An Efficient Virtual System Clock for the Wireless Raspberry Pi Computer Platform

Diego L. C. Dutra, Edilson C. Corrêa, Claudio L. Amorim

February 2016

Abstract

The use of Dynamic Voltage and Frequency Scaling (DVFS) by Energy-Efficient (EE) computer systems considerably increases the requirements regarding the design of efficient system clocks. On the one hand, the operation of a system clock must support the independent operating frequencies of the processor core units, the dynamic migration of the running processes between the processors core units, and the use of synchronization and time interpolation techniques to maintain the accuracy of the system clock. On the other hand, an efficient system clock has to minimize the overhead of its own operation, aiming at energy efficiency of EE computer systems.

In this paper, we present the design and evaluation of the RVEC virtual system clock for the EE Wireless Raspberry Pi (RasPi) platform. In the RasPi platform, the use of DVFS for reducing the energy consumption hinders the direct use of the cycle count (CCNT) of the ARM11 processor core for building an efficient system clock. Therefore, a distinct feature of RVEC is to obviate this obstacle, such that it can make use of the CCNT circuit for precise and accurate time measurements, concurrently with the use of DVFS by the operating system of the ARM11 processor core. Specifically, this paper presents the design and experimental evaluation of an implementation of the RVEC virtual system clock in the Linux kernel of the RasPi platform with DVFS. Our experimental results validate the RVEC virtual system clock as an efficient system clock for the EE RasPi platform that runs the Linux operating system.

1 Introduction

In computer systems, system clocks provide the time measurements that are fundamental to the development and assessment of computer programs. Similarly, distributed applications, including financial transactions, distributed databases, and multiplayer games depend on accurate and reliable system clocks. Conventionally, system clocks guarantee correct and precise time measurements by using a standard technique based on a time-invariant cycle count, such as the

Time Stamp Counter (TSC). Recently, Energy-Efficient (EE) computer systems that use multicore processors with Dynamic Voltage and Frequency Scaling (DVFS) [1] required some advances in the design of system clocks. Specifically, a system clock now must support the independent operating frequencies of the processor core units and the dynamic migration of the running processes between the processor core units. However, such a design of a system clock also faces the negative effects of DVFS on its efficiency, which can potentially reduce both the precision and accuracy of its time measurements.

To counteract the negative effects of DVFS, a system clock frequently maintains its accuracy by using a synchronization mechanism, such as NTP [2], the Flooding Time Synchronization Protocol (FTSP) [3] or the Timing-sync Protocol for Sensor Networks (TPSN) [4]. In addition, to increase the accuracy of a system clock, it is also necessary to resort to some complementary technique that implements a time interpolation method in the operating system. However, both types of techniques have not yet provided an efficient system clock with the property of strictly increasing and precise (SIP) time counting [1] [5]. Moreover, the design of an efficient system aiming at energy efficiency of EE computer systems clock must also minimize the cost of operation of the system clock.

In this paper, we use the Raspberry Pi [6] (RasPi) computer as a representative platform of EE computer systems[]. The RasPi platform is based on the 32-bit ARM1176JZF-S [7] processor core and is used especially in embedded computing systems [8] and Wireless Sensor Networks (WSNs) [9, 10, 11] for remote sensing and monitoring applications [12, 13] because of the energy efficiency of the ARM processors. The sources of the timing events of the RasPi platform consist of periodic interrupt scheduling and the system time counter (STC). STC is a free-running counter that operates at 1 MHz, which the Linux operating system uses to implement a time interpolation technique. However, the use of DVFS prevents the direct use of the CPU cycle count (CCNT) for building an efficient system clock to perform the time measurements (Broomhead et al. [14] report equivalent issue for the x86 architecture cycle counter (TSC)).

This paper extends the material presented in our previous publication to the RasPi platform [15] by extending the experimental results of the RVEC SIP property [5] of both RVEC and the CCNT counter of the ARM11 processor and presents an evaluation of the impacts of the NTP synchronization on the system clocks of the RaspPi platform. The contributions of this paper are the following:

- Development of an RVEC implementation in the Linux kernel of the RasPi platform;
- Assessment of the SIP property of both RVEC and the ARM CCNT counter;

[1] A system clock with the SIP property assures that two consecutive clock readings, T_1 and T_2, will return time measurements $T_2 > T_1$ for any time interval between the two readings

- Validation of the RVEC virtual system clock as an efficient system clock for a computer system that runs Linux;
- Evaluation of the impacts of use of NTP on the system clocks of the RasPi platform.

The remainder of this paper is organized as follows. Section 2 presents related work. In Section 3, we describe the organization and integration of RVEC to protect it from the time drifts that appear in the Linux kernel of the RasPi computer. In Section 4, we present results of our experimental evaluation of RVEC and the analysis of NTP synchronization on the RaspPi platform. Finally, in Section 5, we present our conclusion.

2 Related work

Dutra et al. [5] introduced the original design and implementation of RVEC in the Intel Xeon processor family with DVFS. The authors also showed that the nodes of a cluster of computers can remain synchronized globally after each of the cluster nodes initializes its local RVEC by using a remote synchronization client-server algorithm.

Veitch et al. [16] developed the RADClock system clock, which is built on existing system clocks or cycles of time counters such as the TSC. RADClock provides information on the global time and the absolute global time for synchronization of computer network nodes. However, RADClock depends on NTP for periodic resynchronization and also has limited applicability to multicore processors because of its implementation method.

Tian et al. [17] presented a global clock that uses the TSC as the base clock circuit together with a remote clock synchronization algorithm, which is similar to NTP. Because of the direct use of TSC, such a global clock cannot work properly with processors that have multiple core units or use DVFS.

Souza et al. [18] proposed an auxiliary synchronization network that uses a remote pulse generator to ensure that all nodes of a cluster of computers simultaneously receive the remote clock pulse, which each node uses to update its local clock without involving the operating system. Although the proposed solution guarantees a SIP time count, it depends on dedicated hardware and is not applicable to wireless networks.

With regard to the maintenance of the accuracy of the system clock, the predominant solution is to run the NTP daemon that periodically resynchronizes the system clock by using the remote NTP server [19]. However, under a heavy workload, the execution of the NTP daemon is often delayed, which causes time drifts in the system clock of up to tens of seconds [20]. Moreover, if consecutive readings of the system clock are issued within time intervals shorter than tens of milliseconds the system clock cannot guarantee the SIP property of the time counting. Hong et al. [21] evaluated the accuracy of the system clock based on NTP synchronization by using local GPS hardware that provides an absolute

time reference. The authors reported results that indicate a median error of 2 ms to 5 ms in the system clock by using the standard NTP clients, whereas the work [20] evaluated the NTP clients that cause over-utilization of the CPU. By considering the results of both works, we can infer that the use of NTP synchronization is still less efficient for WSNs in which the stability of wireless links is not guaranteed and the WSN nodes have lower processing capacity; thus, the NTP synchronization cannot ensure the SIP property of the time counting for system clocks used in WSNs.

3 RVEC: A Strictly Increasing and Precise Virtual Clock

Currently, the system clocks of a multicore processor that runs Linux will receive periodic interrupts from the auxiliary circuits of multiple time-invariant cycle count that operate at a lower frequency than that of the multicore processor. As a result, the time measurements of the system clocks will have lower precision and its accuracy will be determined by the degree of stability of the time intervals between the hardware interrupts. In practice, the system clocks are exposed to other interrupts that occur within variable time intervals, so, the efficiency of system clocks will depend on the use of a clock synchronization mechanism, such as NTP or TPSN.

Other complementary solutions for improving the accuracy of system clocks implement a time interpolation technique in the operating system by using the cycle count of the faster processor, without guaranteeing the SIP property for the system clocks. Furthermore, the periodic execution of a synchronization mechanism counteracts the energy efficiency of an EE computing system, whereas the handling of interrupts increases the overheads of power consumption in battery-powered embedded systems.

In past work [5], we introduced the RVEC virtual system clock, which we briefly review next. Assume a running application in a multicore processor with DVFS. Over the execution time of the application, the passage of time will evolve accordingly to its execution time intervals, each of which depends on the operating frequency of the core unit used in the specific time interval. Therefore, the implementation method of RVEC focuses on tracking each of the execution time intervals and the accumulated execution time intervals for the running applications. To this end, RVEC implements a virtual system clock by using a simple data structure stored in main memory composed of a core unit cycle count and the elapsed time, as follows. Note that our choice of using the cycle count of the core units has some significant advantages: the cycle count operates at the same high frequency of the core unit and generates no interrupts; moreover, the core unit cycle count is based on an oscillator that is highly stable and accurate.

The RVEC implementation uses a data structure composed of two fields: base_count and the age_time. The base_count field stores the last value read

from the core unit register, which indicates the current number of cycles. The age_time_ns field stores the elapsed time between the time of the RVEC creation up to the time indicated by the base_count field, which is the last value read from the core unit register. This way, the data structure must be updated whenever the core unit frequency changes. Figure 1 shows the update procedure used by the implementation of RVEC.

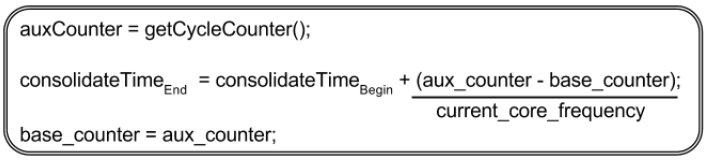

Figure 1: RVEC update procedure when DVFS is used

Figure 2 illustrates the steps of the RVEC update procedure used for an EE computer system that support DVFS. In the figure, we note that just before the core unit operating frequency changes, the value of the core unit cycle count is stored in the base_count field, after which the age_time_ns field is updated. Specifically, the figure shows the change of the operating frequency of a processor core from 800 MHz to 400 MHz when the cycle count reaches the value 4. By using the procedure shown in Figure 1, the RVEC base_count and age_time_ns fields will be updated to 4 ns and 5 ns, respectively.

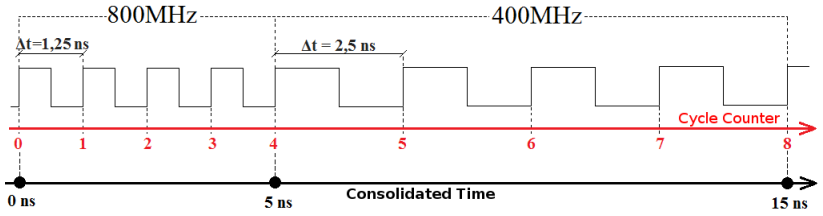

Figure 2: The time calculation performed when the operating frequency of a processor core changes

Figure 3 presents the RVEC read procedure, which is applied to the previous example of Figure 2. Specifically, at the instant when the core unit cycle count reaches the value 8, the reading of RVEC will return the value 15 ns. The combination of RVEC data structure with the maintenance and reading procedures creates a virtual timekeeping mechanism that supports a time count with the SIP property.

3.1 Implementation of the RVEC in the RasPi platform

The RasPi platform has been used in various sensing projects [12, 13], as the computing platform for building Wireless Sensor Networks (WSN) [9]. In

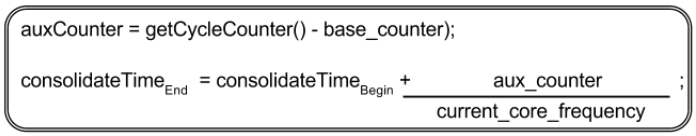

Figure 3: The RVEC reading procedure

fact, the energy efficiency of a computing platform is an important issue for WSN applications, increasing the attractiveness of using of an embedded device that can use DVFS to reduce its energy consumption during periods of low utilization, although the use of DVFS can affect some of the circuits used by the system clock. In this regard, our Linux implementation of RVEC in the RasPi platform introduces an efficient solution that enables the use of both the ARM processor cycle count- namely, the Cycle Count Register (CCNT) [22]- and DVFS for energy-efficient WSN applications.

To work properly, we integrated RVEC into the Linux kernel boot process. For instance, in the x86 platform, the core unit running the boot process is the same one that starts the operating system kernel and performs the initial configuration of RVEC [5]. In contrast, in the RasPi platform, the GPU starts and executes most of the boot process, rather than the core unit.

The initialization process of the RasPi platform consists of four stages, as follows. First, the VideoCore GPU starts the execution of the first-stage bootloader that is stored in the ROM of the BCM2835 SoC. After, the first-stage bootloader reads the SD card and transfers the second-stage bootloader (bootcode.bin) from the SD card to its L2 cache. Next, the execution of the second-stage bootloader enables the SoC DRAM and loads start.elf- i.e., the third-stage bootloader- from the SD card into the DRAM, where start.elf is also the GPU firmware. start.elf then splits the DRAM space between the GPU and the ARM core through the the use of the VideoCore GPU MMU (Memory Management Unit). After that, start.elf loads kernel.img, which is the binary file that contains the Linux kernel, into the Raspi platform and sends a reset signal to the ARM processor core. Finally, the ARM1176JZF-S processor core executes the kernel.img thereby loading the operating system and starting the RVEC initialization procedure.

The ARM1176JZF-S processor core includes the CP15 system control coprocessor that supplies the Cycle Count Register (CCNT), which is a CPU cycle count equivalent to the TSC in the x86 architecture. However, unlike the TSC, access to the CCNT register by RVEC is enabled during the initialization procedure of the kernel. Figure 4 shows how the processor core reaches the CCNT. The access to the CCNT count is performed by reading the c15 register of the CP15 coprocessor. The CCNT is always available within the operating system kernel whereas the MCR instruction is used to read the CCNT, in which the instruction fields must have the following data.

- Opcode_1 defined as 0;

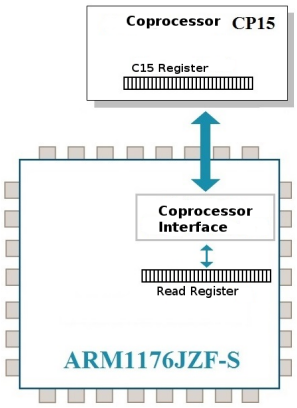

Figure 4: ARM1176JZF-S processor and its Coprocessor CP15

- CRn defined as c15;
- CRm defined as c12;
- Opcode_2 defined as 1.

Afterwards, we addressed the problem of the count limitation of the CCNT's 32-bit count. Clearly, given a core unit that operates at a 1 GHz frequency, CCNT overflow occurs every 4.295 s. Thus, the use of CCNT for measuring time intervals greater than 4.3 s will violate the SIP property of the time counting. Therefore, we expanded the 32-bit CCNT cycle count to a 63-bit CCNT cycle count by using the macro $cnt32\_to\_63$ available in the Linux kernel [2]; this expansion is performed during the Linux task scheduling.

The data structure **struct tb** shown in Figure 5 corresponds to an implementation of the conceptual data structure of RVEC by using the base_count and age_time_ns fields described above. For the correct operation of RVEC, we must include one instance of the **struct tb** data structure in both the data structures that support the processing queue of the core unit (**struct rq**) and a task in the operating system (**struct task_struct**).

The data structure **struct tb** shown in Figure 5 is the implementation of the conceptual data structure of RVEC described earlier in this section, with the fields base_counter and age_time_ns. For its correct operation of RVEC, we must include one instance of the **struct tb** data structure in the data structure representing the core processing queue (**struct rq**) and, also, in the data structure that represents a task in the operating system (**struct task_struct)**.

The final implementation of the RVEC for Linux running in the ARM platform is shown in Figure 6. To simplify the display of RVEC implementation, the figure shows a hypothetical multiprocessor with two core units. In the figure, we can see both subsystems of Linux that have been adapted to guarantee

[2] include/linux/cnt32_to_63.h

```
struct tb{
   u64 base_counter;
   u64 age_time_ns;}
```

Figure 5: RVEC data structure

the correct operation of RVEC. The double adaptation of the Linux kernel is necessary to ensure that the changes to the processor frequency will not cause the effect in cascade of RVEC updates on all running applications tasks in the processor core. Of note, this implementation of the RVEC technique allows different running applications tasks in the RasPi platform, each of which can check its associated RVECs through the system call clock_gettime() [23] by requiring only that each of the applications tasks use the clock identifier CLOCK_RVEC as the input parameter.

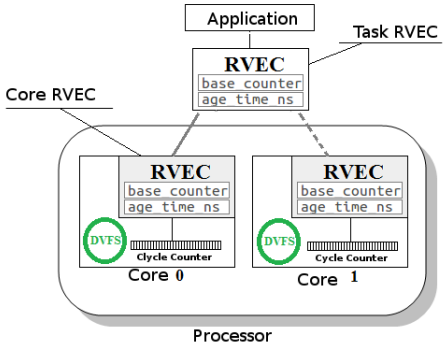

Figure 6: Overview of the RVECs core unit and RVECs application task

To enable the proper operation of RVEC, the necessary adaptations to the Linux DVFS drive are shown in Figure 7. The changes were performed in the bcm2835-cpufreq device driver [3] developed for the ARM11 family of processors, which includes the ARM1176JZF-S processor of the RasPi platform. The original function bcm2835_cpufreq_set_clock()used to change the operating frequency of the processor core was extended to include two new functions- namely, rvec_cpu_freq_change_pre() and rvec_cpu_freq_change_pos(). The rvec_cpu_freq_change_pre() function for RVEC works on the previous change (change_pre) configuration of the operating frequency, which occurs in the management subsystem of the core unit running the update RVEC procedure. After changing the operating frequency of the processor core, the function rve_cpu_freq_change_pos() adds to the current sum of elapsed times the time spent using the current frequency, since the beginning of the execution of the rvec_cpu_freq_change_pre() function, after which the current value of core unit CCNT register is stored in base_count.

[3]./drivers/cpufreq/bcm2835-cpufreq.c

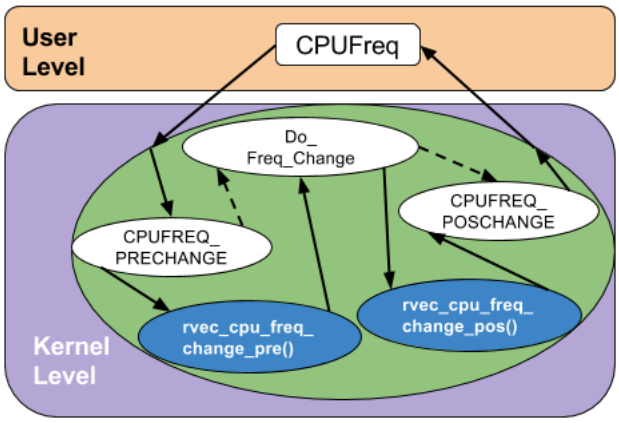

Figure 7: Modifications of the bcm2835-cpufreq driver used by RVEC

4 Experimental Evaluation

In this section, we present an experimental evaluation of the RVEC virtual system clock for the RasPi platform. First, we assess the impact of using NTP on the time synchronization of a WSN node based on the Wi-Fi RasPi platform, followed by an evaluation of the efficiency of the RVEC-based timekeeping technique. Table 1 summarizes the characteristics of our experimental platform. All results have a confidence interval of 99.9% for the sample mean.

Table 1: The RasPi Platform - Components

Component	Description
Raspberry Pi	Model B+
Processor	ARM1176JZF-S
Default ClockSource	STC (1 MHz)
Linux	3.2.27+

4.1 On using the NTP synchronization of the WiFi RasPi platform

In computer systems, the system clock usually maintains its accuracy through periodic synchronization with an absolute time reference, such as the NTP. However, in heavily loaded computer systems, NTP-based synchronization cannot run frequently enough to compensate for the time drifts of the system clocks, which in turn reduces the accuracy of the time counting. In practice, even for a lightly loaded computer system, time adjustments based on the NTP can

advance the local system clock far enough that the deadline of a running application becomes compromised, as we will show next.

We collected the experimental results from the corrections performed by the NTP daemon on the system clock for three distinct computing platforms: a wired desktop computer, a wired RasPi platform, and a wireless RasPi platform using the WPi dongle. For each of the three computing platforms, we evaluated three distinct time intervals between the NTP queries. In the first experiment, we executed the NTP queries continuously with no time interval between the queries. In the next two experiments, we inserted intervals of 10 s and 600 s between the NTP queries. We performed each of the experiments 200 times, each of which sent 20 queries to the NTP server. The computing platform was lightly loaded by using minimal system services, one NTP client and a monitoring application. We also discarded the execution of the experiment in which the time interval between two successive NTP queries did not obey the limit set to the NTP query rate because of some network failure. Note that a loaded computer system can further degrade the accuracy of a local system clock because the hardware of a cycle counter issues interruptions that can be missed.

Table 2: Δ Statistics of the NTP updates to a Local System Clock

Platform	Time Interval (s)	μ (ms)	σ (ms)
Intel i7-870 Ethernet	0 s	0.503	0.112
Intel i7-870 Ethernet	10 s	0.660	0.235
Intel i7-870 Ethernet	600 s	10.169	0.651
Raspberry Pi Ethernet	0 s	0.640	0.415
Raspberry Pi Ethernet	10 s	0.708	0.625
Raspberry Pi Ethernet	600 s	20.348	4.232
Raspberry Pi WiFi	0 s	1.486	1.019
Raspberry Pi WiFi	10 s	2.042	3.995
Raspberry Pi WiFi	600 s	29.174	25.137

Table 2 shows the results of the nine experiments. The first column identifies the computing platform, the second column shows the time interval between the NTP queries, and the third column presents the average time adjustment that the NTP required of the local clock. The last column shows the standard deviation of the time measurements.

The first experiment creates an ideal case for the NTP service, which runs continuously in a lightly loaded computer system. In this ideal case, the table reveals that a running application will be unable to respond to the alerts that occur within time intervals shorter than 1 ms because the NTP service requires 1 ms to perform only two consecutive updates to the system clock of a wired desktop computer.

The results of Table 2 contrast with the execution of the NTP service whether performed by a desktop computer or the RasPi platform. Although both computing platforms can reach the NTP server over the Ethernet connection, the RasPi platform compared with the desktop computer, performed updates of greater time length to its local clock within the same time interval between

NTP queries.

Furthermore, the use of a Wi-Fi connection between the RasPi platform and the NTP server requires a higher synchronization rate of NTP updates to the local system clock. The Wi-Fi communication of the RasPi platform increased the average update time from 0.708 ms to 2.042 ms for the time interval of 10 s between queries with a standard deviation 6.4 times greater. Moreover, in the case of a time interval of 600 s between queries, the average update time of 29.174 ms is 43.38% bigger, and the standard deviation is 5.94 times larger.

Considering the above results, which confirm the observations we found in previous work [24], we expect that a conventional approach to the design of a WSN composed of nodes based on the RasPi platform that uses NTP for the synchronization of the system clocks will significantly limit the accuracy of the time measurements used by the distributed applications. In fact, Corbett et al. [25] presents Spanner, a globally-distributed database based on an implementation of TrueTime API that keeps clock uncertainty small at less than 10_ms by using GPS and atomic clocks. The authors showed that Spanner supports transaction semantics with reduced overheads by enforcing tighter bounds on the clock uncertainty.

4.2 Evaluation of the SIP property of RVEC

An evaluation of the RVEC SIP property must first experimentally confirm the adherence of CCNT to the SIP property. To this end, we built a microbenchmark composed of a block of 12 operations within a loop of 400 iterations, where each of the operations performed an arithmetic sum. The experiment consisted of running the microbenchmark up to 100 times and using the MCR instruction to read the value of the cycle count stored in the CCNT register that was used to infer the microbenchmark runtime.

4.2.1 Assessing the SIP property of the CCNT cycle count

As we previously discussed, CCNT is a 32-bit cycle count register that operates at the same frequency as the processor core and provides time measurements with greater accuracy than those of conventional system clocks. However, CCNT can be used as a time base for the RVEC solution only if CCNT is adherent to the SIP property, as described in Section 1.

Figure 8 shows one of the samples of the CCNT's experimental evaluation described in Section 4.2 and is used to validate the CCNT SIP property. As seen in the figure, the execution of the microbenchmark did not violate the SIP property. The experimental evaluation of the CCNT revealed an average run time of 361.91 ms with a standard deviation of 18.33 ms and the processor frequency set at 800 MHz. The coefficient of variation for the experiments was 0.051, which provides a preliminary indication of the stability of the CCNT.

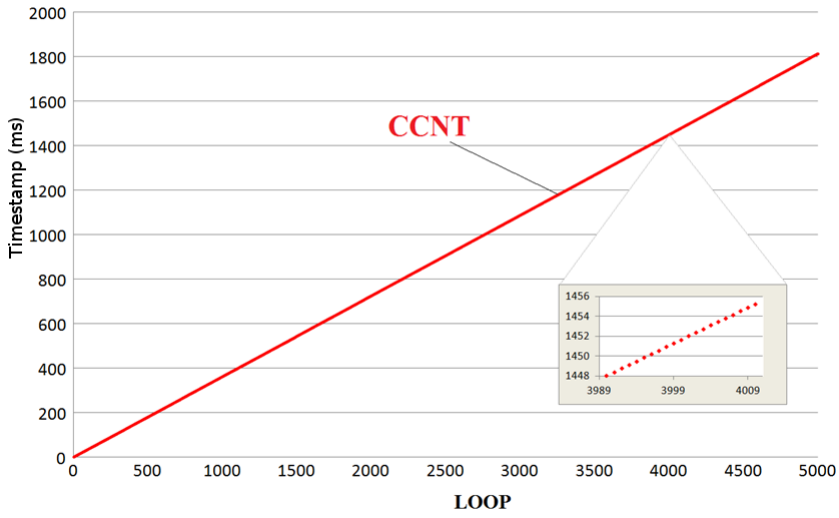

Figure 8: Assessing the SIP property of CCNT

Therefore, based on these results, we concluded that the CCNT is adherent to the SIP property over the time length of all experiments that were performed.

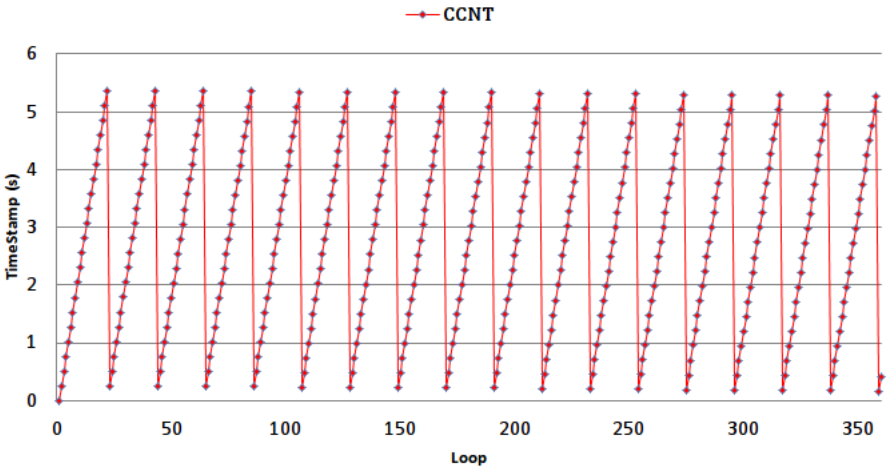

Figure 9: Assessing the overflow of CCNT count

However, we note that the CCNT is SIP adherent under the limited condition provided by the short duration of each of the experiments. Specifically, the CCNT is a 32-bit register, so an overflow will occur when it is used for measuring

intervals greater than 5 s, as shown in Figure 9. In this figure, the average duration of each of the iterations was 255.372 ms, thus causing an overflow of the CCNT count after the execution of every 20 iterations. In other words, these results confirm the limitation of directly using the 32-bit CCNT for the time measurements of running applications in the RasPi platform. As we reported in Section 3.1, the Linux OS provides a C macro to expand the CCNT count from 32-bit to 63-bit, which we used in building the RVEC virtual system clock.

4.2.2 Assessing the SIP property of the RVEC

We collected the following results during the experimental evaluation of the RVEC virtual system clock. Figure 10 shows an execution sample of the experiments we performed to evaluate the RVEC SIP property. For this evaluation, we repeated the same experiments used for the evaluation of the CCNT. Specifically, we ran the same microbenchmark 100 times. The evaluation of the RVEC SIP property produced an average run time of 372.88 μs with a standard deviation 20.02 μs for the processor frequency fixed at 800 MHz. The coefficient of variation for the experiments was 0.054, which gives us a preliminary indication of the stability of the RVEC implementation; therefore, RVEC is adherent to the SIP property. Moreover, it is important to note that the increase in the number of instructions required to gain access to RVEC is responsible for the increased coefficient of variation compared with CCNT.

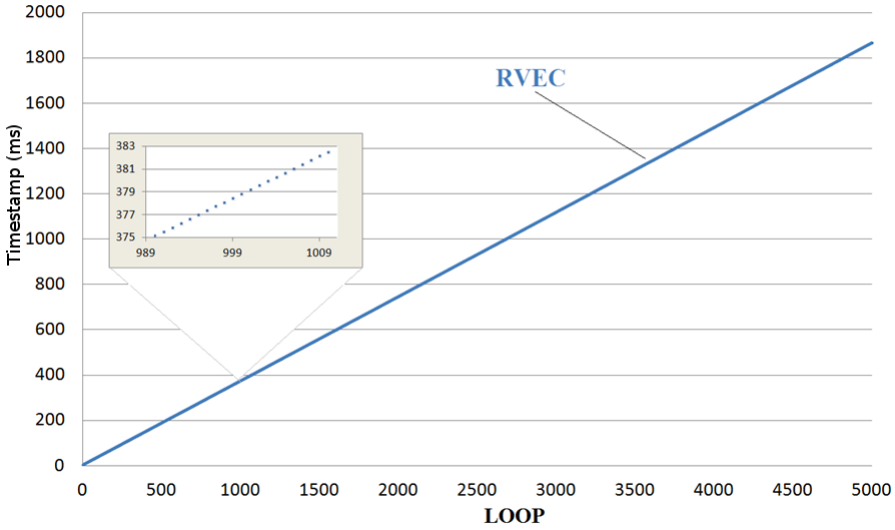

Figure 10: Assessing the SIP property of RVEC

To evaluate the use of RVEC for measuring long periods of time, we submit-

ted RVEC to a loop of 100 iterations, each of which has an average runtime of 1.108 s. The results are shown in Figure 11. Clearly, the use of RVEC solved the existing overflow problem that occurs in the 32-bit CCNT hardware, as this figure shows.

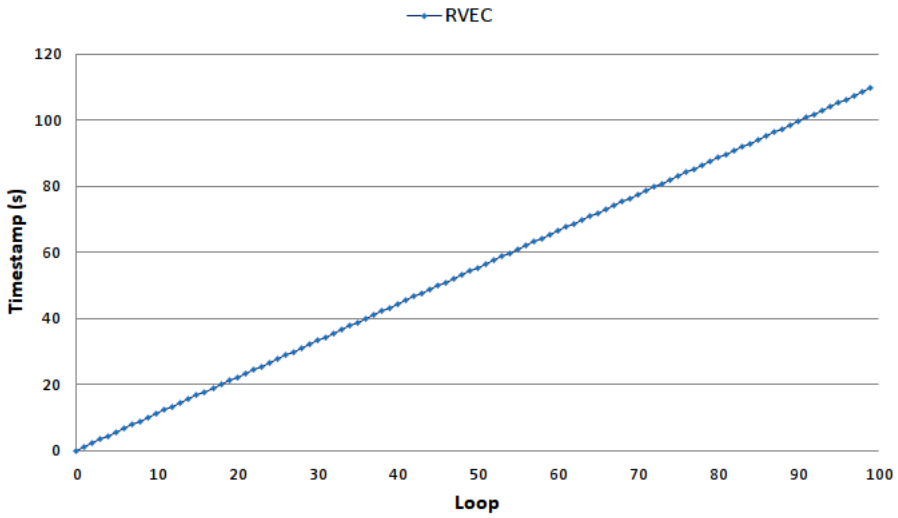

Figure 11: Assessing the RVEC on using CCNT for long periods of time

Figure 12 presents the evaluation of RVEC SIP property when using DVFS for changing the processor frequency. We used a microbenchmark composed of a testing task and a frequency change task. The microbenchmark starts with the execution of the frequency change task that releases the execution of the testing task by calling sem_post(&OkExec) and then waits for a call to sem_wait(&frqS), which will change the core operating frequency. Next, the microbenchmark (post fork) testing tasks continues to wait in a call to sem_wait(&OkExec) to ensure that the creation of the frequency change task occurs before the execution of the testing task. The microbenchmark performed an evaluation of a loop with $5,000$ arithmetic instructions, which was divided into two blocks of $2,500$ instructions; at the end of the first block, the testing task calls sem_post(&frqS) and releases the frequency change task. As a result, the operating frequency decreases from 800 MHz to 400 MHz when executing the second loop concurrently. In the figure, the blue colour represents the RVEC time measurements, whereas the red colour represents those of the CCNT. The figure shows, as we could expect, that the RVEC curve indicates an increase in the average execution time, whereas the associated curve of CCNT remains unchanged.

Moreover, in the figure, the red curve (CCNT) has three different stages of progression: the first stage is the iteration interval (0..2500), the second stage is in the interval range [2500..2650), and the third stage is in the interval range [2650..5000). The rates of the first and third stages of progression are equal, whereas the rate of progression of the second stage is influenced by the interfer-

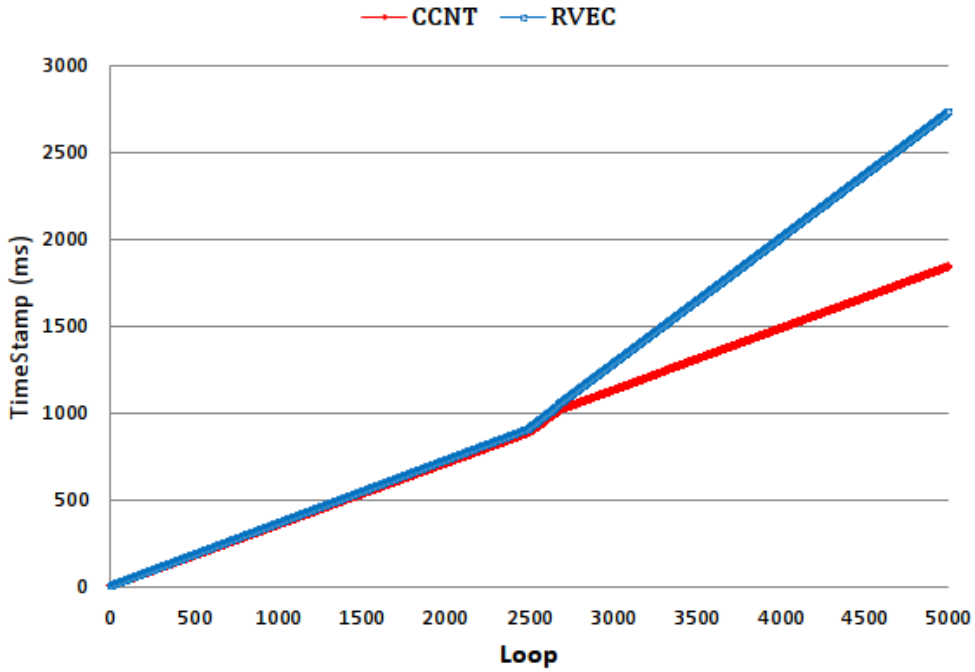

Figure 12: Assessing the SIP property of both RVEC and CCNT by changing both the operating frequency and the concurrent task for execution

ence (concurrent execution) of the frequency change task with the testing task running the second part of the microbenchmark. This hypothesis is based on the method by which we implemented the support to RVEC in the DVFS device driver, which is shown in Figure 7 of Section 3.1. Furthermore, in Figure 7, it is still possible to observe that RVEC (blue curve) shows only two distinct stages of progression with two distinctive rates, despite the reduced time measurements during the [2500..2650) interval.

To confirm the hypothesis we used to explain the behaviour of the previous experiment, we developed a new experiment in which the change in the operating frequency occurs within the same task that will subsequently perform the second loop, instead of using another task. Figure 13 shows the RVEC behaviour when using the same $5,000$ instructions from the previous experiment; it is possible to note the presence of only two progression rates of execution by using CCNT (red colour) and RVEC (blue colour). Therefore, based on the results shown in Figure 13 the presence of three different stages of progression for CCNT in Figure 12 effectively demonstrates the result of the concurrent execution of both the testing task and frequency change task of the microbenchmark.

Figure 12 and Figure 13 show the RVEC and CCNT curves, respectively. We note that the curves exhibit a small divergence that increases progressively

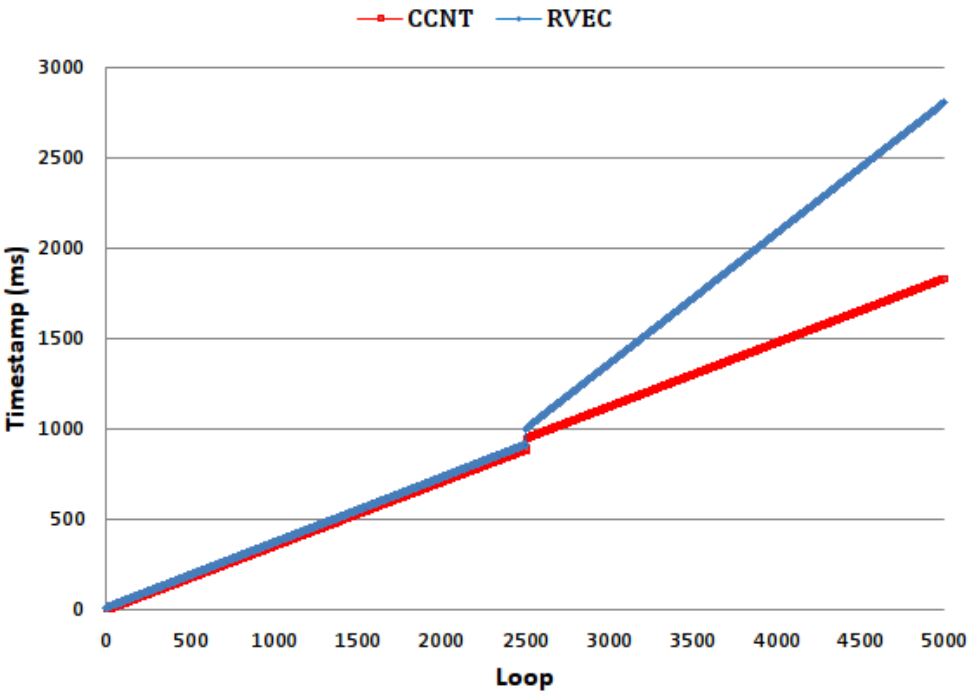

Figure 13: The SIP property of RVEC and CCNT when only the operating frequency changes

along the experiments, which can be explained by the different access costs of RVEC and CCNT.

4.3 The Access Cost of the RVEC virtual system clock

We evaluated the access cost of RVEC by using the CCNT cycle count, which provides the best available accuracy for a cycle count within the RasPi platform. In this way, we could also assess CCNT's own access cost and the access costs of the MONOTONIC and REALTIME system clocks of Linux running in the RasPi platform. We used a microbenchmark that consists of a block of $2,400$ arithmetic instructions within a loop, in which access to the system clock being evaluated will occur after the execution of the first $1,200$ arithmetic instructions in the loop. This evaluation loop was executed $1,000$ times, in which one of the four system clocks- namely, CCNT, RVEC, MONOTONIC, and REALTIME – in turn, was evaluated for each of the loop iterations by using the system call clock_gettime() for the last three system clocks.

Figure 14 presents the five experimental configurations used by the microbenchmark described above. In Scenario 1, no query is performed to any

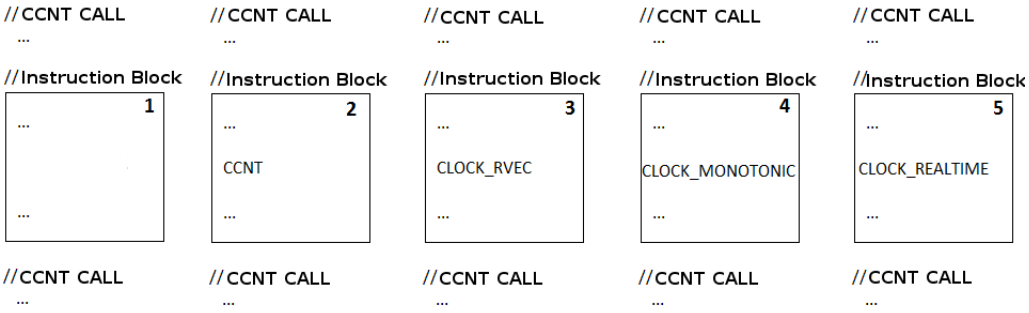

Figure 14: Evaluation of the Configuration Overhead

of the system clocks that served as the reference of the least access cost for comparison purposes. Configurations 2, 3, 4, and 5 evaluate the cost of access to CCNT, RVEC, MONOTONIC, and REALTIME, respectively, where each of the configurations was evaluated 100 times.

Table 3: Execution Time x System Clock (800 MHz)

System Clock	Mean (μs)	Std. Dev. (μs)	Overhead (μs)	Overhead (%)
CCNT	171.687	6.729	1.333	0.78
RVEC	174.502	7.419	4.148	2.43
MONOTONIC	173.107	6.918	2.753	1.62
REALTIME	173.013	6.891	2.659	1.56

Table 3 shows the results of our experimental evaluation of the RasPi platform running at 800 MHz. For the reference configuration, the average access time was 170.354 μs, with standard deviation of 6.507 μs. CCNT, RVEC, MONOTONIC and REALTIME had an average overhead of 1.333 μs (0.78%), 4.148 μs (2.43%), 2.753 μs (1.62%), and 2.659 μs (1.56%), respectively. Compared with MONOTONIC, the small difference that is favourable to REALTIME is explained by the additional validations performed by the MONOTONIC system clock.

We repeated the same experiments by using the RasPi platform but changing its operating frequency to 400 MHz. In this case, as Table 4 shows, the average access time was 341.679 μs for the reference configuration, whereas CCNT, RVEC, MONOTONIC, and REALTIME obtained average execution overhead of 2.702 μs (0.79%), 8.002 μs (2.34%), 5.188 μs (1.52%) and 5.125 μs (1.50%), respectively.

Overall, the foregoing results demonstrate that the overheads of the RVECs access costs are compatible with those of current system clocks (MONOTONIC and REALTIME) for the RasPi platform. In addition, these current system

Table 4: Execution Time x System Clock (400 MHz)

System Clock	Mean (μs)	Std. Dev. (μs)	Overhead (μs)	Overhead (%)
CCNT	344.381	19.831	2.702	0.79
RVEC	349.680	20.448	8.002	2.34
MONOTONIC	346.867	19.187	5.188	1.52
REALTIME	346.804	19.720	5.125	1.50

clocks of the RasPi platform neither guarantee the SIP property nor support the use of DVFS whereas RVEC does.

4.4 Evaluation of the Time Count Precision of RVEC

In this section, we perform an evaluation of the precision of the Linux system clocks for the RasPi platform by using the microbenchmark with configuration 1, as described in Section 4.3. Now, however, the block of $2,400$ arithmetic instructions is measured by using the CCNT, the Linux system clocks (MONOTONIC and REALTIME), and the RVEC implementation for the RasPi platform. Our experimental evaluation consisted of repeating the experiment of the previous section 100 times.

Table 5: Time Precision x System Clock (800 MHz)

System Clock	Mean (μs)	Std. Dev. (μs)
CCNT	170.439	6.554
RVEC	172.820	7.248
MONOTONIC	172.047	7.086
REALTIME	171.459	7.140

Table 5 shows the results of the time precision of the different system clocks. The results were collected for the RasPi platform operating at a frequency of 800 MHz. For the CCNT, the average execution time of the block was 170.439 μs with a standard deviation of 6.554 μs, which represents a coefficient of variation of 0.038. In the case of RVEC, the average execution time was 172.820 μs with standard deviation of 7.248 μs with coefficient of variation of 0.042, whereas the MONOTONIC and REALTIME system clocks measured 172.047 μs and 171.459 μs with coefficients of variation of 0.041 and 0.042, respectively.

4.5 Discussion of results

The results of the experimental evaluation presented above confirmed the stability of the CCNT cycle counter of the ARM1176JZF-S processor. The results also provide preliminary evidence, although significant, of the stability of the RVEC implementation in the RasPi platform. The results presented in Section 4.1 demonstrate that the NTP synchronization, which is predominantly used by computer systems in wired networks, reduces the efficiency of system clocks by reducing their accuracy and precision; ultimately, the current system clocks of the RasPi platform cannot guarantee the property of strictly increasing and precise (SIP) time counting.

In the case of wireless networks, NTP synchronization further reduces the efficiency of system clocks, especially for wireless sensor networks that use the RasPi platform as the WSN node. In fact, the experiments presented in Section 4.4 show that both CCNT (0.038) and RVEC (0.042) have coefficients of variation less than 0.05. Furthermore, in Section 4.2, the results also show that the RVEC virtual system clock is the only system clock that adheres to the SIP property independently of the length of the time interval that is measured.

5 Conclusion

In this work, we used the ARM1176JZF-S processor cycle counter (CCNT) to implement and evaluate the efficiency of the RVEC virtual system clock for the RasPi platform. In contrast to the current system clocks (MONOTONIC and REALTIME) of the RasPi platform, the RVEC virtual system clock guarantees the property of strictly increasing and precise (SIP) time counting independently of the length of time interval being measured.

Our experimental results show that the implementation of RVEC presents an access cost slightly higher than that of the current system clocks of the RasPi platform. However, the results also demonstrate that RVEC preserves the SIP property of time counting even if it is exposed to changes in the operating frequency of the processor core. Therefore, the RVEC can afford for the RasPi platform to increase its energy efficiency by making use of DVFS, in contrast to the current system clocks of the RasPi platform that restrict its use.

Furthermore, our results also indicate that the current system clocks suffer high overhead by using the NTP synchronization in a WSN node that uses the RasPi platform, in contrast with the RVEC virtual system clock which maintains its adherence to the SIP property even if RVEC uses the 32-bit CCNT register. Future research work will address the limit of stability of RVEC over larger time intervals and its effects on the energy efficiency of EE computer platforms considering that RVEC is based on a hardware counter that generates no interrupts.

Overall, our results confirm that RVEC provides an alternative system clock with nanosecond resolution for the RasPi platform while maintaining its access cost equivalent to that of the other system clocks of the RasPi platform.

Most importantly, the idea behind RVEC of using a virtual system clock as an efficient system clock for energy-efficient computer systems fosters the development of new techniques for wireless sensor networks, such as an efficient global synchronization time protocol that uses RVEC as the basic system clock.

References

[1] D. Suleiman, M. Ibrahim, and I. Hamarash, "Dynamic voltage frequency scaling (dvfs) for microprocessors power and energy reduction," in *4th International Conference on Electrical and Electronics Engineering*, 2005.

[2] D. Mills, "Network time protocol (version 3) specification, implementation," United States, 1992.

[3] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The flooding time synchronization protocol," in *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, ser. SenSys '04.

[4] R. Fan, I. Chakraborty, and N. Lynch, "Clock synchronization for wireless networks," in *In Proc. 8th International Conference on Principles of Distributed Systems (OPODIS)*, 2004, pp. 400–414.

[5] D. Dutra, L. Whately, and C. L. Amorim, "Attaining strictly increasing and precise time count in energy-efficient computer systems," in *Computer Architecture and High Performance Computing (SBAC-PAD), 2013 25th International Symposium on*, Oct 2013, pp. 65–72.

[6] B. Corporation, *BCM2835 ARM Peripherals*, 2012. [Online]. Available: https://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf

[7] J. Khan, S. Bilavarn, and C. Belleudy, "Energy analysis of a dvfs based power strategy on arm platforms," in *Faible Tension Faible Consommation (FTFC), 2012 IEEE*, June 2012, pp. 1–4.

[8] T. Simunic, L. Benini, and G. De Micheli, "Energy-efficient design of battery-powered embedded systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 9, no. 1, pp. 15–28, Feb 2001.

[9] V. Vujovic and M. Maksimovic, "Raspberry pi as a wireless sensor node: Performances and constraints," in *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014 37th International Convention on*, May 2014, pp. 1013–1018.

[10] M. Haghighi and D. Cliff, "Sensomax: An agent-based middleware for decentralized dynamic data-gathering in wireless sensor networks," in *Collaboration Technologies and Systems (CTS), 2013 International Conference on*, May 2013, pp. 107–114.

[11] M. Kochlan, M. Hodon, L. Cechovic, J. Kapitulik, and M. Jurecka, "Wsn for traffic monitoring using raspberry pi board," in *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*, Sept 2014, pp. 1023–1026.

[12] T. Nagy and Z. Gingl, "Low-cost photoplethysmograph solutions using the raspberry pi," in *Computational Intelligence and Informatics (CINTI), 2013 IEEE 14th International Symposium on*, Nov 2013, pp. 163–167.

[13] R. Neves and A. Matos, "Raspberry pi based stereo vision for small size asvs," in *Oceans - San Diego, 2013*, Sept 2013, pp. 1–6.

[14] T. Broomhead, J. Ridoux, and D. Veitch, "Counter availability and characteristics for feed-forward based synchronization," in *Precision Clock Synchronization for Measurement, Control and Communication, 2009. ISPCS 2009. International Symposium on*, Oct 2009, pp. 1–6.

[15] E. Corrêa, D. Dutra, and C. L. Amorim, "Relógio virtual estritamente crescente para o computador raspberry pi," in *Simpósio de Sistemas Computaciaonais (WSCAD-SCC)*, Oct. 2015, in Portuguese.

[16] D. Veitch, J. Ridoux, and S. Korada, "Robust synchronization of absolute and difference clocks over networks," *Networking, IEEE/ACM Transactions on*, vol. 17, no. 2, pp. 417–430, April 2009.

[17] G.-S. Tian, Y.-C. Tian, and C. Fidge, "High-precision relative clock synchronization using time stamp counters," in *Engineering of Complex Computer Systems, 2008. ICECCS 2008. 13th IEEE International Conference on*, March 2008, pp. 69–78.

[18] A. F. de Souza, de Souza S. F., C. L. de Amorim, P. Lima, and P. Rounce, "Hardware supported synchronization primitives for clusters," in *PDPTA'08*, 2008, pp. 520–526.

[19] D. L. Mills, "Network Time Protocol (Version 3) Specification, Implementation and Analysis," 1992, Network Working group report RFC 1305.

[20] S. Muir, "The seven deadly sins of distributed systems," in *First Workshop on Real, Large Distributed Systems, WORLDS'04*, Dec. 2004.

[21] C.-Y. Hong, C.-C. Lin, and M. Caesar, "Clockscalpel: Understanding root causes of internet clock synchronization inaccuracy," in *Proceedings of the 12th International Conference on Passive and Active Measurement*, ser. PAM'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 204–213. [Online]. Available: http://dl.acm.org/citation.cfm?id=1987510.1987531

[22] A. Limited, *ARM1176JZ-S Technical Reference Manual*, 2009. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0333h/DDI0333H_arm1176jzs_r0p7_trm.pdf

[23] N. Clifford and A. Brouwer, *Linux Programmer's Manual: clock_gettime system call*. [Online]. Available: http://man7.org/linux/man-pages/man2/clock_gettime.2.html

[24] G. Neville-Neil, "Time is an illusion." *Queue*, vol. 13, no. 9, pp. 30:57–30:72, Nov. 2015. [Online]. Available: http://doi.acm.org/10.1145/2857274.2878574

[25] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford, "Spanner: Google’s globally distributed database," *ACM Trans. Comput. Syst.*, vol. 31, no. 3, pp. 8:1–8:22, Aug. 2013. [Online]. Available: http://doi.acm.org/10.1145/2491245