Energy-Efficient Real-Time Operating Systems: An Approach using Dynamic Frequency Scaling and Worst-Case Execution Time Aware Scheduling

Thomas Jerabek, Benjamin Aigner, Florian Gerstmayer, Jürgen Hausladen

University of Applied Sciences Technikum Wien
Vienna, Austria
Email: {thomas.jerabek, benjamin.aigner, florian.gerstmayer, juergen.hausladen}@technikum-wien.at

***Abstract*—Embedded systems are at the core of many new emerging technologies and applications, deeply integrated into our daily lives. Especially, the demand for battery-powered solutions in consumer-related applications is growing, to support different environments and fields of application. Therefore, energy efficiency measures for embedded systems become even more important. In this paper, a dynamic frequency scaling approach for embedded systems is presented to reduce the overall energy consumption while still meeting time constraints within a real-time operating system. Starting with a general discussion and mathematical derivation along with an elaboration of the state of the art, our concept and implementation is discussed. This includes primarily the developed Worst-Case Execution Time (WCET) aware Earliest Deadline First (EDF) scheduler which is used to dynamically scale the frequency at runtime. Moreover, a use case targeting a real-time smart home application is provided, which was used to evaluate and compare our implementation in regard to it's energy consumption. The respective results are elaborated alongside possible future work and improvements.**

***Keywords*—dynamic frequency scaling; worst-case execution time analysis; energy-efficient computing.**

I. Introduction

Embedded systems are the key to many new technologies, deployed in numerous products and applications, such as smart homes or modern cars. Especially their interconnection and coupling with existing networks – in particular, the Internet – enables new services and functionalities such as sensor fusion, maintenance, firmware updates, or remote access/control.

However, numerous challenges such as safety and security concerns emerge, but also energy consumption needs to be targeted. The latter, is especially of relevance for battery powered devices deployed in constraint environments. By developing new storage technologies, or by further reducing power consumption, battery run- & lifetime can be stretched to reduce the number of recharge cycles or battery replacements. Although devices, e.g., ones used in smart homes, require only a fraction of energy, in sum, the recorded overall power consumption is not to be neglected. Power usage optimizations can thus have a significant impact, facilitating also the development of more maintenance friendly products.

Hence, while originally motivated for general purpose computers and servers, Dynamic Frequency and Voltage Scaling (DFVS) approaches find their way in the embedded systems domain. The idea is to reduce clock frequency and/or voltage, when no computational resources are required, to reduce the overall energy consumption. At the same time, responsiveness and other properties must still be ensured in case computation intensive tasks are raised.

In the context of embedded systems, this is especially of relevance for real time applications which have to deliver results in specified time frames, e.g., to guarantee deadlines. While some being hard ones that have to be met under any circumstances, e.g., X-by-wire systems, as life threatening incidents may be the result, others are soft that may be missed rarely without consequences. The system and its resources are designed to ensure that the deadline of each task is met even in worst-case scenarios. One such critical scenario can be compliance with the Worst-Case Execution Time (WCET), determined either by code instrumentation and runtime measurements or by static analysis. The latter can be done, e.g., by using numerous autonomous tools such as [1] for the respective embedded architecture. However, these worst-case scenarios will scarcely occur in the field.

Therefore, the Central Processing Unit (CPU) will frequently be underutilized, consuming power for doing nothing of purpose for tasks that have already been finished in time before being scheduled again. This circumstance leaves room for improvement, e.g., by applying Dynamic Frequency Scaling (DFS) approaches, to optimize each task according to its deadline. In particular, the CPU's clock frequency can be reduced to a minimum that is required by a task, but which still ensures that all deadlines are met. As a result, the time the CPU is idle and the overall energy consumption can be reduced. Another positive side-effect of reducing the power consumption is, that less heat is generated by the device which directly influences the mechanical design. This in turn can make the difference for the need of a passive or active cooling system, further reducing costs and mean time to failure. However, for this approach, schedulers are required which not only take the task's deadlines into consideration but also their WCET to set the clock frequency accordingly depending on the current workload to prevent deadline violations. In this context, also

 ISBN: 978-1-61208-585-2

numerous requirements regarding the system's architecture and peripherals have to be anticipated. For instance, separate clock domains are necessary to prevent errors in clock sensitive communication channels, e.g., Universal Asynchronous Receiver Transmitter (UART), that are caused by frequency variations. In this paper, the challenges and opportunities of DFS in embedded systems are elaborated. Moreover, the concept and implementation of a Earliest Deadline First scheduler (EDF) is discussed which takes the derived WCETs from [1] into account in this scheduling algorithm. To ease the deployment in existing workflows, an autonomous approach is pursued to reduce entry barriers and enhance usability. The applicability and effectiveness of our approach is shown by a use case targeting a real time smart home application.

The remainder of this paper is structured as follows. In Section II, the theoretical background of this paper is elaborated. Besides the energy consumption model, this also includes the two major strategies in the context of DFS pursued, being race-to-halt and frequency variation. Afterwards, in Section III related work is discussed. Then, in Section IV the developed EDF scheduler is elaborated as well as the architectural prerequisites. Section V discusses the use case and identified problems in regard to DFS and embedded systems. Moreover, the effectiveness of our approach in regard to energy consumption reduction is shown. Finally, in Section VI this paper concludes and gives an outlook regarding future work.

II. Theoretical Background

For the concept and implementation, the effectiveness of energy saving approaches and scheduling algorithms has to be estimated and compared. Therefore, the following paragraphs elaborate the defined energy consumption model and its components which will be used and referenced in this paper. Moreover, possible approaches for energy consumption reduction are discussed.

A. Energy-Model

Generally speaking, power consumption of a CPU is a function of voltage (V), frequency (f), and capacitance (C), as in (1).

$$P = V^2 \ * \ C \ * \ f \tag{1}$$

In other words, depending on the platform's layout, e.g., wire lengths and peripherals, a certain capacitance is present. Hence, optimizations in regard to the printed circuit board design can be made to lower energy consumption. However, one can expect that this results only in marginal improvements. Another key element in this equation is voltage due to its square contribution. Lowering the supply and operating voltage of a device can therefore have a major impact on energy consumption. However, in the context of this paper, embedded devices deployed in a smart home environment are assumed, which already run on lower voltages. Considering that several peripherals also require a minimum supply voltage, further improvements in this field of application are considered marginal in contrast to advances in frequency scaling. Thus, the focus of this paper is primarily on this, last, contributor of the equation. Variations in frequency have a direct proportional impact on energy consumption. For instance, a reduction of frequency by a tenth, yields in an energy reduction of a tenth. In theory, lowering the frequency to zero in (1), results in a power consumption of zero. In reality, this is not feasible due to parasitical effects and leaks on devices, e.g., caused by peripherals and other components, a static power consumption is present. Hence, in (2), a more accurate equation is provided which takes this factor into consideration by adding a constant energy drain.

$$P = (V^2 \ * \ f \ * \ C) \ + \ P_{static} \tag{2}$$

Slight deviations due to different hardware instructions, e.g., more or less high bits that have to be applied on the instruction and data bus are not taken into consideration in our energy consumption model.

Based on (2), several power consumption levels can be derived, depending on the frequency used. Besides zero, the lower bound is derived from the platform dependent minimum frequency (f_{min}). This frequency depends on peripherals or system requirements, e.g., guaranteed response time. The corresponding equation can be seen in (3). The upper bound is, again dependent on the target platform and its maximum frequency f_{max}, shown in (4). In between these boundaries, several discrete frequencies f_{dsc} are applicable, cf. (5). A continuous frequency spectrum is not possible as neither a phase locked loop component does provide that functionality nor do certain peripherals support it, e.g., UART.

$$P_{min} = (V^2 \ * \ f_{min} \ * \ C) \ + \ P_{static} \tag{3}$$

$$P_{max} = (V^2 \ * \ f_{max} \ * \ C) \ + \ P_{static} \tag{4}$$

$$P_{dsc} = (V^2 \ * \ f_{dsc} \ * \ C) \ + \ P_{static} \tag{5}$$

B. Approaches

In regard to how energy consumption can be reduced, two major strategies [2] are prevalent, which are elaborated more in detail in the following paragraphs.

Race-to-Halt: In a race-to-halt strategy, calculations are performed with maximum frequency, so that the result is available as soon as possible. Afterwards, the CPU switches to the idle state which operates at the minimum frequency, until the respective deadline is reached. Hence, energy consumption consists of two parts. A certain amount of time t_1, using the maximum frequency and thus, considering (4), results in maximum power consumption, and the time t_2, while in idle state which has minimum power consumption according to (3). Equation (6) describes this coherence.

$$E_{avg} = t_1 \ * \ P_{max} \ + \ t_2 \ * \ P_{min} \tag{6}$$

Dynamic Frequency Scaling: In case of the dynamic frequency scaling [3] approach, the WCET and the deadline

of a task are used as parameter to modify the operating system frequency of the processor. It is designed for systems that provide high peak performance when needed and in turn dynamically reduces the power consumption by decreasing the operating frequency of the CPU whenever possible. When a task is scheduled, the processor's highest frequency is multiplied with the rate calculated from the previous parameters, which results in the frequency of the processor. The lowered processor frequency in turn stretches the execution of the task that still meets the required deadlines. This principle can be seen in Figure 1. In contrast to (6), the sum of a certain amount

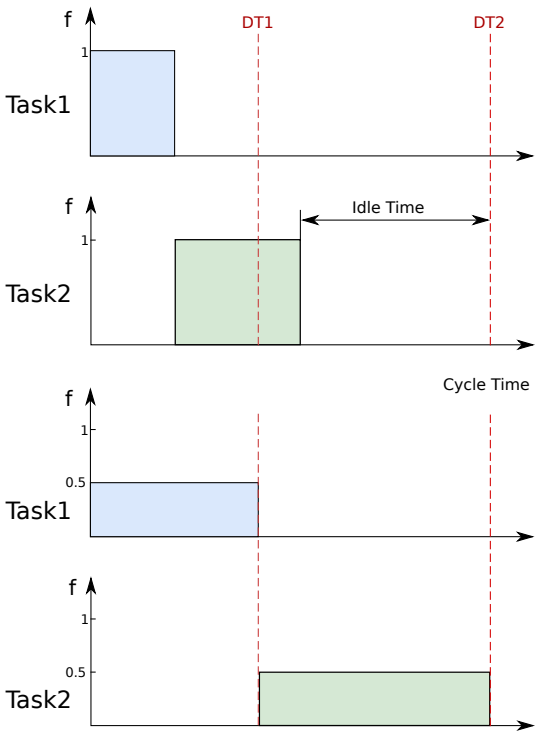

Figure 1. DFS - Concept.

of time t_1 and t_2, using the discrete frequency and thus, considering (5), results in reduced power consumption. Equation (7) describes this coherence.

$$E_{avg} = (t_1 + t_2) * P_{dsc} \quad (7)$$

III. Related Work

In general, power-aware scheduling [4][5] is a significant key strategy for battery powered real-time embedded systems to reduce the power consumption and extend battery life time. Real-time embedded systems consist of a set of tasks that are scheduled in a specific order, leaving time slots in which the processor is underutilized but still draining the battery. Therefore, modern processors offer a range of sleep modes, to reduce power consumption. However, due to periodic task execution (often on an average of a few milliseconds), these are generally not applicable. In contrast, the time required to first enter and later on leaving the sleep mode is in the order of tens of milliseconds which can easily be in the order of a magnitude of a task's period. Hence, sleep modes are often not suitable for real-time embedded systems. However, reducing the frequency of the processor for the execution while still satisfying the given deadlines as done with DFS is a feasible solution, which leads to a remarkable reduction of the energy consumption.

A general overview on the energy-efficiency of DFS in resource constraint embedded devices, as well as desktop and server grade processors is conducted in [6]. Cho et al. [3] proposes a different approach where frequency and voltage is scaled down when processing external peripherals. The idea is to save energy during the time waiting for the results from the external peripheral. Shin et al. [7] developed a tool that converts existing programs into a low-energy version based upon the remaining WCET. The tool automatically retrieves the appropriate locations in the program where voltage scaling mechanisms can be inserted.

There exist different scheduling algorithms such as earliest deadline first that can be used in conjunction with DFS. An evaluation of several scheduling algorithms is provided in [8]. The authors conducted an exhaustive simulation in which they retrieved the most important parameters that affect the energy consumption. In [9], an optimized version of the EDF algorithm has been proposed, that further improves energy savings by about 28% to the original algorithm. [10] extended the DFS approach for the use with multicore processors.

IV. Implementation

The implementation of our DFS concept needs modifications in the Real-Time Operating System (RTOS) components (A) task management, (B) scheduling, (C) context switching, and (D) application tasks, as described subsequently.

A. Task Management

In real-time systems, it is common to split the application into tasks where each of them is responsible for a certain functionality. Each task has an application specific priority and stack size that needs to be configured by the developer. This information along with runtime parameters such as the current task state or associated event flags is maintained in a data structure called Task Control Block (TCB). Every task requires a TCB, which is only accessed by the real-time kernel and never by the application code due to consistency reasons. For our implementation, an extension of the TCB is necessary to preserve the task's (a) WCET and (b) deadline. These two parameters are provided as 32-bit unsigned integer and specified in microseconds. Thus, the maximum value for a WCET or deadline can be 71.58 minutes which is precise enough for most applications. The deadline is set to zero in case of an uncritical task where no deadline is given. The implemented scheduler needs to consider this as well as an unspecified WCET (equals zero) to avoid starvation. In addition, the task control block is extended by two parameters to define (c) the deadline type, and (d) the time when a task's

 ISBN: 978-1-61208-585-2

deadline will be reached. Parameter (c) defines if a deadline occurs cyclic (e.g., every 10 ms) or depends on a certain event (e.g., 10 ms after falling edge on a designated input pin). Parameter (d) defines the absolute deadline as time value in order to create a reference point for the scheduler. In summary, these four arguments are added to the TCB for scheduling purposes.

B. Scheduling

The scheduler selects a process from the ready list to execute, which is determined by scheduling criteria. Our scheduling algorithm is based on an earliest deadline first approach with additional knowledge about the WCET as shown in Figure 2. The scheduler creates a temporary task control block for the Task To Schedule (TTS) and sets its deadline initially to zero. Afterwards, an iteration loop on the list of Ready Tasks (RT) evaluates the task with the earliest deadline. In the case of equal deadlines of two or more tasks, the algorithm chooses the task with the longest WCET. If there are no deadlines specified, a task selection is not possible with this algorithm, therefore, a priority based scheduling will be performed. At this point, the scheduling is completed and the RTOS continues with the dynamic frequency scaling algorithm followed by the context switch and the clock adjustment.

```
1  WCET_AWARE_EDF_Sched()
2    DISABLE_INTERRUPTS()
3    TCB TTS
4    TTS.curDeadline = 0
5    i = 0
6    while i < MAX_TASKS
7      if RT[i].curDeadline < TTS.curDeadline
8        TTS = RT[i]
9      else if RT[i].curDeadline == TTS.curDeadline
10       if TTS.WCET < RT[i].WCET
11         TTS = RT[i]
12   i = i +1
13
14   if TTScurDeadline == NONE
15     TTS = PRIO_Sched()
16
17   dfs_div = DFS(TTS)
18   if(dfs_div < 1)
19     dfs_div = 1
20     error
21   OS_TASK_SW()
22   cpu_clk = Set_CPU_CLK(dfs_div)
23   Adj_PB_CLK(cpu_clk)
24   SysTickUpdate(cpu_clk/TickRate)
25
26   ENABLE_INTERRUPTS()
27   return
```

Figure 2. WCET-Aware EDF Scheduler.

C. Context Switching

The dynamic frequency scaling algorithm is executed right before the actual context switch, as shown in Figure 2. This algorithm (cf. Figure 3) starts with the maximum clock divider and evaluates if the frequency scaling does not violate any deadlines.

Since the ratio between execution time and processor frequency is considered linear, a temporary WCET for the task to schedule can be calculated by multiplying its WCET with the

```
1  DFS(TTS)
2    dfs_flag = true
3    dfs_div = MAX_DFS_DIV
4    sort(RT,DEADLINE_ASC)
5    while dfs_div > 0
6      WCETtemp = dfs_div*TTS.WCET + AO
7      if WCETtemp < TTS.curDeadline
8        i = 0
9        dfs_flag = true
10       while  i < MAX_TASKS
11         WCETtemp = WCETtemp + RT[i].WCET + AO
12         if WCETtemp >= RT[i].curDeadline
13           DFS_Flag = false
14           break // config not possible!
15         i = i + 1
16
17       if dfs_flag == true // config works!
18         return dfs_div
19
20     dfs_div = dfs_div - 1
21   return dfs_div
```

Figure 3. Dynamic Frequency Scaling.

clock divider. An Administrative Overhead (AO) for context switch, scheduling etc. is added. If this DFS dependent WCET is below the task's deadline, the DFS divider is applicable. However, it is mandatory to consider that all subsequent tasks need to meet their deadlines too. By adding the WCET and AO of the task with the next larger deadline, one can verify its compliance. This step is repeated for all ready tasks. For this iteration (cf. Figure 3 Line 9-14), the algorithm assumes that the RT array is sorted by ascending deadlines, which is done at the very beginning of the function. Once a divider is applicable for the entire system, the algorithm returns this value. A divider of one means, the deadlines can only be reached without downscaling. An early detection of deadline violation in the RTOS is also done by returning a value of zero, which means that at least one deadline cannot be fulfilled even without reducing the CPU clock. This can be used to settle appropriate measures, e.g., implement a callback function to bring the system to a safe and secure state.

D. Deadline Handling by Application Tasks

Since an absolute value for each deadline is necessary to calculate the DFS divider, the function in Figure 4 needs to be implemented. It updates the task's deadline either on a periodic base where the deadline is added to the current value or event triggered where the deadline is set according to the current system time. To guarantee a proper functionality of the scheduler and DFS algorithm, this function is called after each task completion (e.g., task cycle done) by the application's tasks themselves. For event triggered use cases, it is mandatory to update the deadline at the occurrence of influencing events either by the task itself or another task. Therefore, the developer is responsible to trigger the deadline updates.

E. Deployment

The selected real-time operating system is Micrium uCOS-III because it enables the necessary kernel modifications and supports the chosen target hardware, the Infineon XMC4500-F100K1024 microcontroller. The microcontroller features an

 ISBN: 978-1-61208-585-2

```
updateDeadline(TCB Task)
  if Task.DeadlineType == PERIODIC
    tempDeadline = Task.curDeadline + Task.Deadline
  else // deadline event triggered
    tempDeadline = getSysTime_uS() + Task.Deadline
  Task.curDeadline = tempDeadline
  return
```

Figure 4. Deadline Handler.

ARM Cortex-M4 processor core and allows a system clock division up to 256 in single steps. Thus, one can adjust the system clock from 120 MHz down to 468.75 kHz, which is ideal to test the presented concept. In previous work, the OTAWA Stack and Worst-case execution time Analysis (OSWA) tool [1] was developed to evaluate a tasks WCET and is therefore used for the evaluation in Section V. This analysis tool is provided by our cloud-based Integrated Development Environment (cloud-IDE), as presented in [11].

V. Evaluation

For the evaluation of our concept, a dedicated use case with a generic implementation of a gateway for smart home devices was used. The requirements for the use case are as follows:

- Bluetooth Low Energy (BLE) to ZigBee Gateway
- Soft-deadlines for usability reasons

According to [12], a device with human-machine interaction requires that the maximum response time is $\leq$ 100 ms to be experienced as reacting instantaneously by the operator. In order to provide a well-founded maximum execution time limit for the implementation of this use case, the following boundary conditions are defined:

- 60 ms reaction time, sensor from or to ZigBee gateway
- 13 ms reaction time, BLE gateway from or to actuator

The ZigBee latency is assumed and based on the work of Baviskar et al. in [13], where an average latency of 58 ms was measured. Moreover, an additional safety margin is added, resulting in an assumption of 60 ms latency from the ZigBee device to the gateway. A BLE network is usually fast regarding the time required for the connection establishment and subsequent data transfer. According to [14], BLE needs approximately 3 ms for these tasks. Concerning a relay-based actuator, an additional time overhead of 10 ms for the relay operation is required.

When using the maximum 100 ms reaction time (T_{maxRT}), the subtraction of the delay times for each hardware interface results in a maximum execution time limit of 27 ms for the defined task structure, as shown in Equation (8).

$$\begin{aligned} T_{tasklimit} &= T_{maxRT} - T_{ZigBee} - T_{BLE} \\ T_{tasklimit} &= 100ms - 60ms - 13ms = \mathbf{27ms} \end{aligned} \quad (8)$$

Each radio frequency interface (BLE and ZigBee) utilizes two different tasks, one for receive operations and one for transmit operations, as depicted in Figure 5. Data is shared via queues, which are used to transport data from the RF-controller to the bridge task and vice-versa. The bridge task is used to determine any operations that are required to share data between these two links. In addition, a processing task can be used to insert data, e.g., a gateway condition.

To eliminate influencing external factors on the DFS measurements, the BLE and ZigBee connections are replaced by a physical loopback via the corresponding UART RX/TX pins, as shown in Figure 5.

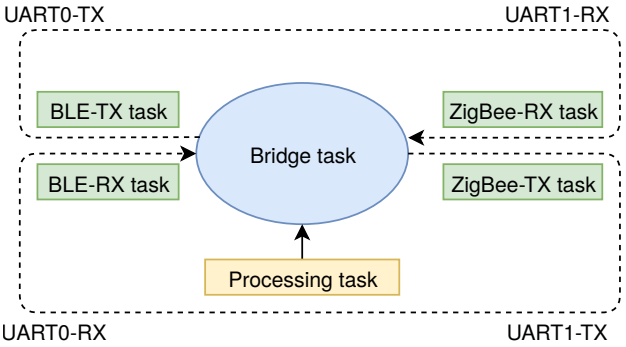

Figure 5. Task Structure with Loopback.

A. Results

According to Equation (8), the maximum execution time limit for receiving, processing and transmission is 27 ms. Thus, the deadlines of involved tasks are estimated in order to do not exceed this limit. In particular, the segmentation is as follows: 7.5 ms for the receive task, 12 ms for the bridge task, and another 7.5 ms for the transmit task. The deadlines and types are described in the application, as shown in Table I.

Table I. Task parameters.

Task	Deadline type	Deadline	WCET
BLE RX	once	7.5 ms	1.21 ms
BLE TX	once	7.5 ms	1.26 ms
ZigBee RX	once	7.5 ms	1.16 ms
ZigBee TX	once	7.5 ms	1.20 ms
Bridge	once	12 ms	0.96 ms
Processing	periodic	1000 ms	1.13 ms

The tasks WCET already include an administrative overhead [1] and were estimated with the OSWA tool. The WCET analysis also considers internal RTOS functions that are relevant. However, no program flow information, e.g., loop bounds are available for RTOS internal sections. The evaluation of these is the most sophisticated part of the entire analysis because loop bounds cannot be directly derived from the RTOS source code, since they depend on application specific parameters. Once the WCET analysis is successfully accomplished for each task as can be seen in Table I, the results are imported into the application.

Depending on the deployed peripherals, it is not feasible to use any CPU frequency because it can prevent specific peripheral clock configurations that are necessary for external devices or certain tasks. In case that the peripheral clock is derived from the CPU clock, the DFS implementation is rather limited as the peripherals cannot operate at their

 ISBN: 978-1-61208-585-2

appropriate frequency. Therefore, this use case only considers the maximum CPU frequency (120 MHz) and the half CPU frequency (60 MHz), since the peripheral clock can remain on the same 60 MHz independent of the selected frequency.

Table II shows the three evaluated configurations where the CPU clock is either set to 120 MHz, variable (DFS) or set to 60 MHz. The measurements show, that in our use case the power consumption can be reduced by 19.01% using the DFS implementation. This is very close to the case where the CPU clock is generally set to the lower frequency. However, there are situations where the RTOS switches the CPU frequency to 120 MHz to avoid deadline violations. Operating only on 60 MHz is no option as it would lead to deadline violations.

TABLE II. COMPARISON OF POWER CONSUMPTION.

CPU frequency	current	power consumption
120 MHz	151.5 mA	499.95 mW
variable (DFS)	122.7 mA	404.91 mW
60 MHz	121.5 mA	400.95 mW

Our DFS implementation closes the gap between power efficiency and performance for this application. Other use cases may leave even more space for optimization, while others cannot be optimized at all because the peripheral clock configuration would be too restrictive for DFS.

VI. CONCLUSION

In this paper, the implementation and evaluation of a WCET aware earliest deadline first scheduler for uCOS-III is presented. The principle of dynamic frequency scaling is applied therein to achieve a power consumption reduction for real-time embedded systems applications.

The proof-of-concept and benefits are shown in an exemplary use case, by reference of a smart home application that implements a ZigBee to BLE gateway requiring certain responsiveness. By the use of the implemented WCET aware scheduler, a power reduction of 19% was achieved while still meeting the given deadlines.

The process of deriving the WCET bounds was accomplished with the aid of our previously implemented and in [1] presented OSWA tool. Its integration in state-of-the-art IDEs provides the possibility to offer the herein presented DFS implementation with ease to numerous developers due to it's seamless integration in prevalent workflows. Therefore, a widespread field of possible applications is derived as a developer neither requires in-depth knowledge on DFS nor its implementation.

The contribution of the herein presented approach leads to the conclusion, that a major energy consumption reduction is achieved through the application of the DFS algorithm while preserving the full capabilities of the system. This is especially beneficial for low-power devices through:

- Reduced hardware costs by reducing battery capacity
- Dynamic adjustment of CPU utilization determined by deadline constraints
- Real-time and power constraints are met in different usage scenarios

Future work is geared towards the possible problem of deadline inversion, similar to the well known priority inversion problem. In particular, considering task 1 having a disproportional or no deadline, task 2 might have to wait for task 1 to complete, resulting in a miss of the deadline of task 2. This problem might be solved by the application of mechanisms, such as priority inheritance, e.g., by inheriting deadlines.

ACKNOWLEDGMENT

This work has been conducted in the context of the public funded R&D projects Toolbox for Rapid Design of Smart Homes & Assistive Technologies (ToRaDes) and Software Analysis Toolbox (SAT) managed by the Vienna City Council MA23.

REFERENCES

[1] T. Jerabek and M. Horauer, "Static worst-case execution time analysis tool development for embedded systems software," in *9th International Conference on Dependability (DEPEND)*, Jul. 24-28, 2016, pp. 7–14.

[2] L. Tan and Z. Chen, "Slow down or halt: Saving the optimal energy for scalable hpc systems," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '15. New York, NY, USA: ACM, 2015, pp. 241–244.

[3] K. M. Cho, C. H. Liang, J. Y. Huang, and C. S. Yang, "Design and implementation of a general purpose power-saving scheduling algorithm for embedded systems," in *2011 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, Sept 2011, pp. 1–5.

[4] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo, "Energy-aware scheduling for real-time systems: A survey," in *ACM Trans. Embed. Comput. Syst.* ACM, 2016, pp. 7:1–7:34.

[5] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg, "Fast: Frequency-aware static timing analysis," in *Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS)*. IEEE Computer Society, 2003, pp. 40–51.

[6] E. Le Sueur and G. Heiser, "Slow down or sleep, that is the question," in *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*. USENIX Association, 2011, pp. 1–6.

[7] D. Shin, J. Kim, and S. Lee, "Intra-task voltage scheduling for low-energy hard real-time applications," in *IEEE Design Test of Computers*, March 2001, pp. 20–30.

[8] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, 2001, pp. 89–102.

[9] H. El Ghor and E. Aggoune, "Energy efficient scheduler of aperiodic jobs for real-time embedded systems," in *International Journal of Automation and Computing*. Springer, 2016, pp. 1–11.

[10] J. L. March, S. Petit, J. Sahuquillo, H. Hassan, and J. Duato, "Dynamic wcet estimation for real-time multicore embedded systems supporting dvfs," in *2014 IEEE International Conference on High Performance Computing and Communications (HPCC)*, Aug 2014, pp. 27–33.

[11] J. Hausladen, B. Pohn, and M. Horauer, "A cloud-based approach to development of embedded systems software," in *2015 ASME/IEEE International Conference on Mechatronic and Embedded Systems and Applications*, Aug. 2-5, 2015., pp. 1–7.

[12] R. B. Miller, "Response time in man-computer conversational transactions," in *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, ser. AFIPS '68 (Fall, part I). New York, NY, USA: ACM, 1968, pp. 267–277.

[13] J. Baviskar, A. Mulla, M. Upadhye, J. Desai, and A. Bhovad, "Performance analysis of zigbee based real time home automation system," in *2015 International Conference on Communication, Information Computing Technology (ICCICT)*, Jan 2015, pp. 1–6.

[14] A. J. Jara et al., "Evaluation of bluetooth low energy capabilities for tele-mobile monitoring in home-care." in *Journal of Universal Computer Science*, vol. 19, no. 9, 2013, pp. 1219–1241.

 ISBN: 978-1-61208-585-2