Enhanced virtual release advancing algorithm for real-time task scheduling*

D. Duy and K. Tanaka

School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa, Japan

ABSTRACT
In real-time task scheduling, response time and time complexity are two of important requirements that draw many attentions. Virtual release advancing (VRA) [Tanaka, K. (2015, June). Virtual release advancing for earlier deadlines. ACM SIGBED Review, 12(3), 28–31] is an effective technique for shorter response times in the Earliest Deadline First scheduling [Liu, C. L., & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the Association for Computing Machinery, 20(1), 46–61], but not adaptive to precise systems due to its high time complexity. In order to mitigate the time complexity, a new technique, called enhanced VRA, is presented in this paper. Applied to the Total Bandwidth Server context, the enhanced technique significantly improves the time complexity while guaranteeing the responsiveness and schedulability. The technique is implemented on an ITRON real-time operating system running on an ARM Cortex-A9 processing core with a field programmable gate array. With supporting of accelerator hardware, the new algorithm shows that the maximum additional runtime overhead per tick is reduced by up to 30% compared with that of the original (software) VRA.

1. Introduction
Among substantial requirements of real-time task scheduling, response time is a factor that inspires many researchers. For the aim of improving response time for aperiodic tasks, the following techniques were introduced: the adaptive total bandwidth server (Adaptive TBS) (Tanaka, 2013a), the adaptive Earliest Deadline First (EDF) (Tanaka, 2013b), and the virtual release advancing (VRA) (Tanaka, 2015). These techniques are based on the EDF algorithm (Liu & Layland, 1973) in the TBS context (Spuri & Buttazzo, 1994). With their own approach, the techniques have been trying to assign the target aperiodic task an earlier deadline that may lead to an earlier scheduling under EDF algorithm. In the case of VRA, an earlier deadline can be obtained by introducing a virtual release time earlier than task’s actual release time in the deadline calculation. As a result, this technique...
achieves a substantial improvement in terms of task’s responsiveness. However, there still exists an obstacle to its applicability; that is, the time complexity is relatively high. The runtime overhead of the technique is up to 0.9% of the tick length of 1 millisecond (Tanaka, 2015). The high time complexity can prevent the technique from being applied to precise real-time systems the tick of which may be less than 0.1 millisecond in length.

Making the VRA technique reasonable to be applied to precise real-time systems becomes the goal of the enhanced virtual release advancing (EVRA) technique, which is proposed in this paper. EVRA acquires earlier deadlines, which are equivalent to those by the original VRA, with relatively low runtime overhead. This algorithm introduces boundary deadlines as limits of advancing and then tries to satisfy these limits to obtain suitable virtual deadlines. Besides, this algorithm successfully reduces the loop counts that mainly cause the runtime overhead in the original algorithm. The contribution of the proposed algorithm is that the time complexity is significantly reduced while the schedulability and responsiveness are still guaranteed. In addition to the algorithm, an accelerator hardware supporting the EVRA in deadline calculation is targeted to further mitigate the runtime overhead.

The rest of this paper is composed as follows: Related work is in Section 2 where the original VRA algorithm is presented. Section 3 is the description of the proposed method in details. Next, Section 4 shows the implementation of the accelerator hardware for the EVRA. The proposed method is then evaluated in Section 5. Finally, the paper is concluded in Section 6.

2. Related work

During recent decades, scheduling algorithms for mixture systems of both periodic and aperiodic tasks have been engaging to researchers in the field of embedded systems. There are a number of scheduling algorithms proposed for such systems. They are basically classified into two groups: fixed-priority servers and dynamic-priority servers. For the former group of fixed priority, the fundamental scheduling strategy is based on rate monotonic algorithm (Liu & Layland, 1973) where the higher-priority tasks have benefit of lower jitters. Priority Exchange (Lehoczky, Sha, & Strosnider, 1987), Sporadic Server (Sprunt, Sha, & Lehoczky, 1989), and Slack Stealing (Lehoczky & Ramos-Thue, 1992) were typically known as fixed-priority servers. On the other hand, the latter group of dynamic-priority servers are developed based-on the EDF scheduling (Liu & Layland, 1973) which is known as an optimal scheduling algorithm on uniprocessor systems for maintaining the schedulability under the processor utilization of 100%. Dynamic Priority Exchange (Spuri & Buttazzo, 1994), TBS (Spuri & Buttazzo, 1994), and Constant Bandwidth Server (Abeni & Buttazzo, 1998) are representatives of this group.

These introduced algorithms are aimed to shorten the response time of the aperiodic requests while keeping the implementation complexity reasonable. TBS shows remarkable achievements on responsiveness with relatively low implementation complexity. In TBS context, hard tasks are supposed to be invoked periodically and soft tasks are supposed to be invoked aperiodically. While the periodic tasks’ deadlines are explicitly designed to be equal to their period ending, the aperiodic tasks’ tentative deadlines are calculated using Equation (1). In this equation, $k$ means the $k$th aperiodic task, $d_k$ is the absolute deadline of the $k$th task, $r_k$ is release (arrival) time of the $k$th task, $d_{k-1}$ is the absolute deadline of
the \((k - 1)\)th (previous) task, \(C_k\) is worst-case execution time (WCET) of the \(k\)th task, and \(U_s\) is the bandwidth of the server.

\[
d_k = \max (r_k, d_{k-1}) + \frac{C_k}{U_s}
\]  

\(1\)

VRA (Tanaka, 2015) is developed based-on the TBS server and the EDF algorithm. The VRA algorithm is aimed to improve the responsiveness of the TBS server by obtaining earlier deadlines. The technique tries to virtually and retroactively move a release time backward to the past while maintaining the past schedule.

In the VRA, \(r_k\) is substituted by a virtual release time \(v r_k\) that is obtained by the virtual advancing. Since, \(v r_k\) is always the same as or earlier than \(r_k\), earlier deadlines can be calculated using Equation (2).

\[
d_k = \max (v r_k, d_{k-1}) + \frac{C_k}{U_s}.
\]  

\(2\)

The earlier deadline then leads to an earlier scheduling of the target task under the EDF algorithm. Figure 1 shows an example of the technique with two periodic tasks. The first periodic task, \(t_1\), has a period \(T_1 = 3\) and an execution time \(C_1 = 1\). Similarly, the second one, \(t_2\), has \(T_2 = 4\) and \(C_2 = 2\). The processor utilization by the periodic tasks is \(U_p = 1/3 + 1/2 = 5/6\) and then the bandwidth of the TBS server is \(U_s = 1 - 5/6 = 1/6\). It is assumed that there is an aperiodic task entering the system at \(t = 6\) with its execution time of 1. According to the original TBS scheduling, the aperiodic task has deadline of \(t = 12\), finishing time at \(t = 8\), and response time of 2.

In this case, by applying the VRA technique, an earlier virtual release time for the aperiodic task is set up at \(t = 2\) as indicated by an upward dashed arrow. According to this new release time, the deadline calculation in TBS would give a deadline at \(t = 8\), four ticks earlier than the original deadline. This new deadline, as depicted by a downward dashed arrow, is earlier than that of the third instance of \(t_1\) (its deadline at \(t = 9\)). Under the EDF algorithm, the target aperiodic task is hence immediately executed ahead of the periodic instance and finishes sooner at \(t = 7\) with the response time of 1.

![Figure 1. Example of VRA.](image-url)
In the VRA algorithm, an earlier virtual release time for the target aperiodic task is obtained by using the advancing technique that is set up based-on three following factors: the previous aperiodic task’s deadline, the last empty slot, and the maximum previously-used deadline (Tanaka, 2015). These three factors are known as the limits of how long the release time can be advanced. It is stated that moving release time backward over any one of these limits is either ineffective or changing the past schedules (Tanaka, 2015). The advancing technique can be here simply described as follows: Initiating at the original release time \( r_k \), the virtual release time \( v_r_k \) is checked whether it satisfied the limits or not. Being satisfied means that \( v_r_k \) is larger than any of limits. If it is satisfied, \( v_r_k \) is move backward to the past by one slot. Otherwise, the advancing stops. This checking is repeated slot by slot until any of limits is reached.

In spite of its significant improvement on the response time of the target tasks, the virtual release advancing technique has a problem with its high time complexity. Due to the high runtime overhead, the algorithm has just been evaluated on systems with a relatively long tick length of 1 millisecond. In order to be adaptive to more precise systems, an algorithm with light-weight complexity is considered.

3. Enhanced virtual release advancing

3.1. Causes of overhead in the original algorithm

As introduced above, the VRA technique checks limit factors slot by slot from the target task’s arrival time \( r_k \) backward to the past. The checking loop stops and the virtual release time as well as the virtual deadline is obtained when one of the limit factors is reached. On this checking method, the algorithm increases runtime overhead as follows. Firstly, the algorithm checks two factors, previous deadline and last empty slot, in every iteration. However, there are actually only one previous deadline and one last empty slot and they remain constant during the algorithm’s execution. Thus, it is desirable to check only once for each factor. Secondly, consecutive slots spent by the same instance of a task are checked one by one. It is obviously unnecessary since such slots have the same associated deadline. Once again, it is desirable to check only the representative one of slots. Thirdly, the limit how long past the release time is advanced is not defined.

By observing these sources of runtime overhead, the EVRA technique is introduced to overcome these inefficiencies.

3.2. Deadline-base advancing

The EVRA technique uses a direct approach, called deadline-base advancing, to obtain earlier deadlines. It is different from the original VRA where the release times are moved backward to the past and then earlier deadlines are calculated. In the EVRA, the absolute deadline of the target task is advanced directly in the limit of boundary deadlines. The rest of this section explains in details these boundary deadlines and their related definitions including check-bounding slot and representative slots. After that we will show clearly how the deadline-base advancing works in Section 3.3.
3.2.1. Boundary deadlines

In the original VRA algorithm, the virtual release time is advanced in the limit of three factors: the previous aperiodic task’s deadline, the last empty slot, and the maximum previously-used deadline. The first two limits are compared with a tentative virtual release time for each loop iteration of the advancing. Differently in the proposed algorithm, these two factors are each employed as $v_r_k$ in Equation (2) to calculate two corresponding deadlines. These corresponding deadlines are then dealt with outside of the loop.

For example in Figure 2, there are two periodic tasks and two aperiodic tasks are observed. The second aperiodic task as the target task here arrives at $t = 9$ and has $C_k = 1$. The previous aperiodic task’s deadline ($d_k$) is at $t = 6$ and the last empty slot is slot 7. Using the previous deadline ($t = 6$) and the tick time ($t = 8$) just after the empty slot as replacements for $v_r_k$ in Equation (2), two corresponding deadlines ($v_d_k'$ and $v_d_k''$) for the target task are calculated to be at $t = 12$ and $t = 14$, respectively.

These two deadlines are now regarded as two boundary deadlines. Together with another boundary deadline determined by the check-bounding slot which is coming in the next subsection, these boundary deadlines become the limits of advancing of the proposed algorithm. In order to satisfy the limits of the algorithm, the expected virtual deadline for the target task hence has to be greater than or equal to all of these boundary deadlines.

3.2.2. Check-bounding slot

For ease of understanding, let us define:

- $\tau_{\text{max}}$: one of periodic tasks that has the maximum period;
- $T_{\text{max}}$: the period of $\tau_{\text{max}}$ (the maximum period);
- $l_{s_{\text{max}}}$: the starting time of the last instance of $\tau_{\text{max}}$.

Under the EDF algorithm, the associated deadline of $l_{s_{\text{max}}}$-th slot is greater than or equal to the associated deadlines of slots before it. That is, we have:

$$\forall x < l_{s_{\text{max}}}, \ dl[x] \leq dl[l_{s_{\text{max}}}], \quad (3)$$

![Figure 2](image.png)

**Figure 2.** Example of boundary deadlines.
where $d[t][x]$ and $d[t][ls_{\text{max}}]$ are the associated deadlines of the $x$th slot and the $ls_{\text{max}}$th slot, respectively.

Figure 3 shows an illustration for $\tau_{\text{max}}$ and $ls_{\text{max}}$. In this example, $\tau_2$ corresponds to $\tau_{\text{max}}$. The current aperiodic task’s release time is 9, and then the last instance of $\tau_{\text{max}}$ is the instance executed from slot 7 to slot 9. Therefore, $ls_{\text{max}}$ is identified as $t = 7$. Note that the maximum previously-used deadline which has to be determined at each iteration of the advancing is one of the limits in the original VRA. It can be derived from Equation (3) that the associated deadline of the $ls_{\text{max}}$th slot is greater than or equal to that of previous slots. As a result, $ls_{\text{max}}$th slot is considered as the check-bounding slot when tracking for the maximum previously-used deadline. The EVRA algorithm therefore considers this slot’s associated deadline as the third boundary deadline of the advancing.

### 3.2.3. Representative slots of instances

As discussed above, checking all of slots of the same instance one by one is unnecessary. In the EVRA algorithm, the first slot of each instance is definitely chosen as the representative one. In Figure 4, for instance, referring to the target task’s release time ($r_k$) at $t = 4$ slot 0 and slot 1 are considered as representative slots for $\tau_1$’s first instance and $\tau_2$’s first instance, respectively.

For this example, the original VRA algorithm checks all slots from $t = 3$ backward to $t = 0$, that is, slot 2, 1, and 0. However, only slot 1 and slot 0 are checked for advancing in the EVRA algorithm. It is obviously more efficient in terms of the check loop count and memory use. In cases where task’s execution is separated into several portions by preemption, each portion is considered as an instance. Note that in the EVRA algorithm, the associated deadlines of representative slots are used for deadline advancing.

### 3.3. Algorithm for EVRA

The whole EVRA algorithm is shown in Algorithm 1. In the algorithm, $r_k$ and $v_k$ are the real and virtual release times of the target aperiodic task, respectively. $vd_k$ is the expected virtual deadline for the target task. $C_k$ is the worse-case execution time of the target task. $U_t$ is the bandwidth of TBS server. $d_{k-1}$ is the deadline of the $(k - 1)$th (previous) aperiodic task. For other variables, $last\_empty$ is the slot number of the last empty slot (which is

![Figure 3. Example of check-bounding slot ($ls_{\text{max}}$).](image-url)
assumed to be $-1$ for no empty slot); $n$ and $l_{\text{max}}$ are the number of released instances and the starting time of the last instance of $t_{\text{max}}$ (Section 3.2.2), respectively. $dl[m]$ and $S[n]$ are the arrays saving past slots’ associated deadlines and released instances’ starting time. Then, three temporary deadlines ($vd'_k$, $vd''_k$, $vd'''_k$) defined in Section 3.2 are calculated corresponding to the previous deadline, last empty slot, and last starting time of $t_{\text{max}}$. When advancing to the past, ‘max_gen’ holds the maximum associated deadline of traced $S[n]$ elements. Under the mentioned condition of the advancing, the expected virtual deadline $vd_k$ cannot be earlier than $vd'_k$, $vd''_k$, $vd'''_k$, or $max_gen$.

The algorithm’s execution can be made clear by an illustration in Figure 5 where the algorithm is applied for the target aperiodic task in the situation of tasks shown in Figure 1 above. Here, when the scheduler is invoked at time $t = 6$ as the target task is released, the algorithm is executed. According to this situation, at the beginning of the algorithm’s execution, $vd_k$ (the tentative virtual deadline for the target task) is initialized to the virtual deadline using the original TBS’s formula at line 2 where the original release time $r_k$ is applied. $vd_k$ then results in value of 12.

Then three temporary deadlines are calculated from line 4 to line 14. The first temporary deadlines ($vd'_k$) corresponding to the previous deadline is calculated outside the loop at

![Figure 4](image1)

*Figure 4.* Example of representative slot of instances.

![Figure 5](image2)

*Figure 5.* Example of Algorithm’s execution for deadline advancing.
line 5. Since the previous deadline \((d_{k-1})\) is assumed to be 0 for no previous aperiodic task, \(v_d^k\) then results in value of 6. The if statement from line 6 to line 7 says that if the tentative virtual deadline is earlier than or equal to \(v_d^k\), the tentative virtual deadline is set to \(v_d^k\), and the algorithm finishes. The second temporary deadline of empty slot \((v_d^\prime_k)\) is also calculated outside of the loop at line 12. For this example, \(v_d^\prime_k\) gets value of 6 due to no empty slot in this situation. The third temporary deadline, \(v_d^{\prime\prime}\), is then calculated at line 14 as the \(ls_{\text{max}}\)th slot’s associated deadline. \(v_d^{\prime\prime}\) is calculated to be 8 for this example where \(ls_{\text{max}} = 4\) and \(dl[ls_{\text{max}}] = 8\). Combining these temporary deadlines, a variable \(\text{bound}\) at line 15 presents the check-bounding deadline of the algorithm, which is of 8 for this example.

**Algorithm 1** The EVRA algorithm

```plaintext
1: /*Definition*/
2: \(v_d^k = r_k + C_k/U_s\)
3: /*Limit of k-1-th deadline*/
4: \(v_d^\prime_k = d_k - 1\)
5: \(v_d^\prime_k = v_d^\prime_k + C_k/U_s\)
6: if \(v_d^k \leq v_d^\prime_k\) then
7: \(v_d^k = v_d^\prime_k\)
8: goto End
9: endif
10: /*Limit of last empty slot*/
11: \(v_d^{\prime\prime}_k = \text{last_empty} + 1\)
12: \(v_d^{\prime\prime}_k = v_d^{\prime\prime}_k + C_k/U_s\)
13: /*Limit of previously-used slots*/
14: \(v_d^{\prime\prime\prime}_k = dl[ls_{\text{max}}]\)
15: \(\text{bound} = \max(v_d^k, v_d^{\prime}_k, v_d^{\prime\prime}_k, v_d^{\prime\prime\prime}_k)\)
16: \(i = n - 1\)
17: \(\text{max_dl} = 0\)
18: while \(v_d^k > \text{bound}\) do
19: \(v_r_k = S[i]\)
20: if \(\text{max_dl} < dl[v_r_k]\) then
21: \(\text{max_dl} = dl[v_r_k]\)
22: endif
23: if \(v_d^k \leq \text{max_dl}\) then
24: break
25: endif
26: \(v_d^k = v_r_k + C_k/U_s\)
27: if \(v_d^k \leq \text{max_dl}\) then
28: \(v_d^k = \text{max_dl}\)
29: break
30: else
31: \(i = i - 1\)
32: endif
33: endwhile
34: Label: End
```

Next, to satisfy the limit of previously used maximum deadline, for each of the traced \(S[n]\) elements, the corresponding \(v_d^k\) is compared to \(\text{max_dl}\) from line 18 to line 33. The value of \(v_d^k\) obtained after the loop execution satisfies conditions of \(\text{bound}\) and \(\text{max_dl}\) and therefore it is the expected deadline.

For this example, temporary variables for the loop execution are initiated as: \(i = 3\) for 4 released instances \((n = 4)\) and \(\text{max_dl} = 0\). Next, since condition \(v_d^k > \text{bound}\) is satisfied for \(12 > 8\), the advancing loop starts to setting \(v_r_k = S[3] = 4\) (the starting time of the third released instance). Then, the if-state from line 20 to line 22 updates \(\text{max_dl}\) to be 8 \((dl[4] = 8)\). Since \(v_d^k > \text{max_dl}\) \((12 > 8)\), the algorithm’s execution continues to update
$vd_k$ to be 10 at line 26. The updated value of $vd_k$ remains greater than $max_{dl}$ and the if-state at line 27 is not satisfied. Consequently, the else-state at line 31 is executed to decrease variable $i$ by 1 and the advancing loop repeats.

For the second repeat of the advancing loop, with the similar procedure, considered variables result in $i=2$, $vr_k = S[2] = 3$, $max_{dl} = 8$, and $vd_k = 9$. With these values, the advancing loop continuously repeat for the third time of $i=1$. At this time of iteration, $vr_k = S[1] = 1$, $max_{dl} = 8$, and $vd_k = 7$. The condition of if-state at line 23 is now passed, then $vd_k$ is set to be equal to $max_{dl}$ (or 8) and the advancing loop stops. After three times of deadline advancing, the algorithm finishes with the tentative virtual deadline of 8.

Note that, if the original VRA is applied for this example, the virtual release time is moved backward to the past from $t=6$ until $t=2$. Due to moving slot by slot, the advancing takes four times of repeat for $vk_k$ to reach value of 2, which leads to the virtual deadline of 8. This example hence shows the effectiveness of EVRA algorithm with reduction in the number of loop count compared to the original VRA.

### 3.4. Enhanced points of the EVRA algorithm

In Section 3.1, main sources of considerable runtime overhead in the original VRA are repeated checking of limits. The deadline-base advancing of the EVRA algorithm is significant to reduce the sources of runtime overhead as follows:

- First, definition of boundary deadlines allows the algorithm to process two limits of previous task’s deadline and last empty slot outside of advancing loop. This helps to reduce the influence of repeated checking.
- Check-bounding slot and its associated deadline can be used to solve the limit how long past the deadline advancing is performed.
- Representative slots allow advancing deadlines instance by instance. This is beneficial in not only reducing loop count, but also memory costs. In the original VRA, in order to advance slot by slot, the associated deadlines of all of used slots have to be recorded by array $dl[m]$ sized by $m$. The new algorithm, on the other hand, uses only the first slots of instances recorded by array $S[n]$ sized by $n$. Actually, $n$ (the number of released instances) is in most cases smaller than $m$ (the number of past slots). Therefore, advancing over $n$ is obviously more efficient than over $m$.

### 4. Implementation of EVRA algorithm

The EVRA algorithm is implemented first in software system using ITRON real-time operating system (Tanaka, 2006; TRON Association, 2002). The software implementation is to examine the performance as well as the applicability of the EVRA algorithm in comparison with other algorithms including the original TBS, original VRA, improved TBS (Buttazzo & Caccamo, 1999; Buttazzo & Sensini, 1999), and CBS. The software implementation of EVRA algorithm was originally presented at the KSE2016 (Duy & Tanaka, 2016). As an extension of the software implementation, this work includes designing an accelerator hardware to further reduce runtime overhead of deadline calculation. The designed hardware is connected to an ARM Cortex-A9 processing core integrated on field programmable gate
array (FPGA) (Cortex-A9 Technical Reference Manual, 2010; Zynq-7000 Technical Reference Manual, 2015) and controlled by the ITRON real-time operating system.

**4.1. Overview of the software system**

ITRON is a specification of embedded operating systems and widely used in industry and research fields (TRON Association, 2002). This is a flexible operating system, which supports various mechanisms to manage tasks including task scheduling, so that system developers can implement their own design conveniently. The original scheduler in ITRON is based on static priority, where each task of application is assigned a fixed priority. In implementation of this research, the scheduler is replaced by our own dynamic-priority one.

Figure 6 shows a flow of the scheduling system including an additional component, Deadline Calculation. The Deadline Calculation mainly performs the proposed algorithm to calculate the absolute deadlines for the target aperiodic tasks. Task execution and system recorder are two important parts which support the algorithm. The other parts are basic system mechanisms consisting of system initiation, time management, scheduling, and dispatching.

As shown in the flow, after initiating system, the time management increases the time tick. The system then starts observing of task activation. If there is no task activation, the system continues to task scheduling followed by task dispatching. Whereas if there are tasks requested at the tick, the Deadline Calculation is activated to calculate the absolute deadline for the requested task. The requested task is then inserted into the ready queue and ready to be scheduled for execution. The priorities of ready tasks are dynamic and assigned based on the EDF algorithm. During the any task’s execution, there are two

![Figure 6. Overview of the implementation on ITRON.](image-url)
events happening: One is a time-out event which occurs at the end of each system tick. The other is a task-finishing event which occurs at the time when a task finishes its execution. These events are illustrated in Figure 7. When any one of the events occurs, the system recorder is activated to record information for running the EVRA algorithm. The whole processing flow is repeated tick by tick under the control of the time management.

4.2. The accelerator hardware

In order to increase reduction rate of runtime overhead due to amount of time spent to execute the algorithm, an accelerator hardware is integrated to support deadline calculation. The block diagram of the accelerator hardware and its connections to CPU are depicted in Figure 8. In this system, the CPU is an ARM Cortex-A9 processing core and communicates with the accelerator through input/output (GPIO) ports.

The I/O registers are deployed to connect directly to the GPIO ports of the CPU for mutual communication. According to the communication protocol (described in Section 4.3), each of I/O registers is employed to temporarily store information as follows:

- ACK_IN: storing an ACK request which has been sent by the CPU.
- ACK_OUT: storing an ACK response which the hardware needs to send to the CPU.
- DATA_IN: storing input data which has been sent by the CPU.
- DATA_OUT: storing output data which the hardware needs to send to the CPU.

Figure 7. An example of task’s execution.

Figure 8. Block diagram of the accelerator hardware.
These registers can be accessed specifically by the operating system through their fixedly assigned address in the physical address map.

Besides the I/O registers, the accelerator hardware includes a Register Area which plays a role of an internal memory. This area is separated into two segments: local data and global data. The local segment is composed of registers where local data are stored. The accelerator only can internally access this segment. On the other hand, the global segment consists of indexed registers where global data of the system are stored. These global registers are designed for global accesses by register IDs as described in Section 4.3. Therefore, both the accelerator and the CPU can access the global segment.

A main part of the accelerator is the Calculating and Data Control unit, which performs deadline calculations and is controlled by a State Control unit. The state control unit is implemented as a state machine the state transitions of which are displayed in the diagram in Figure 9. There are totally six states as follows:

- S0 is as the initial state of the system.
- S1 and S2 are processing states. In these states, the hardware performs processing of the deadline advancing.
- S3, S4 and S5 are stop states in which a calculated virtual deadline is fixed and available in the designated register.

As shown in the diagram, the state changes from one to another under the conditions denoted as C1, C2, C3, C4, and the reset signal (‘RS’). The conditions are decided corresponding to the checking points at line 6, 18, 23, and 27, respectively, of the Algorithm 1 (in Section 3.3). The system always initiates at S0, where the machine is initially set up, and then changes to S1 unconditionally (or no condition, ‘nc’). Next, under conditions C2, C3, and C4, the system’s state changes between S1 and S2 for deadline advancing. The deadline advancing is repeated until one of the conditions ‘C1 = 1’, ‘C2 = 0’, ‘C3 = 1’, and ‘C4 = 1’ is satisfied and then one of the stop states is reached.

The virtual deadline is valid to be accessed only at a stop state and an ACK response is sent to the CPU on the ACK_OUT connection. After receiving a response signal, the CPU then reads the expected virtual deadline at the appropriate register by sending a corresponding command described in Section 4.3 to the accelerator.

![Figure 9. State transition diagram.](image-url)
The accelerator implementation has an advantage that condition checking and deadline calculation can be done in parallel. Let us consider conditions C1 and C2 in Figure 9 for example, these are dealt with in the same state S1, taking only one clock cycle. This is the case for conditions C3 and C4 in S2. The Calculating and Data Control block in Figure 8 performs the necessary calculations to obtain the expected deadline in several steps. In each step, the system has to manipulate varied data and values for algorithm’s execution. While software implementation would process the data one by one, all the manipulation can be done in a clock cycle by the accelerator hardware. Consequently, with the same amount of information, the hardware processing takes less clock cycles than the implementation by software only. This is the way on which the accelerator improves the performance of the system.

4.3. Communication between CPU and accelerator

The communications between the CPU and the accelerator are set up by a request-response protocol using 16-bit request commands and 32-bit data buses. In the connection, two 16-bit ports, named GPIO and GPIO2, are used for the CPU to send requests to and receive responses from the accelerator. Similarly for mutual data transmissions, two 32-bit ports GPIO_1 and GPIO2_1 are employed. A typical communication procedure to request the accelerator is as follows: The CPU first sends a necessary data to the data port (GPIO2_1), and then sends a corresponding request command to the request port (GPIO2). After waiting for a response (at port GPIO) acknowledging of valid data from the accelerator, the CPU can load needed data at port GPIO_1.

A request command is structured of 16-bit information as shown in Figure 10. The least significant 8 bits compose a register ID to specify a target register in the global segment of the Register Area. The other 8 bits form an operation code (op-code) to determine exactly which operation is requested. The op-code consists of:

- The 8th bit (L/S) specifies loading/storing operation. It is set to 1 for loading and clear to 0 for storing.
- The 9th bit (RS) is reset operation. This activates with value of 1.
- The 10th bit (SE) is for scheduling operations (or enabling the deadline calculation). This activates with the value of 1. When one of reset or scheduling operations is active, the least significant 8 bits are regarded as ‘don’t care’.
- The op-code field includes reserved bits from 11 to 15. The current hardware implementation limits the operations to four types: storing, loading, reset, and scheduling. By exploiting the reserved field, additional operations are extended for supporting other operating system functions.

| Operation code | Register ID |
|----------------|-------------|
| 15...........11 | 10 9 8 7...0 | Reserved SE RS L/S | Register ID (L/S only) |

Figure 10. Structure of a request command.
An example procedure of enabling the deadline calculation by accelerator hardware is composed as in Code 1. This procedure uses ARM instruction set (Zynq-7000 Technical Reference Manual, 2015) where \texttt{mov}, \texttt{str}, and \texttt{ldr} are moving, storing and loading instructions on registers, respectively, while \texttt{cmp} and \texttt{bne} are comparing and branch instructions.

| Code 1: Enabling hardware |
|---------------------------|
| 1: \texttt{mov r1, #16h0400} //prepare reset command |
| 2: \texttt{mov r2, #16h0200} //prepare enabling command |
| 3: \texttt{str r1, GPIO2} //reset the hardware |
| 4: \texttt{str r2, GPIO2} //start the hardware |
| 5: \texttt{CK: ldr r1, GPIO} //load response |
| 6: \texttt{cmp r1, #0400} //check the response |
| 7: \texttt{bne CK} //repeat if not equal |
| 8: \texttt{ldr r1, GPIO_1} //load the data (result) |

Here, the scheduling and reset commands are defined as 16’h0400 and 16’h0200, respectively. To enable the accelerator hardware for deadline calculation, the operating system first moves the reset and enabling commands successively to the registers at lines 1 and 2 and then stores these commands to the designated command ports at lines 3 and 4. This step is corresponding to sending requests to the accelerator hardware. According to the order of sending requests, the accelerator is reset before starting deadline calculation. The operating system is then waiting for the response from the hardware for the available data by \texttt{CK} loop at lines from 5 to 7. The response code #0400 is supposed as a well-done calculation response. Receiving the response code #0400 at port GPIO means the expected deadline is valid and the operating system can load the data at the data port as code at line 8.

5. Evaluation

5.1. Evaluation criteria

The performance of the system is evaluated based-on two target criteria: response time and additional overhead. There are two types of overhead taken into account:

- \textbf{The maximum additional overhead per tick}: Given \( O_t[i] \) and \( O_e[i] \) are the additional overhead by recording necessary information and executing the scheduling algorithm during the \( i \)th tick period, respectively. Then the total additional overhead in the \( i \)th tick (\( O_{tot}[i] \)) is calculated as:

\[
O_{tot}[i] = O_t[i] + O_e[i].
\]  
(4)

(For tick periods during which any tasks are not executed, the corresponding \( O_e \) is equal to 0.) The maximum additional overhead per tick (\( O_{max/tick} \)) is then calculated as:

\[
O_{max/tick} = \max (O_{tot}[i]).
\]  
(5)

- \textbf{The total execution overhead}: Given \( O_{e,k} \) is the additional overhead by executing the scheduling algorithm for the \( k \)th aperiodic task’s activation. Then, the total execution overhead (\( O_{e,tot} \)) for \( n \) (1 \( \leq k \leq n \)) times of activation of aperiodic tasks is calculated
in accumulation as:

\[ O_{e,\text{tot}} = \sum_{k=1}^{n} O_{e,k} \]  

(6)

5.2. Evaluation of software implementation

The evaluation of software implementation is conducted to evaluate the performance of the EVRA algorithm in comparison with other algorithms including the original TBS, original VRA, Improved TBS, and CBS algorithms. A simulation environment for evaluation is set up based-on the instruction set of the Cortex-A9 processor (Cortex-A9 Technical Reference Manual, 2010). Performances on response time and additional overhead are observed within the period of 100,000 ticks for task sets of both periodic and aperiodic tasks. The utilization of the periodic tasks \( U_p \) changes from 60% to 95% with utilization interval of 5%, while that of the aperiodic tasks is about 2%. The utilization of aperiodic servers \( U_j \) are all assumed to be \( 1 - U_p \).

Figures 11 and 12 display the performance on improving the responsiveness and the maximum additional overhead, respectively. As shown in Figure 11, the EVRA algorithm (Soft-EVRA) can achieve the same responsiveness as the original VRA (TBS+VRA). The EVRA algorithm clearly generates shorter response times than the original TBS. However, on the criterion of responsiveness it still performs worse than the improved TBS (TB*/TB(n)) and CBS algorithms.

On the criterion of runtime overhead, the EVRA algorithm shows significant improvement over both the original VRA and the improved TBS. This achievement satisfies the primary goal of reducing the runtime overhead of algorithm’s execution. Nevertheless, the proposed algorithm still has higher time complexity than the TBS and CBS algorithms. Overall, the CBS is the most competitive on both responsiveness and time complexity.

Figure 11. Performance on improving the responsiveness.
5.3. Evaluation of hardware implementation

The accelerator hardware here is evaluated for its contribution to further reduce the runtime overhead, especially compared to the software implementation (without the accelerator). The evaluation environment is built with the Zynq7000 device (Zynq-7000 Technical Reference Manual, 2015) on the ZedBoard FPGA KIT (Zedboard Hardware User’s Guide, 2012). The ARM core in Zynq7000 runs at the frequency of 666 MHz. The accelerator hardware is implemented in the programmable logic part of the Zynq7000. The additional overhead is derived by observing the number of instruction cycles which can be obtained by accessing the cycle count register (PMCCNTR) in the ARM Cortex-A9 core. In this evaluation, the execution of EVRA with the accelerator hardware (Hard-EVRA) is compared with the software implementations of the original VRA algorithm, the original TBS, and the EVRA without accelerator (Soft-EVRA).

Five periodic task sets and three aperiodic task sets are created for the evaluation. Each periodic task set consists of six periodic tasks while each aperiodic task set has 10 points of task activation. Periodic and aperiodic task sets are randomly mixed to compose five scenarios. Table 1 shows as an example one of scenarios where the WCET and period of periodic tasks and 10 entering points of aperiodic tasks are listed. In each scenario, the total processor utilization (Up), which is different among scenarios, does not include the runtime overhead of the operating system execution such as context switching and scheduling.

Table 1. An example one of scenarios with periodic and aperiodic task sets.

| Periodic tasks (Up = 67.5%) | Aperiodic task Entering points |
|-----------------------------|--------------------------------|
| Task name | WCET | Period | Points |
| PT SK_1 | 3 | 27 | 90, 167, 271, 385, 497, 603, 724, 847, 965, 1116 |
| PT SK_2 | 3 | 30 |
| PT SK_3 | 4 | 28 |
| PT SK_4 | 5 | 40 |
| PT SK_5 | 4 | 56 |
| PT SK_6 | 5 | 40 |

Figure 12. Performance on improving the runtime overhead.
Table 2 shows results of absolute deadlines and response times for aperiodic tasks over 10 entering times. Here, the outputs of Hard-EVRA are compared with those of the original TBS. Note that, in terms of response times, the Soft/Hard-EVRA and the original VRA generate the same results. In the table, the second column shows the number of time when there are earlier deadlines calculated. How much earlier these deadlines are calculated is then displayed by the average number of ticks in the third column. The forth column shows the number of time when the response times of aperiodic task are shortened. Definitely, Hard-EVRA continues guaranteeing a better responsiveness than the original TBS. For example, in scenario 2, eight among 10 points of aperiodic task activation have earlier deadlines. Some of the earlier deadlines then lead to shorter response times.

For the runtime overhead, Table 3 shows results of the original VRA, Soft-EVRA and Hard-EVRA. Overall, all of the scenarios show that EVRA alleviates the runtime overhead compared to the original VRA. Furthermore, the EVRA with the accelerator hardware exhibits better results compared to those of the case without the hardware support. In details, the results indicate that the maximum runtime overhead per tick of the Hard-EVRA is lower by 30% and by 13% on average than that of the original VRA and Soft-EVRA, respectively.

The total overhead throughout the observation period shown in Table 4 affirms substantial improvement by EVRA, especially with the hardware support. That is, Hard-EVRA has 34% and 9% on average lower total execution overhead compared to the original VRA and Soft-EVRA, respectively.

Note that the response times evaluated in Table 2 are in ticks. Actually, the exact response times in cycles supports the Hard-EVRA in terms of mitigation of the overhead.

| Scenario | # of earlier deadlines | # of earlier tick of deadline | # of shorter response time |
|----------|------------------------|-------------------------------|---------------------------|
| 1        | 4/10                   | 10                            | 1/10                      |
| 2        | 8/10                   | 16                            | 3/10                      |
| 3        | 7/10                   | 6                             | 1/10                      |
| 4        | 6/10                   | 12                            | 2/10                      |
| 5        | 9/10                   | 16                            | 2/10                      |

Table 3. Results of maximum overhead per tick (cycles).

| Scenario | Original VRA | Soft-EVRA | Hard-EVRA |
|----------|--------------|-----------|-----------|
| 1        | 178          | 135       | 126       |
| 2        | 177          | 146       | 114       |
| 3        | 183          | 150       | 124       |
| 4        | 154          | 122       | 105       |
| 5        | 149          | 124       | 113       |

Table 4. Results of total execution overhead (cycles).

| Scenario | Original VRA | Soft-EVRA | Hard-EVRA |
|----------|--------------|-----------|-----------|
| 1        | 593          | 432       | 406       |
| 2        | 711          | 459       | 409       |
| 3        | 554          | 420       | 389       |
| 4        | 523          | 440       | 381       |
| 5        | 658          | 483       | 439       |
As an example, Table 5 displays response times in cycles for the 2nd activation of an aperiodic task in the scenario in Table 1. In this example, the original TBS, original VRA, Soft-EVRA and Hard-EVRA have the same response time of 13 ($= 180 - 167$) ticks, since all of them generates the same deadline. On the other hand, offset cycles present the response times in cycles in tick 180. The target task finishes earliest by the original TBS with the lowest overhead, while the Hard-EVRA with lower overhead than the Soft-EVRA (and even the original VRA) leads to earlier finishing.

In addition, the results in Table 3 show that the maximum overhead per tick is not a substantial amount. In the Scenario 1, where the Hard-EVRA experiences higher maximum overhead per tick than in the other scenarios, it takes only 0.08% of a tick length. This fact suggests high feasibility of applying hardware implementation of EVRA algorithm to high-precision real-time systems.

6. Conclusion

In this paper, an EVRA algorithm was proposed. This is an enhancement of VRA in terms of runtime overhead. The runtime overhead is significantly reduced while the schedulability and responsiveness are guaranteed.

In addition, an implementation of the accelerator hardware supporting the EVRA was described. With the support of accelerator hardware, the EVRA algorithm can help to reduce the runtime overhead of the original VRA by up to 30%. The resulting small runtime overhead per tick of 0.08% indicates that accelerator hardware of the EVRA algorithm has potential for being applied to high-precision real-time embedded systems.

The currently EVRA algorithm calculates tentative deadlines based on WCETs of tasks. Unfortunately, WCETs are pessimistically estimated and in most cases longer than the actual execution times. The overestimation prevents the scheduling algorithm from giving the best schedule. In the near future, it is expected to combine the EVRA algorithm with the adaptive TBS (Tanaka, 2013a), which uses predicted shorter execution times in deadline calculations, to further improve the response time.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work was supported by JSPS KAKENHI [Grant Number JP 15K00073].
Notes on contributors

**Doan Duy** received his B.E. degree from the University of Information Technology, Vietnam (2013), and his M.S. from Japan Advanced Institute of Science and Technology (JAIST), Japan (2016). Currently, he is in his Ph.D. program at JAIST. His research interests are computer architecture, real-time embedded systems, and VLSI design.

**Kiyofumi Tanaka** received his B.S., M.S., and Ph.D. degrees from the University of Tokyo in 1995, 1997, and 2000, respectively. His research interests are computer architecture, operating systems, and real-time embedded systems. He is a member of IEEE, ACM, IPSJ, and IEICE.

References

Abeni, L., & Buttazzo, G. C. (1998). Integrating multimedia application in hard real-time systems. Proceeding of real-time systems symposium (Cat. No.98CB36279, pp. 4–13), IEEE, Madrid, Spain. Cortex-A9 Technical Reference Manual. (2010). ARM. Retrieved from http://infocenter.arm.com/help/topic/com.arm.doc.ddi0388f

Buttazzo, G. C., & Caccamo, M. (1999, February). Minimizing aperiodic response times in a firm real-time environment. *IEEE Transactions on Software Engineering*, 25(1), 22–32.

Buttazzo, G. C., & Sensini, F. (1999, October). Optimal deadline assignment for scheduling soft aperiodic tasks in hard real-time environments. *IEEE Transactions on Computers*, 48(10), 1035–1052.

Duy, D., & Tanaka, K. (2016). Enhanced virtual release advancing for EDF-based scheduling on precise real-time systems. Proceeding of the 2016 Eighth International Conference on Knowledge and Systems Engineering (KSE2016), IEEE, Hanoi, Vietnam, 43–48.

ITRON4.0 Speciation. (2002). *ITRON Committee, TRON ASSOCIATION(Japan)*. Version 4.00.00.

Lehoczky, J. P., & Ramos-Thue, S. (1992). An optimal algorithm for scheduling soft aperiodic tasks in fixed-priority preemptive systems. In *IEEE real-time systems symposium* (pp. 110–123). Vienna: IEEE Computer Society.

Lehoczky, J. P., Sha, L., & Strosnider, J. K. (1987). Enhanced aperiodic responsiveness in hard real-time environment. In *IEEE real-time systems symposium* (pp. 261–270). San Jose: IEEE Computer Society.

Liu, C. L., & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1), 46–61.

Sprunt, B., Sha, L., & Lehoczky, J. (1989). Aperiodic task scheduling for hard-real-time systems. *Journal of Real-Time Systems*, 1(1), 27–60.

Spuri, M., & Buttazzo, G. C. (1994). Efficient aperiodic service under earliest deadline first scheduling. In Proceedings of real-time systems symposium, IEEE, San Juan, Puerto Rico, USA, (pp. 2–11).

Tanaka, K. (2006). Real-time operating system kernel for multithreaded processor. In Proceeding of IEEE international workshop on innovative architecture for future generation high-performance processors and systems (IWIA’06), International Workshop on, Kohaha Coast, HI, USA, (pp. 91–99).

Tanaka, K. (2013a). Adaptive total bandwidth server: Using predictive execution time. In Proceeding of international embedded systems symposium (IESS), Springer Berlin Heidelberg, Paderborn, Germany, (pp. 250–261).

Tanaka, K. (2013b, December). Adaptive EDF: Using predictive execution time. *ACM SIGBED Review*, 10(4), 41–44.

Tanaka, K. (2015, June). Virtual release advancing for earlier deadlines. *ACM SIGBED Review*, 12(3), 28–31.

Zynq-7000 All Programmable SoC Technical Reference Manual. (2015). *Xilinx*. Version 1.1 August 1st.

Zynq™ Evaluation and Development Hardware User’s Guide. (2012). *AVNET*. Version 1.1 August 1st. Retrieved from https://reference.digilentinc.com/reference/programmable-logic/zedboard/start