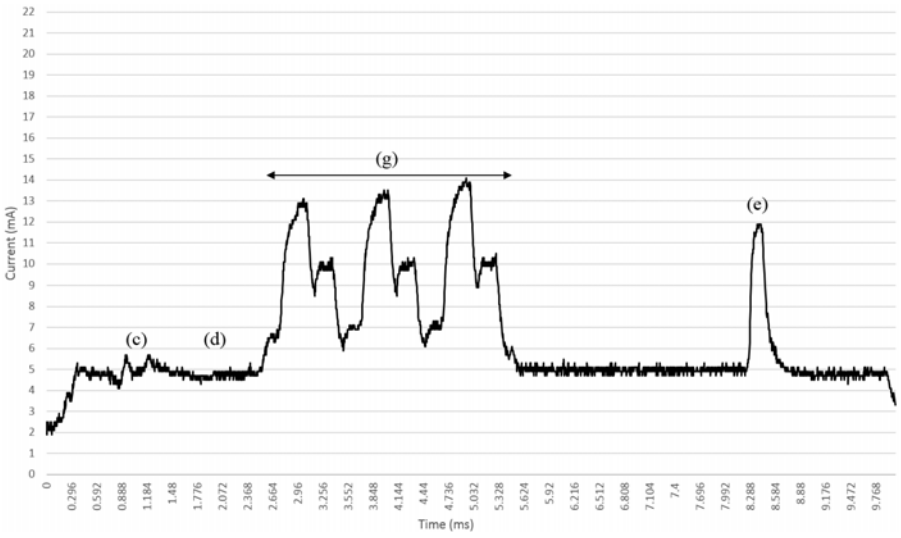

Figure 6. Use of the scheduler concept to delay the FLASH write current to run after the BLE ADV transmission has occurred. Point (e) has now been delayed 5 ms to after the start of point (g), separating the large current into two processes constraining the peak current.

4. Discussion

Through the application scenario that we have experienced, and the results presented in this paper, we have formed a position statement to propose that there are significant advantages to be gained when modifying embedded RTOS task schedulers to further consider the electrical current required to process each task (including peripherals and radio transceivers) when performing the task scheduling process. It is not just a case of scheduling to minimize power, but also to constrain peak power. Clearly, the past work has been very successful regarding deciding when to run tasks in order to minimize the overall electrical current. Such an approach is very useful to enable battery-powered devices to last longer. However, tiny healthcare sensors are constrained by not only their overall available current, but also by their ability to supply large peaks of current due to constrained batteries and voltage regulators. The ability to spread the current drawn over time, rather than to have a large current peak, has significant advantages to simplify power systems that drive wearable healthcare devices.

It is not surprising that in many application scenarios, there are peaks of activity over time (e.g., sampled sensor data) that result in large peaks of current. However, as presented in this paper, the scheduler can delay tasks to a point in the future to constrain peak current while not actually consuming any more current.

The implications of a power-aware scheduler with the ability to constrain electrical current are significant. Not only can such technology be used in smart watches and smart phones, but it can also be used in the control of tasks in smart cars, where considerably large peaks of current can be present.

5. Conclusions

This paper has considered the use of task scheduling in the context of constraining peak electrical current in embedded systems, particularly power sensitive wearable healthcare devices. We have shown that it is possible that such devices can, at a given time, draw peak currents leading to short-term peak demands on batteries and power circuits, ultimately causing supply voltage drops and brownouts, and difficulties in energy harvesting systems.

We propose that an RTOS task scheduler should consider not only the task priority to decide when to schedule tasks, but also the power (current) demands of each task. To avoid large peaks in current, this research work associated an electrical current profile to a software task. Therefore, the objective of the scheduler is to separate over time tasks requiring large current drains rather than schedules to reduce the global power drain. Our task scheduler can delay tasks that have large current profiles so as to constrain the current drawn from the supply within a given limit.

By constraining the peak power current that a device may draw, this then enables simpler power supply designs, energy harvesting, and can reduce electromagnetic interference effects from large pulses of current.

Author Contributions: Conceptualization, R.S.S., B.J., T.H., W.S.H. and D.D.-S.; methodology, N.D., F.S. and J.W.; software, R.S.S.; validation, B.J.; investigation, R.S.S.; writing—original draft preparation, R.S.S.; writing—review and editing, R.S.S., W.S.H., N.D. and T.H.

Funding: This research was partially funded by the UK Engineering and Physical Sciences Research Council (EPSRC), grant number EP/K031910/1, and in part by the Royal Society and National Natural Science Foundation of China International Exchanges 2017 Cost Share (China), grant number IEC\NSFC\170363. The APC was kindly funded by MDPI as part of the special issue on Low-power Wearable Healthcare Sensors.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Singh, S.; Chana, I. A Survey on scheduling in cloud computing: Issues and challenges. *J. Grid Comput.* **2018**, *14*, 217–264. [CrossRef]
2. Bambagini, M.; Marinoni, M.; Aydin, H.; Buttazzo, G. Energy-aware scheduling for real-time systems: A survey. *ACM Trans. Embed. Comput. Syst.* **2016**, *15*, 7. [CrossRef]

3. Lin, X.; Wang, Y.; Xie, Q.; Pedram, M. Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment. *IEEE Trans. Serv. Comput.* **2015**, *8*, 175–186. [CrossRef]
4. Li, X.; Xie, N.; Tian, X. Dynamic voltage-frequency and workload joint scaling power management for energy harvesting multi-core WSN node SoC. *Sensors* **2017**, *17*, 310. [CrossRef] [PubMed]
5. Arora, H.; Sherratt, R.S.; Janko, B.; Harwin, W. Experimental validation of the recovery effect in batteries for wearable sensors and healthcare devices discovering the existence of hidden time constants. *J. Eng.* **2017**, *2017*, 548–556. [CrossRef]
6. Li, Y.; Chen, M.; Dai, W.; Qiu, M. Energy optimization with dynamic task scheduling mobile cloud computing. *IEEE Syst. J.* **2015**, *11*, 96–105. [CrossRef]
7. Liu, J.; Mao, Y.; Zhang, J.; Letaief, K.B. Delay-optimal computation task scheduling for mobile-edge computing systems. In Proceedings of the 2016 IEEE International Symposium on Information Theory (ISIT), Barcelona, Spain, 10–15 July 2016. [CrossRef]
8. Ghofrane, R.; Gharsellaoui Hamza, G.; Samir, B.A. New optimal solutions for real-time scheduling of reconfigurable embedded systems based on neural networks with minimisation of power consumption. *Int. J. Intell. Eng. Inform.* **2018**, *6*. [CrossRef]
9. Li, G.; Wu, Z. Ant colony optimization task scheduling algorithm for SWIM based on load balancing. *Future Internet* **2019**, *11*, 90. [CrossRef]
10. Nguyen, B.H.; Binh, H.T.T.; Anh, T.T.; Son, D.B. Evolutionary algorithms to optimize task scheduling problem for the IoT based bag-of-tasks application in cloud–fog computing environment. *Appl. Sci.* **2019**, *9*, 1730. [CrossRef]
11. Ahmad, S.; Malik, S.; Ullah, I.; Park, D.-H.; Kim, K.; Kim, D. Towards the design of a formal verification and evaluation tool of real-time tasks scheduling of IoT applications. *Sustainability* **2019**, *11*, 204. [CrossRef]
12. Zagan, I.; Găitan, V.G. Hardware RTOS: Custom scheduler implementation based on multiple pipeline registers and MIPS32 architecture. *Electronics* **2019**, *8*, 211. [CrossRef]
13. Singh, P.; Khan, B.; Vidyarthi, A.; Alhelou, H.H.; Siano, P. Energy-aware online non-clairvoyant scheduling using speed scaling with arbitrary power function. *Appl. Sci.* **2019**, *9*, 1467. [CrossRef]
14. Ahmad, S.; Malik, S.; Ullah, I.; Fayaz, M.; Park, D.-H.; Kim, K.; Kim, D. An Adaptive approach based on resource-awareness towards power-efficient real-time periodic task modeling on embedded IoT devices. *Processes* **2018**, *6*, 90. [CrossRef]
15. Chowdhury, P.; Chakrabarti, C. Static task-scheduling algorithms for battery-powered DVS systems. *IEEE Trans. Very Large Scale Integr. Syst.* **2005**, *13*, 226–237. [CrossRef]
16. Fafoutis, X.; Vafeas, A.; Janko, B.; Sherratt, R.S.; Pope, J.; Elsts, A.; Mellios, E.; Hilton, G.; Oikonomou, G.; Piechocki, R.; et al. Designing wearable sensing platforms for healthcare in a residential environment. *EAI Endorsed Trans. Pervasive Health Technol.* **2017**, *17*, e1. [CrossRef]
17. CC2650 SimpleLink Multi-Standard 2.4 GHz Ultra-Low Power Wireless MCU. Available online: http://www.ti.com/product/CC2650 (accessed on 1 May 2019).
18. Bluetooth Archived Specifications. Available online: https://www.bluetooth.com/specifications/archived-specifications (accessed on 1 May 2019).
19. IEEE 802.15.4-2015-IEEE Standard for Low-Rate Wireless Networks. Available online: https://standards.ieee.org/standard/802_15_4-2015.html (accessed on 1 May 2019).
20. MX25U6435F. Available online: http://www.macronix.com/Lists/Datasheet/Attachments/7411/MX25U6435F,%201.8V,%2064Mb,%20v1.5.pdf (accessed on 1 May 2019).
21. ADXL362. Available online: https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL362.pdf (accessed on 1 May 2019).
22. Villeneuve, E.; Harwin, W.; Holderbaum, W.; Sherratt, R.S.; White, R. Signal quality and compactness of a dual-accelerometer system for gyro-free human motion analysis. *IEEE Sens. J.* **2016**, *16*, 6261–6269. [CrossRef]
23. LSM6DSO. Available online: https://www.st.com/resource/en/datasheet/lsm6dso.pdf (accessed on 1 May 2019).
24. TPS62746. Available online: http://www.ti.com/lit/gpn/tps62746 (accessed on 1 May 2019).
25. TI RTOS. Available online: http://downloads.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/tirtos/index.html (accessed on 1 May 2019).

26. Ghamari, M.; Villeneuve, E.; Soltanpur, C.; Khangosstar, J.; Janko, B.; Sherratt, R.S.; Harwin, W. Detailed examination of a packet collision model for bluetooth low energy advertising mode. *IEEE Access* **2018**, *6*, 46066–46073. [CrossRef]
27. Lee, J.K.; Robinovitch, S.N.; Park, E.J. Inertial sensing-based pre-impact detection of falls involving near-fall scenarios. *IEEE Trans. Neural Syst. Rehabil. Eng.* **2014**, *23*, 258–266. [CrossRef] [PubMed]