COMPUTERS AND INFORMATICS
2757-8259 2022, Volume 2 Issue 1
Research Article

Real-Time calculation of common power quality indices on Linux RT-Patch

Mehmet Gök
Kahramanmaraş İstiklal University, Department of Digital Game Design, Kahramanmaraş, Turkey, gokmehmet@outlook.com
İbrahim Sefa
Gazi University, Department of Electrics and Electronics Technology Engineering, Ankara, Turkey

Submitted: *22.04.2022*
Accepted: *07.06.2022*
Published: *30.06.2022*

Abstract:

Use of Linux with real-time capabilities is a common study in control systems and data processing due to programmer-friendly libraries, huge community support and free software. Real-time Linux with low-cost single board computers (SBCs) present a powerful evaluation environment for developers and researchers today. In this paper, a power signal processing application using Linux RT-Patch is realized. To evaluate the application, both real-time and non-real-time versions tested on single board computers Beaglebone Black and Raspberry Pi 3. Under load conditions, the non-real-time version of the application failed on Beaglebone Black by exceeding the real-time dead-line time of 20 ms consecutively. Raspberry Pi 3 succeeded even with non-real-time operation under load conditions. For both computers, the real-time version of the applications succeeded with a mean processing time under 10 ms. According to processing times, real-time operation brings a great performance enhancement particularly on Beaglebone Black with single core ARM processor. The measurement results show that the proposed system can be used for smart power metering and power signal acquisition purposes.

Keywords: *Harmonics detection, Real-time operating system, RT-Patch, Single Board Compute*

Cite this paper as: Gök, M. & Sefa, İ. Real-Time calculation of common power quality indices on Linux RT-Patch, *Computers and Informatics*, 2022, *2*(1), 1-13.

1. INTRODUCTION

In an AC electricity network, harmonics are defined as periodical voltage and current components overlapping the mains signal. The fundamental harmonic sources are non-linear time-varying loads, digitally controlled equipment such as variable speed motor drives, solid state switching components and lighting ballasts and etc. Overheating of wires and transformers, malfunction of relays and breakers, over voltage and flicker effect are the major effects of harmonics. Increasing the harmonic level may also result in economic losses. Hence harmonics have to be monitored and determined with adequate equipment to be taken the preventive measures [1-4].

There have been many different hardware and software applied to determine the harmonics in real-time in the literature. DSP (Digital Signal Processor), FPGA (Field Programmable Gate Arrays) and MCU (Microcontoller Unit) and combined solutions integrating those units together, have been used so far [5-9]. In most of those studies bare metal software or RTOS (Real Time Operating System) were used to

determine the harmonics. Recently, Linux based approaches attract the attention of researchers because of the integration with a general-purpose OS (Operating System) advantages [10].

Nowadays, Linux based operating systems are common among servers, desktop computers and embedded devices in contrast to mobile devices running Android OS [11]. Linux based operating systems provide system developers with wide range of libraries and development tools. Being royalty free and availability for embedded systems move Linux derived operating systems one step ahead. In addition to this, having real-time capabilities Linux based operating systems are preferred more in industrial applications [12-14]. Also, real-time applications with Linux may be promising for many research areas such as robotics and signal processing.

In this study, hardware equipped with RT-Patched Linux is proposed to monitor and determine the harmonics with logging capability. This hardware consists of two parts: USB connected data acquisition device and a single board computer: Raspberry Pi 3 or Beaglebone Black. These devices were benchmarked against harmonics monitoring and determination. Experiment results show that both devices are feasible for the task with the use of RT-Patched Linux. In this way, a low-cost harmonics determination and recording task can be executed without the use of complex and costly commercial harmonics analyzers.

This study investigates the advantages of using RT-Patch for a real-time power signal processing application. The main contribution of this study is to show that real time power signal analysis and monitoring are possible with-use of low power and cost-effective single board computers running RT-Patched Linux.

In the next sections, the related studies in the literature are first given and then the proposed system is also presented. Evaluation results of the real-time harmonics detection system are then discussed. Lastly, we give conclusions of the study based on obtained results and highlights directions for future work.

2. RELATED WORK

The literature survey for this manuscript is classified into two sections: the studies on power signal processing and research on real-time feature enablement within Linux. This work aims at contributing to the research in real-time harmonics measurement by merging knowledge given in those sections.

2.1. Model

The paper [10] present a preliminary application for power signal processing with use of RT-Patched Linux. The authors illustrated a data acquisition test at two different sampling rates (1 ksps and 5 ksps). The application is evaluated with non-real time and real-time with Linux running on a single board computer with Allwinner Tech A20 System On Chip (SoC). Analog-digital converter (ADC) integrated circuit is interfaced to SoC via serial peripheral interface (SPI) bus. RT-patch powered Linux kernel is successful at 1 ksps sampling rate. However, the jitter is too high for the target application at 5 ksps sampling rate. A Field Programmable Gate Array (FPGA) based design is recommended to control ADC and buffer the acquired data. Because, gathering data from an ADC requires a hard real-time tasking.

A multi-channel electrical power signal data acquisition and power metering system is illustrated in [5]. The authors developed the system by using FPGA and AD7606 multi-channel simultaneous ADC. Digitized signal data is read and power parameters voltage, current, active, apparent and reactive power and CosΦ are calculated. The parameters are displayed on an LCD. This application is designed by using

NIOS II soft-core CPU providing a flexible way for further applications. But the system does not record acquired signal data and has no network connection.

The authors in [6] and [7] propose a DSP and FPGA employed power signal processing systems that are high performance solutions. On the other side, hardware designs of these systems are complex and developing DSP software may require extra effort. Both the systems do not provide database connectivity. ARM and DSP based solutions with operating systems µc/Cos-II and Linux are demonstrated in the papers [8] and [9]. Hardware designs for both solutions are complex as in previous papers. Also, performance measurements and data logging features are not reported.

Power quality analysis by using virtual instrumentation (VI) with LabVIEW are demonstrated in the papers [15] and [16]. Software packages like LabVIEW and MATLAB provides researchers with ready-to-use and powerful functions and visual blocks to evaluate power signal processing algorithms. But these tools require specialized hardware for real-time operation and end-user equipment design is not a trivial task with these tools.

2.2. Real-Time Linux Approaches

There are both commercial and open-source approaches to add Linux real-time operating features. MontaVista Linux and TimeSys Linux can be given as examples for commercial distributions that use modified Linux kernel to meet real-time requirements. Xenomai Framework and Real Time Application Interface for Linux (RTAI) are open-source alternatives (extensions) that use dual-kernel structure. Within this method, standard kernel runs in low priority mode and controlled by real-time kernel [12, 13, 17].

Another approach comes to the stage so called RT-Patch recently. RT-Patch modifies standard Linux kernel to be able to operate in soft real-time domain for intended purposes. On Linux with RT-Patch, user processes can be executed within a priority close to kernel processes. Maintaining and deploying a single kernel operating system may be easier for most cases [12, 13, 17, 18].

A water level control application for two tanks by using RT-Linux and autonomous system tailoring tool (RT-LEAST) is demonstrated in [19]. RT-Linux is another dual kernel method to gain real-time benefit of embedded Linux. Standard Linux kernel runs as a process of RT-Linux with the possible lowest priority. Real-time processes read the water level from sensors and controls the pumps to keep the water level at a desired setpoint. Water levels of tanks can be observed or set by using GUI of the application runs over low priority Linux layer. GUI process and real-time process communicates each other via RT-FIFOs. In this work, measured maximum context switching time is 20 µs and this value is sufficient for such control tasks. Authors also demonstrate how to customize the Linux to run for a resource constraint embedded system by using RT-LEAST.

The authors in [17] demonstrate process delay, jitter and rescheduling time measurements on VxWorks, RTAI, Xenomai and Linux operating systems. VxWorks is a commercial UNIX-like real time operating system (RTOS). Authors developed a test application that reads data from a multi-channel analog-to-digital converter (ADC) board and writes a single output on digital-to-analog (DAC) board corresponding to analog inputs. Test procedure is performed by calculating the time elapsed between ADC conversion and DAC response measured with an oscilloscope. In the first test case, an interrupt service routine (ISR) is used for the measurement of delay time. This ISR reads analog inputs and writes DAC output directly. The minimum delay time is accomplished with VxWorks at 69.20 µs. The minimum jitter is measured with RTAI at 0.15 µs. However, Linux with no-load condition is better than Xenomai with 72.8 µs in delay time and better than VxWorks with 0.4 µs in jitter time. In the second test set, ADC ISR fires a kernel task instead of writing DAC output directly. In that case, rescheduling time is measured and RTAI has the best timing for jitter and rescheduling delay. Delay time is very close for all operating systems. Eventually, both Xenomai and RTAI can be considered good open-source alternatives to

VxWorks. However, RTAI and Xenomai application development is harder than VxWorks because the need for kernel mode development.

The paper [12] evaluates the real-time performance of RT-Patched kernel in comparison to original kernel with the version 2.6.25-4-rt4. They performed two tests by using synthetic real-time and synthetic best-effort task at the same time. The operating system is booted into single-user mode to minimize the effect of other processes on the measurement. In these tests, the activation times and jitter are observed. The results in the first test are nearly the same for both original and patched kernel. First test is performed by using two cores of CPU and real-time task's CPU load is fixed at 25%. The second test realized by using single core of CPU and CPU load of real-time task is fixed at %75. In this test, patched kernel's activation period performance is better than original kernel test.

As seen in previous works, several real-time Linux solutions for control and processing fields, have been developed by engineers up to now. RTAI and Xenomai are the dual kernel approaches including a real-time microkernel and a Linux kernel with low priority. The lack of this approach is the need to maintain the microkernel for different types of hardware platforms. In this approach, low priority kernel runs on hardware abstraction layer (HAL) requiring additional maintenance.

With RTAI, a real-time application is designed as a micro-kernel module making design and debugging tasks are harder. Descendant of RTAI, Xenomai has moved one step ahead the effort by providing developers with userspace application programming interface (API) via an emulation layer. However, the emulation layer development is an additional task over microkernel and HAL. The main advantage of the dual-kernel approach is providing a safer operation by keeping most of original Linux kernel untouched [12, 13].

The single kernel real-time Linux is obtained by patching the original Linux kernel without the need for an extra effort. RT-Patch is directly maintained by kernel developers. Both real-time and non-real-time applications can run on this system without any modification for the new kernel. The paper [13] reports that dual-kernel approach guarantees slightly lower latency and better determinism. But ease of use can make RT-Patch an ideal solution depending on the application. From developer's perspective, developing a real-time application is not too much different than standard userspace application development. Application's main thread or other worker threads are updated with priority modification C function "sched_setscheduler" to obtain real-time version over RT-Patched system [10, 18]. As a result, RT-Patch provides a native method for soft real-time application development.

3. THEORETICAL BACKGROUND

In this study FFT (Fast Fourier Transform), THD (Total Harmonics Distortion), SNR (Signal-to-Noise Ratio) and frequency calculations are implemented in real-time. FFTW3 library is employed for FFT calculation [20]. FFT is one of the most used methods used for harmonics calculation [2, 3]. Digitized AC data is processed in groups of ten periods in accordance with IEC 61000-4-7 specification for 50 Hz power line frequency [2, 3, 20]. Incoming cycle data is appended to the end of the FFT window and the data of the first cycle is disposed of. This is a basic implementation of STFT (Short Time Fast Fourier Transform) transform. STFT is a widely used method for time-frequency analysis of non-stationary signals. STFT performs DFT operation by using moving window approach with adequate overlapping segments of the power signal. STFT can be expressed using the Eq. (1) [22].

$$STFT(t,\omega) = \int_{-\infty}^{\infty} x(\tau)\omega(t-\tau)e^{-j\omega\tau}d\tau \tag{1}$$

where the variable x(τ) is the signal and $\omega(t-\tau)$ is the window function. The discrete form of STFT can be represented in Eq. (2) [3].

$$X_{SFTF}(m,f_k) = \sum_{n=0}^{N-1} x(n)h(n-m)e^{-j\left(\frac{2\pi}{N}\right)kn} \tag{2}$$

where f_k is the k^{th} harmonic frequency and m is an integer indicating the window position on time scale [3]. The frequency resolution Δf can be determined by using Eq. (3).

$$\Delta f = \frac{f_s}{N} \tag{3}$$

where N is represents the number of samples acquired at the sampling frequency f_s [23]. In our case, the frequency resolution is 5 Hz with the values $N=2000$ and $f_s=10$ *ksps*.

Total harmonic distortion (THD) can be expressed as a ratio representing the nonlinear distortion in circuits in which harmonics (signals whose frequency is an integer multiple of the input signal) are generated. THD ratio is measured in percent or in decibels (dB), and can be calculated by using Eq. (4).

$$THD = \frac{\sum_{k=2}^{N} U_k^2}{U_1} \tag{4}$$

where: U_1 amplitude of first (or fundamental) harmonic, U_k amplitude of *k-th* harmonic [15, 21]. This parameter can be calculated by using amplitude values obtained after FFT operation in the previous step. In this study, harmonics up to 100th can be used for THD calculation due to the 10 ksps sampling rate. The quality of power signal can be described by the signal-to-noise ratio (SNR). Signal-to-noise ratio can be calculated by using Eq. (5);

$$SNR_{db} = 10log_{10}\left(\frac{P_x}{P_n}\right) = 20log_{10}\left(\frac{U_x}{U_n}\right) \tag{5}$$

where: U_x amplitude of base frequency signal, U_n sum of amplitudes of other signal components. Amplitudes are used instead of power parameters as in calculation of SNR. In this study, the signal components up to 6th harmonic are excluded in SNR calculation as the sources of noise resides out of the fundamental signal components.

Counting of zero crossings is one of the most commonly used method to calculate the power signal frequency. IEC 61000-4-30 clearly defines the method of the frequency calculation as: "the ratio of the number of integral cycles counted during the maximally 10 second's time clock interval, divided by the cumulative duration of the integer cycles." In this study, frequency measurement is based on zero-crossing method. 10 periods of FFT window is used for calculations. Last eight cycles, taking place in the end of the FFT window, are used for frequency calculation. The number of integer cycles is determined

by counting the consecutive negative to positive transitions. Then the frequency is calculated by dividing elapsed time between first and last zero-crossing points by the number of integer cycles (Eq. 6);

$$f = \frac{t[zci_n] - t[zci_0]}{n} \tag{6}$$

where f is the line frequency, z_{ic} is the zero-crossing index, t is the time vector of data samples, n is the number of integer cycles. 10 periods are used to obtain the lowest possible spectral leakage [3].

4. ARCHITECTURE OF REAL-TIME HARMONICS DETECTION SYSTEM

In order to evaluate of power signal processing within real-time Linux environment, a data acquisition device and a single board computer are used. Power signal data are digitized and streamed to the computer through USB connection. On the computer side, a power signal processing application runs to perform signal processing and required calculations tasks.

The data acquisition device is proposed in our previous work [24]. The device has a high-speed USB-FIFO interface chip: FT2232H. This chip has two communication channels. First channel is used for control data acquisition (eg. start or stop acquisition); second channel is used for buffering and transferring signal data (Fig. 1). Streaming data is buffered by using of 4 KB on-chip memory.

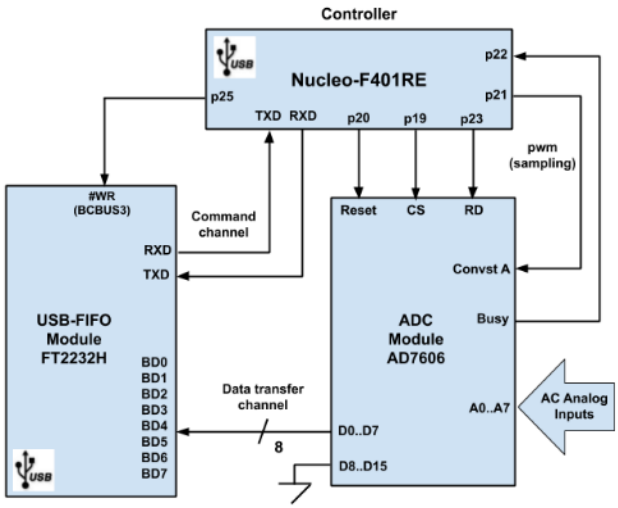

Figure 1. Power signal data acquisition device.

On the computer side, power signal processing application incorporates FTD2XX driver library so as to read streaming data. This application creates two threads with the same priorities at startup: thDataGetter and thDataProcessor. First thread, thDataGetter is used to read data brought by device driver of FT2232H. Then the data is queued by using C++ queue type using push method. Flowchart of this operation is given in Figure 2. ADC data is queued before processing to avoid data overflow in hardware FIFO.

Second thread thDataProcessor pops queued signal data from the queue. This data is parsed into related channels. Parsed data for each channel is added to end of the signal window, which is ten periods length. Finally power signal data is processed as voltage-current pairs for three phases (Six channels of ADC is used). FFT, THD, SNR and frequency calculations are implemented in this sequence. For every new period of analog data (20 ms), these operations are repeated as seen in Fig. 3. These calculations are more time-

consuming than the previous implementations in [17] and [19]. For the real-time versions of the threads, memory locking function "mlockall" is used to avoid remapping page tables causing latencies in context switching [17].

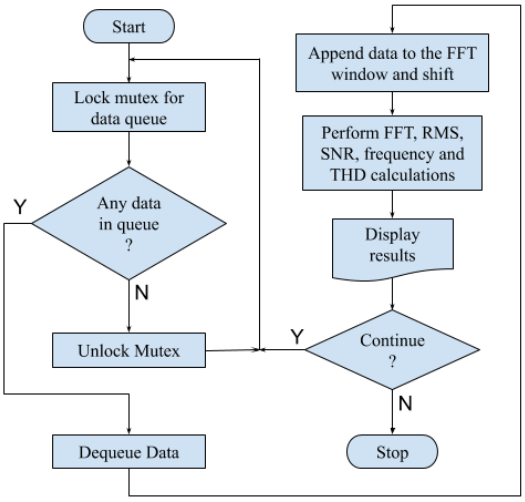

Figure 2. Flowchart of thread thDatagetter.

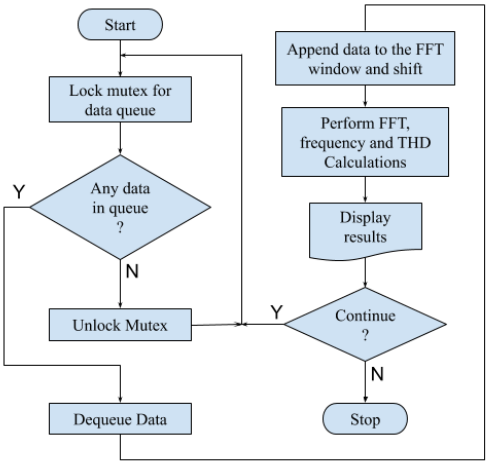

Figure 3. Flowchart of thread thDataProcessor.

In this application, signal sampling rate is chosen as 10 ksps. For every period of acquired data in 8 channels, 3200 bytes are transferred to the processing computer. Raw signal data is recorded to SQL compatible database by using SQLite 3 database engine [25]. Data is inserted to relevant table inside thDataGetter thread. SQLite database write operations are performed by using "synchronous off" option in order to protect a real-time thread from being blocked by a disk write operation. This option is safe unless a power loss or an operating system crash according to SQLite documentation.

The signal processing application is a console application built with ncurses library as seen in Figure 4. This library provides easy-to-use application programming library (API) to create text-based user interface design. Signal processing application has two versions: real-time and non-real-time. The real-time version of the application is obtained by adding priority modification calls to the non-real-time version.

Recorded raw signal data can be downloaded over network by a web interface running independently from data acquisition and data processing application shown in Fig. 5. This web interface is served by using Python and Flask. Flask is a micro framework for Python web application development [26]. The

web application serves acquired data from database file to user interface via AJAX (Asynchronous JavaScript and XML) calls. Also, power signal processing application can be controlled remotely by using web interface. In this way complexity of the application is resolved by using a low-priority separate application model.

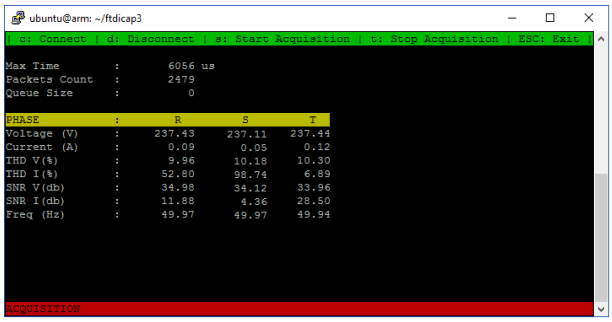

Figure 4. Native user interface of the proposed application.

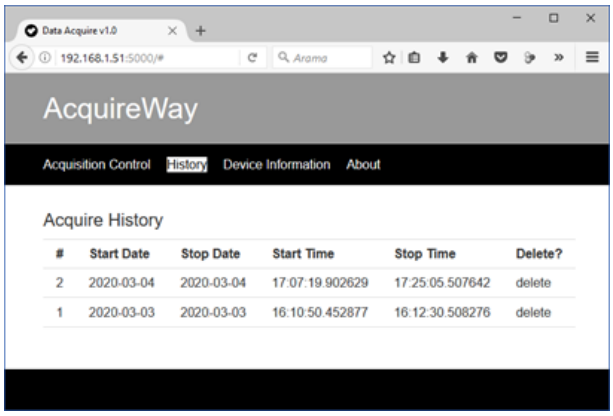

Figure 5. Web-based interface of proposed application.

5. EXPERIMENTAL SETUP

Performance tests of the system are realized by measuring the harmonics on an AC power line with an RMS value of 220 volts with 50 Hz as fundamental frequency. SCT-013-30A split core current transformer is used to convert current to voltage. A potential transformer with 6 volts RMS seconder voltage is used to reduce the voltage of the line seen in Fig. 6. Seconder voltage is applied to differential inputs by using voltage divider to half. Single phase voltage and current are monitored in this way illustrated in Fig. 7. Other phases of the line are monitored by using same scheme.

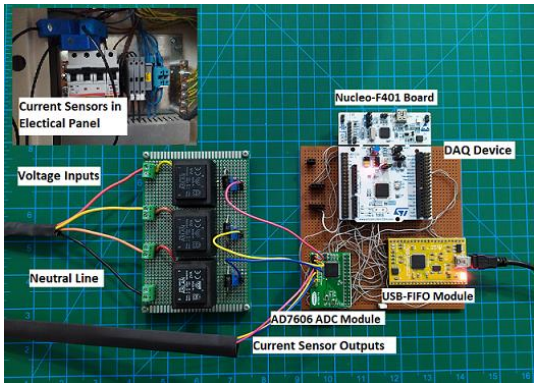

Figure 6. Photograph of the experimental setup.

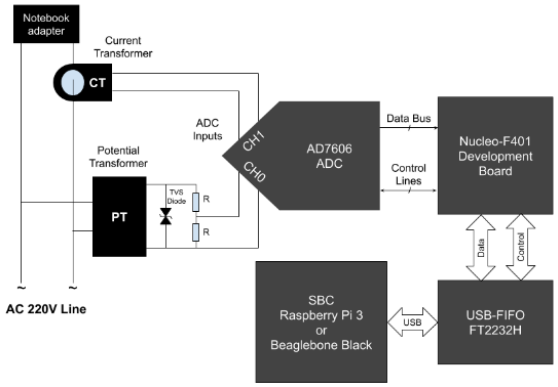

Figure 7. Block diagram of the testbed.

6. PERFORMANCE RESULTS

Performance measurements have been carried out on single board computers Raspberry Pi 3 and Beaglebone Black by using both real-time and non-real-time versions of the application. On Beaglebone Black, Ubuntu 16.04 OS with kernel version 4.1.29-bone-rt-r22 is used and on Raspberry Pi 3, Debian 8 OS with kernel version 4.4.9-rt17-v7+ is preferred. Both real-time and non-real-time tests have been performed with these kernels.

Package processing time is the metric to exhibit the differences of two versions. Processing time is the sum of the computation times of FFT, RMS, Frequency, THD and SNR parameters. Timing calculations are observed for 2500 periods of power line signal (During 0.02 x 2500 = 50 sec).

For the non-real-time versions of application, the performance measurements are given in Table 1. In this case, there are no other user application or operation running on the systems. These test results show that processing time takes lower time on the Raspberry Pi 3 with multi-core CPU as expected. Standard deviation on Raspberry Pi 3 is significantly lower than on Beaglebone Black as can be seen in Figure 8. The horizontal axis of the graph indicates the number of data processing sequence repeated for every period of ac signal data; the vertical axis indicates the time taken by processing phase. According to this measurement, both SBCs are able to meet real-time requirements.

Table 1. Processing times for the non-real-time versions of proposed software with no load.

Processing Time (µs)	Beaglebone Black	Raspberry Pi 3
Minimum	5669	2307
Maximum	21778	8038
Mean	7785	2516
Standard Deviation	2548	233

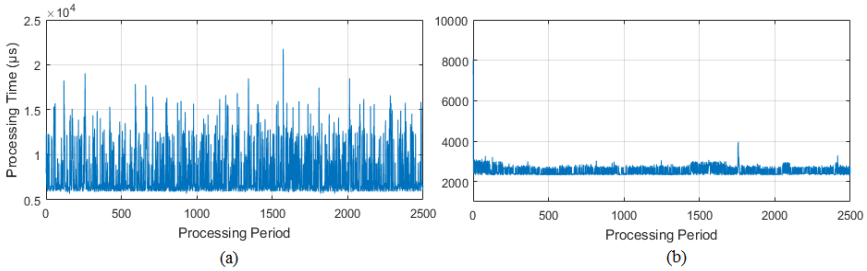

Figure 8. Processing times for non-real-time version with no load; (a) Beaglebone Black, (b) Raspberry Pi 3.

The results given in Table 2 are the measurement results when a file copy operation over Secure Shell (SSH) performed with non-real-time version of signal processing application. File copy operation uses CPU, memory, network and disk resources together. Therefore, this operation is selected to realize a real load condition for the OS. The starting and finishing times of the file copy operations can be clearly observed in the graph given in Figure 9. For the Raspberry Pi 3, timings do not change much between load and no-load conditions. These results show that the multi-core and multi-threaded application model can be considered as a solution for real-time applications. Non real-time version of the proposed application on Beaglebone Black has moved away from meeting real-time requirements under load condition.

Table 2. Processing times for non-real-time versions of proposed software under load.

Processing Time (µs)	Beaglebone Black	Raspberry Pi 3
Minimum	5530	2313
Maximum	99130	18724
Mean	16821	2708
Standard Deviation	12728	1201

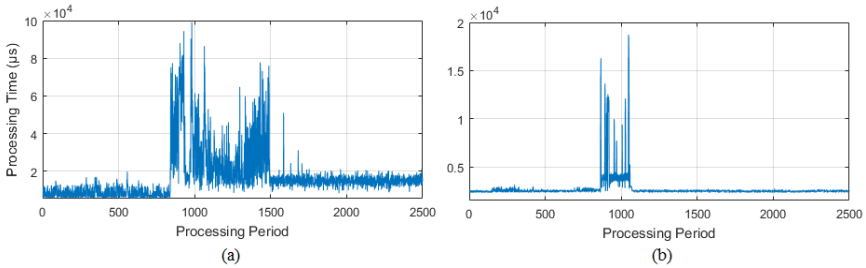

Figure 9. Processing times for non real-time version under load; (a) Beaglebone Black, (b) Raspberry Pi 3.

Maximum processing time for Beaglebone Black is too high (99130 µs) and this duration may cause overflow of FIFO buffer on FTDI interface chip. File copy operation takes about 20 seconds on Beaglebone Black and about 5 seconds on Raspberry Pi 3.

The performance measurements for real-time version of the application are given in Table 3. Minimum, maximum and mean processing times show that both Raspberry Pi 3 and BeagleBone Black are suitable for a multi-channel power quality parameter calculation application. Low deviation of processing delays also can be observed in Fig. 10.

Table 3. Processing times for real-time versions of proposed software with no load.

Processing Time (µs)	Beaglebone Black	Raspberry Pi 3
Minimum	5124	2298
Maximum	6014	2955
Mean	5681	2407
Standard Deviation	78	50

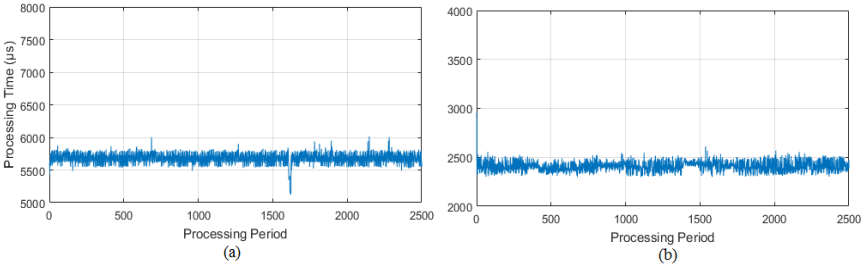

Figure 10. Processing times for non real-time version under load; (a) Beaglebone Black, (b) Raspberry Pi 3.

The performance measurements under load condition for real-time version of the application are given in Table 4. Under load condition Raspberry Pi 3 performs the calculations under 5 ms. The jitter in processing time is more on Beaglebone Black. The maximum processing time on Beaglebone Black is lower than non-real-time version, though. This is a significant enhancement over non real-time version. During file transfer operation, on Beaglebone Black maximum processing time is 8.2 ms satisfying the real-time requirement. This processing time is far lower than non-real-time version of the application running on Beaglebone Black under load condition. The changes in processing times can be observed in Figure 11.

Table 4. Processing times for real-time versions of proposed software under load.

Processing Time (µs)	Beaglebone Black	Raspberry Pi 3
Minimum	5114	2258
Maximum	8222	2990
Mean	5987	2475
Standard Deviation	491	103

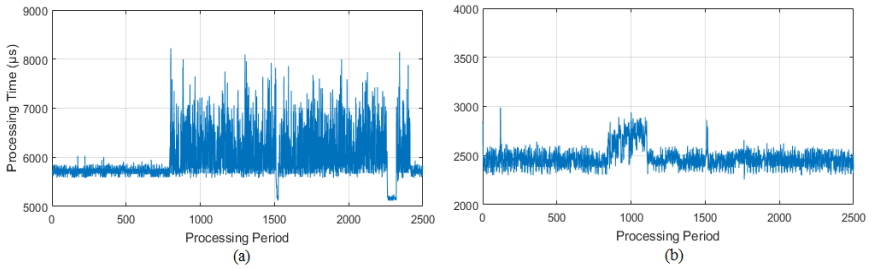

Figure 11. Processing times for the real-time versions of the application under load; (a) Beaglebone Black, (b) Raspberry Pi 3.

In this experiment, file copy operation takes about 30 seconds on Beaglebone Black and 6 seconds on Raspberry Pi 3. Beaglebone Black spends nearly 10 seconds more for file transfer operation because real-time tasks steals more time from kernel-level tasks in this experiment. The difference is about 1 second for Raspberry Pi 3 because of CPU core-distributed tasks. As seen in Figure 10 and Figure 11, real-time operation presents more stable processing times.

7. CONCLUSIONS

This study demonstrates a real-time power quality parameter calculation application on single board computers Beaglebone Black and Raspberry Pi 3 running on RT-patched Linux. On the first occasion, Raspberry Pi 3 is the winner for both real-time and non-real-time versions of the application under load and no-load conditions. This result shows that how a multi-threaded application model can be efficient on multi-core CPUs. However, choosing the real-time version of the application is the fair way to assure the real-time operation due to non-deterministic nature of the scheduler. On the second occasion, non-real-time version of the application running on Beaglebone Black fails under load conditions. This is an expected situation due to the kernel level operations have higher priorities than user level processes. Beaglebone Black has a single core CPU which cannot handle this type of operations. Real-time version of the application performs real-time operation in 11 ms satisfying the requirements as in no-load condition. In this case real-time threading keeps the calculations being suspended by kernel level processes. This result shows that a single board computer with single-core CPU can be used for real-time power system harmonics detection and power metering applications with RT-pathed OS.

For the future work, ADC will be interfaced to Beaglebone Black processor via an FPGA by eliminating the need for USB connection. Latencies due to the USB packet processing mechanism, will be further decreased in this way. Test cases will be repeated by employing both RT-Patched kernel and Xenomai Framework in a comparative manner.

REFERENCES

[1] Yılmaz, A.S., Alkan, A., & Asyalı, M.H. Applications of parametric spectral estimation methods on detection of power system harmonics. *Electr Power Syst Res* 2008, *78*(4), 683-693.

[2] Sadinezhad, I. & Agelidis, V.G. Slow sampling on-line harmonics/interharmonics estimation technique for smart meters. *Electr Power Syst Res* 2011, *81*(8), 1643-1653.

[3] Jain, S.K. & Singh, S.N. Harmonics estimation in emerging power system: key issues and challenges. *Electr Power Syst Res* 2011, *81*(9), 1754-1766.

[4] Li, P., Li, X., Li, J., You, Y., & Sang, Z. A real-time harmonic extraction approach for distorted grid. *Mathematics* 2021, *9*(18), 1-20.

[5] She, X. & Xiong, J. Multi-channel electrical power data acquisition system based on AD7606 and NIOSII. *2011 International Conference on Electrical and Control Engineering* 2011, 1625-1627.

[6] Wenyi, L. & Hongcheng, Y. Design of high speed synchronous multi-channel data acquisition and processing system based on TMS320C6747. *2010 The 2nd International Conference on Computer and Automation Engineering* 2010, 758-760.

[7] Zhang, M. & Li, K. DSP-FPGA based real-time power quality disturbances classifier. *2010 IEEE PES Transmission and Distribution Conference and Exposition* 2010, 1-6.

[8] He, Z. & Liao, Y. The design of analog acquisition system in distribution automation. *2012 China International Conference on Electricity Distribution* 2012, 1-4.

[9] Wang, A., Pan, F., Li, Y., & Tao, R. The design of power quality detecting system based on OMAP-L138, *IEEE 13th Workshop on Control and Modeling for Power Electronics* 2012, 1-4.

[10] Dias, R.A., Souza, T.E., & Noll, V. Experimental analysis of the Linux RT-patched for acquisition applied to power sector. *Int J Comput Appl* 2014, 101, 43-49.

[11] Perneel, L., Fayyad-Kazan, H., & Timmerman, M. Can Android be used for real-time purposes? *2012 International Conference on Computer Systems and Industrial Informatics* 2012, 1-4.

[12] Betz, W., Cereia, M., & Bertolotti, I.C. Experimental evaluation of the Linux RT patch for real-time applications. *2009 IEEE Emerging Technologies & Factory Automation Conference* 2009, 1-4.

[13] Vun, N., Hor, H.F., & Chao, J.W. Real-time enhancements for Embedded Linux. *2008 14th International Conference on Parallel and Distributed System* 2008, 737-740.

[14] Vujović, V. & Maksimović, M. Raspberry Pi as a Sensor Web node for home automation. *Comput Electr Eng* 2015, 44, 153-171.

[15] Chen, Y. Research and design of intelligent electric power quality detection system based on VI. *J Comput* 2010, *5*(1), 158-165.

[16] Miron, A., Chindriş, M.D., & Cziker, A.C. Software tool for real-time power quality analysis. *Adv Electr Comp Eng* 2013, *13*(4), 125-132.

[17] Barbalace, A., Luchetta, A., Manduchi, G., Moro, M., Soppelsa, A., & Taliercio, C. Performance comparison of VxWorks, Linux, RTAI and Xenomai in a hard real-time application. *IEEE Trans Nucl Sci* 2008, *55*(1), 435-439.

[18] Reghenzani, F., Massari, F., & Fornaciari, W. The real-time Linux kernel: A survey on Preempt_RT. *ACM Comput Surv* 2019, *52*(1), 1-36.

[19] Vidal, J., Mendoza, P., Vila, J., Crespo, S., & Sáez, S.A. Minimal RT-Linux embedded system for control applications. *2002 IFAC 15th Triennial World Congress* 2002, 249-254.

[20] Frigo, M. & Johnson, S.G. The design and implementation of FFTW3. *Proceedings of the IEEE* 2005, *93*(2), 216–231.

[21] Radil, T. & Ramos, P.M. Methods for estimation of voltage harmonic components. *Power Quality, A. Eberhard (Ed.), Rijeka, Croatia: InTech* 2011, 13, 255-270.
[22] Han, J., Kim, W., & Kim, C. Fault type classification in transmission line using STFT. *11th IET International Conference on Developments in Power Systems Protection* 2012, 1-5.
[23] Lima, A.A.M., Cerqueira, A.S., Carlos, A.D., & Oliveira, E.J. Estimation of harmonics and interharmonics based on single channel independent component analysis. *16th International Conference on Harmonics and Quality of Power* 2014, 298-302.
[24] Gök, M., Görgünoğlu, S., & Sefa, İ. Design of a real-time USB interfaced multi-channel power system harmonics detection system. *9th International Conference on Electrical and Electronics Engineering* 2015, 521-524.
[25] SQLite database engine. Accessed on: Oct. 3, 2020. [Online]. Available: https://www.sqlite.org
[26] Flask: a Python micro-framework for web. Accessed on: Oct. 3, 2020. [Online]. Available: https://flask.palletsprojects.com