TU Delft

Delft University of Technology

Smart DFT Based PMU Prototype

Radević, Isidora; Naglic, Matija; Mansour, Omar; Bijwaard, Dennis; Popov, Marjan

DOI
10.1109/SGSMA.2019.8784477

Publication date
2019

Document Version
Final published version

Published in
2019 International Conference on Smart Grid Synchronized Measurements and Analytics, SGSMA 2019

Citation (APA)
Radević, I., Naglic, M., Mansour, O., Bijwaard, D., & Popov, M. (2019). Smart DFT Based PMU Prototype. In *2019 International Conference on Smart Grid Synchronized Measurements and Analytics, SGSMA 2019* (pp. 1-7). Article 8784477 (2019 International Conference on Smart Grid Synchronized Measurements and Analytics, SGSMA 2019). IEEE. https://doi.org/10.1109/SGSMA.2019.8784477

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

Green Open Access added to TU Delft Institutional Repository

'You share, we take care!' - Taverne project

https://www.openaccess.nl/en/you-share-we-take-care

Otherwise as indicated in the copyright section: the publisher is the copyright holder of this work and the author uses the Dutch legislation to make this work public.

Smart DFT Based PMU Prototype

Isidora Radevic[(1)], Matija Naglic[(1)], Omar Mansour[(2)], Dennis Bijwaard[(2)], Marjan Popov[(1)]

[(1)] Delft University of Technology, Delft, The Netherlands
[(2)] Smart State Technology, Hengelo, The Netherlands

e-mail address: i.radevic@tudelft.nl

Abstract— **This paper presents recent innovation in the field of the Synchronized Measurement Technology. An advanced, low complexity algorithm for synchrophasor estimation and the processing platform running embedded Linux are used to develop a cost-efficient Phasor Measurement Unit (PMU) prototype. The Smart Discrete Fourier Transform (SDFT) is adapted for the purpose of synchrophasor estimation, being implemented by using Python programming language. The developed prototype sends synchro-measurements according to the IEEE Standard C37.118 specifications. The prototype performance characteristics are evaluated by using RTDS power system simulator as hardware-in-the-loop. The obtained results suggest further algorithm improvements to fully comply with the IEEE Standard C37.118.1a-2014 specified requirements. The developed prototype offers an affordable PMU solution for improving the grid observability.**

Index Terms—**Phasor Measurement Unit, Discrete Fourier Transform, embedded platform, real-time measurements, enhanced grid monitoring.**

I. Introduction

This paper presents an advanced technique for synchrophasor estimation and the design and application of a PMU prototype. Typically, a PMU device is used to monitor power system dynamics by providing voltage and current synchrophasors, frequency and rate-of-change-of-frequency measurements with 50 frames per second reporting rate. Despite significant benefits of PMU measurements compared to typical SCADA type of measurements, PMUs are still not massively deployed in the grid.

The commercially available PMUs are typically based on a modular design, composed of a processor board, transformer boards, GPS time synchronization and communication modules. The choice of the implemented estimation technique and the final product performance depend on the main processor performance (measured with respect to the clock frequency and multi-core processing possibilities) and the accuracy of the remaining components. To provide an affordable solution without jeopardizing the performance capabilities, in this research, the time synchronization is realized in the platform hardware instead of software. For waveforms sampling, a multi-channel time-synchronized analogue to digital converter (ADC) board is used. For algorithm processing and implementation, an affordable power-efficient ARM CPU boards (Raspberry-Pi [1] like, the family of embedded single board computers) are selected, with the embedded Linux operating system.

The main challenge is to choose an efficient and accurate synchrophasor estimation algorithm in order to obtain the best possible results accuracy, meanwhile guaranteeing real-time processing on the applied embedded hardware. So far, many algorithms were used by different authors, utilizing variable window length [2] and variable sampling rate [3] based algorithms, interpolated-iterative DFT techniques [4] and techniques that relay on estimated system frequency. Generally, the complexity of the algorithms differs based on the applied mathematical approach. Besides, the algorithm should be computationally efficient and should provide accurate estimates under different grid conditions. In [5], a Smart Discrete Fourier Transform (SDFT) computationally efficient method for precise calculation of power system frequency is introduced. Additionally, it was shown that the frequency estimates can be further used to recalculate the phasor values. In this way, less erroneous results are obtained when frequency deviates from the nominal value. In [6], the improvements of the proposed SDFT were investigated for synchrophasor applications by implementing the SDFT based PMU model in a real-time simulator.

The aim of this work is to implement the enhanced version of SDFT into the processing platform and investigate its performance capabilities according to the IEEE Standard C37.118.1a-2014 requirements. In the first step, the efficiency of the basic SDFT is verified according to the standard requirements under different testing conditions, by using MATLAB simulation environment. In next step, the Python programming language is used to implement the enhanced SDFT algorithm on the processing platform. In order to keep within the processing resources limitations of the embedded platform, a number of code optimization is performed.

978-1-7281-1607-5/19/$31.00 ©2019 European Union

II. Procedure for Algorithm Development and Verification

A. Smart Discrete Fourier Transform

SDFT method for phasor estimation is based on a mathematical approach, that efficiently solves limitations of the standard DFT method. The method relies on frequency estimation obtained by three consecutive DFT fundamental components. By using an estimated frequency, one SDFT phasor is obtained, having significantly higher accuracy than previously estimated DFT phasors. The SDFT algorithm requires additional computational steps compared to the standard DFT, but the benefits of reduced filtering requirements outweigh additional processing expenses. The following equations shortly illustrate the flow of SDFT [5], [6]:

$$x(k) = X_m \cos\left(\frac{2\pi f k}{N f_0} + \phi\right) \tag{1}$$

being: $x(k)$ a pure sinusoidal signal sampled at discrete instants, X_m the input signal amplitude, f the signal frequency, ϕ the initial phase angle, f_0 the nominal signal frequency and N the sampling rate in samples/cycle.

The main frequency component in DFT spectrum of $x(k)$, evaluated at the r^{th} sample is given as:

$$\hat{x}_r = \frac{2}{N}\sum_{k=0}^{N-1} x(k+r)e^{-j\frac{2\pi k}{N}} \tag{2}$$

If we consider the system frequency deviation to be Δf:

$$f = f_0 + \Delta f\,, \tag{3}$$

and parameters $\hat{x}_r$ and a as:

$$\hat{x}_r = A_r + B_r \tag{4}$$

$$a = e^{j\frac{2\pi(f_0+\Delta f)r}{N f_0}}\,, \tag{5}$$

the following relations can be obtained:

$$w = a + a^{-1} = 2\cos\frac{2\pi(f_0+\Delta f)}{N f_0}\,, \tag{6}$$

$$A_r = \frac{X_m}{N}\frac{\sin\frac{\pi\Delta f}{f_0}}{\sin\frac{\pi\Delta f}{N f_0}} e^{j\left(\frac{2\pi(f_0+\Delta f)r}{N f_0} + \frac{\pi(N-1)\Delta f}{N f_0} + \phi\right)}\,. \tag{7}$$

while B_r can be neglected, since the Δf is small.

The frequency deviation Δf can be derived as:

$$\Delta f = \frac{N f_0}{2\pi}\cos^{-1}\Re\left(\frac{w}{2}\right) - f_0\,, \tag{8}$$

and the amplitude X_m and the phase ϕ can be extracted:

$$X_m = |Ar|\frac{N\sin\frac{\pi\Delta f}{N f_0}}{\sin\frac{\pi\Delta f}{f_0}} \tag{9}$$

$$\phi = angle(A_r) - \frac{\pi(N-1)\Delta f}{N f_0}. \tag{10}$$

The derived expressions for the parameters w and A_r are:

$$w = \frac{\hat{x}_r + \hat{x}_{r+2}}{\hat{x}_{r+1}} \tag{11}$$

$$A_r = \frac{a\hat{x}_{r+1} - \hat{x}_r}{a^2 - 1}\,. \tag{12}$$

Finally, after calculating the values of Ar, and Δf, the value of the estimated SDFT phasor can be obtained.

B. SDFT Based PMU Algorithm

As stated in [6], the simulations with a PMU model based on SDFT shows numerical oscillations in estimated frequency. In order to get more accurate values, the estimate w is filtered by using a mean filter with an order of 1.5*N*.

C. Verification of SDFT in MATLAB

The performance of the SDFT algorithm is verified through the set of test conditions according to the IEEE Standard C37.118.1-2011 and IEEE Standard C37.118.1a-2014. At this point the mean filter is not implemented. Developed MATLAB script can be presented with the following pseudo-code:

Step 1 - perform sliding DFT algorithm for the complete duration of the input signal (80 samples/cycle).
Step 2 - by taking three consecutive DFT fundamental components, estimate the frequency and recalculate the phasor for a complete input signal.
Step 3 - calculate TVE and errors for the estimated frequency by considering the reference and estimated values.

Different test conditions are simulated by changing the input signal model and corresponding reference phasor:

Test 1 - Nominal and Off-nominal input signal frequency
Test 2 - Frequency ramp
Test 3 - Input signal magnitude modulation (Test 3.1) and phase modulation (Test 3.2)
Test 4 - Step changes in input signal magnitude and phase

The simulation results are partially shown in Fig. 1 through Fig. 9. It can be noticed that oscillations occurring in frequency estimations (Fig. 6) affect the phase estimation accuracy (Fig. 7). Additionally, under the condition of modulated phase, the limit of frequency estimation error (0.3Hz) was violated by 0.2Hz (Fig. 9). However, the TVE is within the limitations for all simulated cases being less than 1% for (quasi) steady-state conditions (Fig. 1) and less the 3% for dynamic conditions (Fig. 5) [7].

D. SDFT Implementation using Python

Following the SDFT validation in MATLAB, the code is optimized for real-time implementation in the embedded platform.

The post signal processing (block processing) approach is used by default in MATLAB implementation. For embedded systems, Python implementation is restricted by real-time constraints, requiring the usage of state variables, loop control and direct stream-processing of the arrived samples in order to meet the algorithm requirements within limited processing resources.

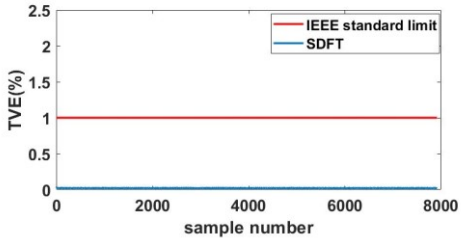

Fig 1. TVE (Test 2)

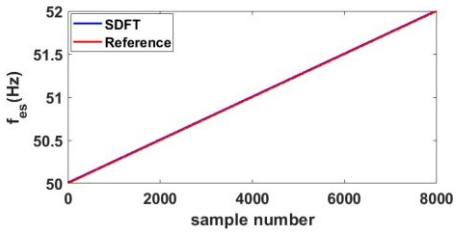

Fig 2. Frequency estimation (Test 2)

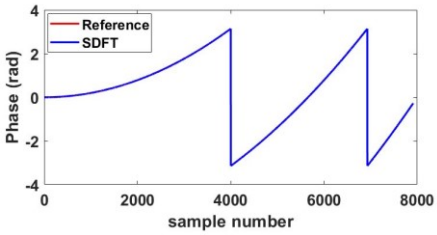

Fig 3. Phase estimation (Test 2)

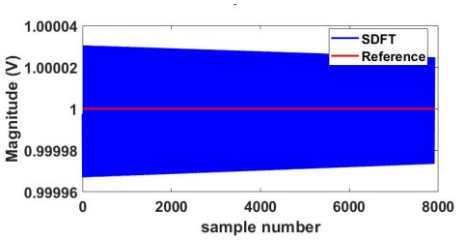

Fig 4. Magnitude estimation (Test 2)

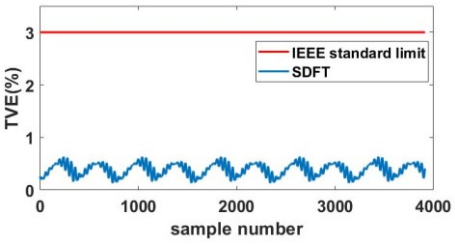

Fig 5. TVE (Test 3.1)

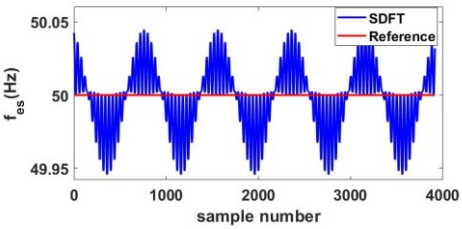

Fig 6. Frequency estimation (Test 3.1)

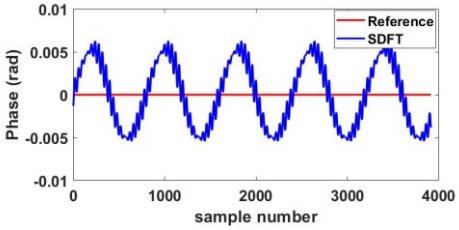

Fig 7. Phase estimation (Test 3.1)

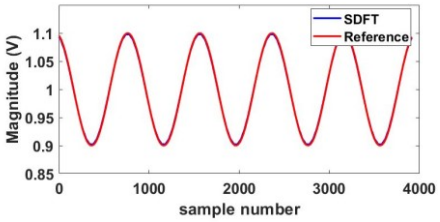

Fig 8. Magnitude estimation (Test 3.1)

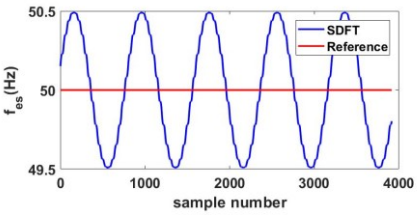

Fig 9. Estimated frequency (Test 3.2)

In order to achieve CPU time and network bandwidth efficiency, multiple samples (40 in this case) are sent in one message introducing an initial measurement delay of 0.01s. In order to utilize the available processing power, many optimization steps are taken. Initially, the A/D sampling rate is set to 4kHz. The data window is chosen to be 80 samples long (one cycle of the input signal at 50 Hz). The flow of DFT and SDFT calculations and the loop control is explained in [8]. Non-recursive approach for DFT calculations is initially performed, causing that the CPU utilization is significantly increased affecting the performance of the sampling module and introducing a delay in real-time processing of samples. Many different ways for minimizing the CPU utilization are considered such as decreased precision in the calculations and decreased number of redundant calculations by using constant variables. For the purpose of implementation the recursive DFT method is used to further reduce the CPU utilization. The new parameter named *max_recursions* is introduced to define the number of recursive DFT calculations performed between two non-recursive DFT calculations. The frequency estimation and calculation of SDFT phasors appeared to be computationally demanding when considering each DFT in a sliding way. Therefore, frequency and SDFT phasors are estimated and reported only 10, 20 or 50 times/s. Due to the availability of frequency values only at reporting moments, the mean filter could not be implemented because of the significant delay that would be introduced in that way.

After performing the set of test experiments, it is noticed that the frequency estimates oscillate around the mean value, affecting the phasor estimation accuracy. Also, the recursive DFT calculation causes numerical oscillations in the estimated magnitude in case of off-nominal input signal frequencies.

The solution is found in a decreased number of samples considered in calculations. Each forth value is taken, that is equivalent to the reduction of ADC sampling rate to 1 kHz. The size of the array is decreased to 40 and window length is set to 20. All other state variables are scaled accordingly. In this way, CPU utilization is significantly decreased enabling the following features:

- The number of non-recursive DFTs can be increased
- Frequency and phasor estimation can be calculated for each new DFT value.
- The mean filter of the 1.5 N order can be implemented. 2.5N + 1 frequency estimates are used to estimate a phasor. The time tag is set at the middle of the window.

The final optimization step is the hardware reduction of the sampling rate of A/D converter from 4 kHz to 1 kHz, resulting in the most efficient CPU utilization of the embedded platform.

As a result, following two implementation versions of SDFT are realized:

Basic SDFT (b-SDFT): The sampling rate of the ADC is 4 kHz. The algorithm makes use of a recursive DFT approach. The frequency and SDFT phasors are estimated in reporting moments. The mean filter is not implemented.

Enhanced SDFT (e-SDFT): The sampling frequency is reduced to 1kHz. The number of non-recursive DFT calculations is increased. The algorithm includes the mean filter for frequency estimates.

The detailed implementation codes can be found in [8].

E. PyPMU Library and PMU Connection Tester

The estimates are reported in a data format according to the IEEE C37.118 - 2005 standard for Data Transfer, by using pyPMU Python Library [9]. PMU Connection Tester is used to validate, test and troubleshoot connections and data streams from the voltage processing platform and graphically visualize the synchophasor estimates in real-time [10].

III. EMBEDDED PLATFORM

The embedded platforms, developed at Smart State Technology (SST) [11], [12], provide open access to synchronized real-time grid measurements in order to easily experiment with new algorithms in the LV grid for future intelligent distribution networks.

The platforms have variable sampling rates that can go up to 128 kHz, 3 phase-voltage measurement ranges that can go up to 600V and 4-phase current measurement ranges that can go up to 600A. The 4-phase current measurements allow for measuring the 3 phases and the neutral, which can be beneficial in networks with high impedance earthing. In order to cope with the variation in measurement ratings, the system allows to use different split-core CT's (and/or Rogowski coils) each with its own rating. Because of these features the platform has a sufficient basis for the PMU prototype described in this paper. In order to obtain high sample synchronization at various measurement locations, the system utilizes dedicated time beacon transmitters (TBT) as it is shown in Fig. 10. The TBT has an embedded GPS receiver for receiving accurate GPS time information. It transmits the Pulse Per Second (PPS) time information wirelessly to all platforms within its measurement cluster. The transmission delay is sub-microsecond range and introduces constant time-synchronization error.

Besides, the embedded platform can also receive the PPS information directly from the GPS receivers or PTP supported time sources, which makes them suitable for various measurement scenarios and topologies. At the platform side the received PPS information is used within a dedicated closed control loop (see Fig. 11) to synchronize the analogue to digital converter (ADC) with global time. This means that ADC sampling is in lock with the GPS time and thus the acquired samples (samples with similar indexes) from different platforms can be correlated to the same time instance. In order to avoid loss of time synchronization in the platform, in case of losing wireless PPS information, the control loop implements a PPS-tracking-estimator algorithm that uses additive time information from the embedded computer.

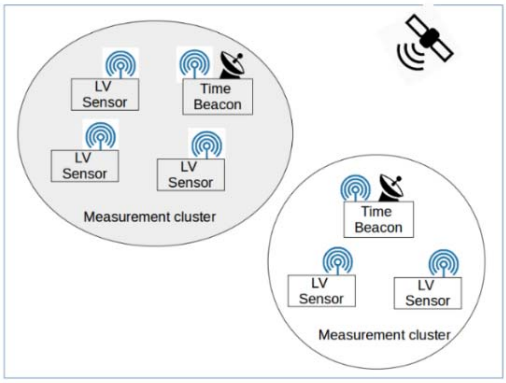

Fig 10. Synchronization of various platform clusters using Time Beacon and GPS. GPS PPS information is broadcast wirelessly with the measurement cluster

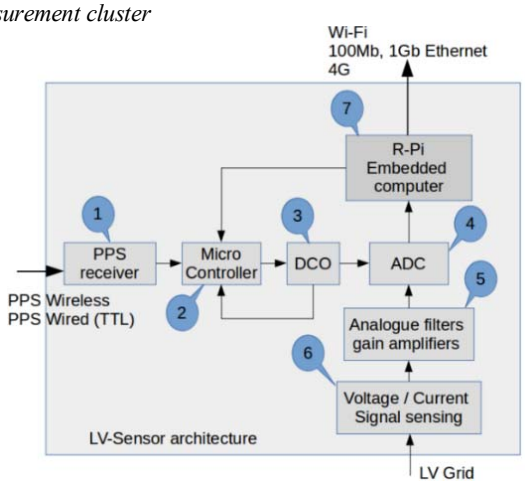

Fig 11. The architecture of the platform with an embedded control loop for ADC sample synchronization. 1) wireless PPS receiver, 2) embedded microcomputer which implements the time synchronization control loop, 3) digital controlled oscillator, 4) analogue to digital converter, 5&6) signal sensing, filters and amplifiers, 7) embedded computer board

The PPS-tracking-estimator algorithm calculates the frequency and phase deviation between the PPS and the computer time and it can extrapolate accurate future PPS arrival times when the frequency and phase deviations are stable. The measurement time information comes from the embedded platform processor board which synchronizes time with Network Time Protocol (NTP) or Precision Time Protocol (PTP, IEC 1588) over Ethernet. This time information is usually accurate enough for synchronization since the PPS pulse information is used. When the kernel time of two platforms is not more out-off sync than 0.5 seconds, the samples can be lined up using the sample sequence number sn, which is reset to 0 at the PPS pulse by the kernel module that reads the ADC. The GPS/UTC sample-time t can be derived by using the kernel timestamp ts with the following formula:

$$t = floor\left(ts + 0.5 - \frac{sn}{F_s}\right) + \frac{sn}{F_s}, \qquad (13)$$

where F_s is sampling frequency.

The embedded platform is an Arm-core, raspberry-Pi (like) computer board [13]. The software framework and drivers on the platform take care of sampling rate configuration of the ADC. The real-time platform information and calculated values such as raw samples, estimated phasors and frequency are made available in a flexible and distributed manner using ZeroMQ messaging [14], such that this information can easily be used both on the platform and on other (embedded) computers in the same network.

IV. Results

The testing of the PMU prototype is realized by using RTDS, real-time power system simulator [15]. Different testing conditions are simulated by the PMU Test Utility Tool for RTDS. However, the complete testing is not done in this tool at this point. Instead, the estimated values are extracted from the embedded platform to MATLAB and graphically plotted. The part of the test setup is shown in Fig. 12.

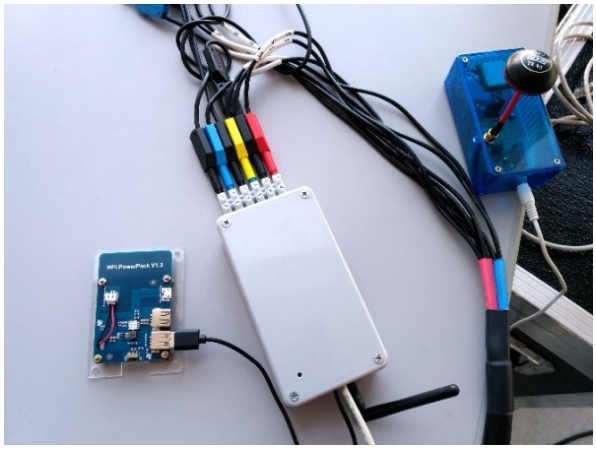

Figure 12. Test Setup – Connection of the embedded platform to the amplifier.

A. Testing under the nominal frequency conditions

The testing of the estimation accuracy obtained by PMU prototype started by applying a pure voltage sine wave at 50Hz, with the magnitude of 100Vrms and phase of 0 radians, to the low voltage processing platform. In Fig. 13, estimated magnitudes for all three phases are shown. It can be seen that the most precise estimate is obtained for phase1, while phase2 and phase3 experience an offset of maximum 0.5 Vrms from the nominal value. The estimates are obtained by running the basic (b-SDFT) algorithm in the duration of 7s.

After introducing the enhancements in the algorithm, by decreasing the number of recursive DFTs and implementing the mean filter, the accuracy of the estimated magnitudes is improved. The maximum error is decreased to 0.05 Vrms, particularly present in the phase3 (Fig. 14). The estimates are obtained for the time period of 12s.

As it can be seen in Fig. 15, the estimated phases are shifted by 120 degrees (since 120 degrees equals 2.0994). The phase value depends on the sample sequence when the e-SDFT starts. However, by choosing the phase1 as a reference with the initial phase 0, two other phases would get the values equal to -120 and -240 (120 degrees) respectively.

The result of frequency estimation is given in Fig. 16. The errors of almost 0.03 Hz caused by a basic algorithm are well suppressed by introducing the mean filter.

B. Testing under the off-nominal frequency condition

The testing conditions are changed by setting the frequency to 52 Hz. As a result of the frequency estimation (b-SDFT), the frequency experiences oscillations around 52 Hz, with the greatest deviation of 0.5 Hz. However, with the e-SDFT algorithm, the maximum error is suppressed to 0.03 Hz (Fig. 17). Due to the recursive DFT approach, numerical oscillations in the magnitude estimation based on b-SDFT can be seen. The values vary significantly from the reference value of 100 Vrms (Fig. 18). By reducing the sampling rate to 1 kHz, it was ensured that CPU can handle non-recursive DFT calculations. In this way, the numerical oscillations are avoided and accurate magnitude estimates are obtained. The existing error is reduced from 8 Vrms to 0.3 Vrms. As a result of the increased input signal frequency, the estimated phasors are rotating in the complex plane with the frequency equal to the deviation from the nominal value (Fig. 19).

The results of the phase estimation are shown in Fig. 20. Since the frequency is higher than the nominal, the phase estimation has a positive slope. The slight improvement is obtained by running the enhanced version of SDFT.

C. The results of the synchrophasor estimation algorithm under the dynamic condition of the frequency ramp from 48Hz to 52Hz

Fig. 21 and Fig. 22 show the results of the frequency estimation when the input signal frequency linearly varies from 48 Hz to 52 Hz for a duration of 4 s, for b-SDFT and e-SDFT, respectively. The improved accuracy of the e-SDFT is noticeable for all three parts of the signal at different frequencies: nominal start frequency, the frequency ramp and off-nominal stop frequency. The estimated magnitude during the frequency ramp is presented in Fig. 23. Clearly, the estimation accuracy is deteriorated during the transition period of 4 s. The deviations from nominal value vary from 0.05 to 0.5 Vrms.

D. The results of the synchrophasor estimation algorithm under the dynamic conditions-modulated input signal magnitude and phase

In order to check the performance of the algorithm when the input signal has a modulated magnitude, the magnitude is adjusted to vary with the modulation index of +/- 10 % at a frequency of 2 Hz. In Fig. 24, it is shown that during the period of 1 s, the estimated magnitude value varies from 90 to 110 Vrms.

Fig. 25 indicates the presence of sinusoidal oscillations around the referent value of 50 Hz. The deviations are small with a maximum value of only 0.005 Hz.

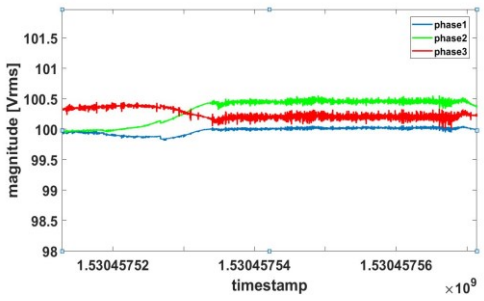

Figure 13. Magnitude estimation (Test A)

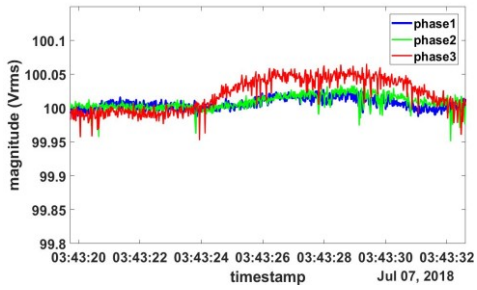

Figure 14. Magnitude estimation with filter (Test A)

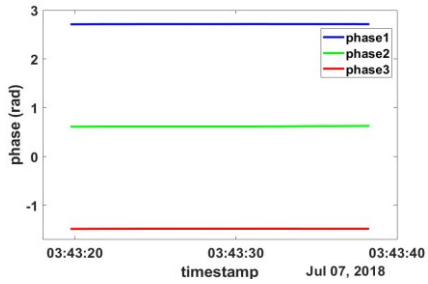

Figure 15. Phase estimation (Test A)

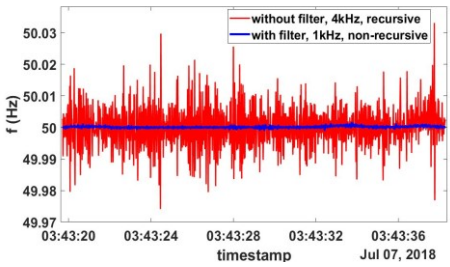

Figure 16. Frequency estimation (Test A)

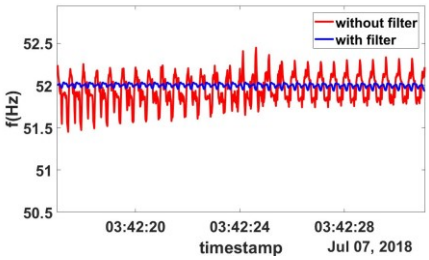

Figure 17. Frequency estimation (Test B)

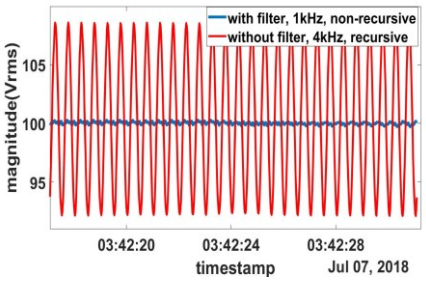

Figure 18. Magnitude estimation (Test B)

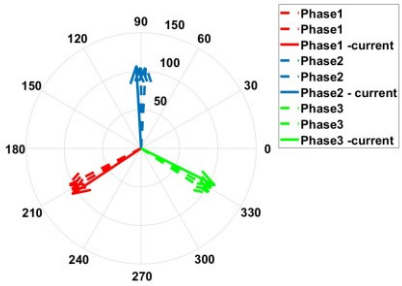

Figure 19. Phase rotation (Test B)

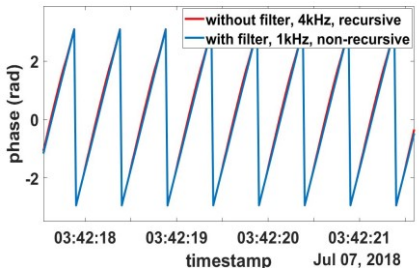

Figure 20. Phase estimation (Test B)

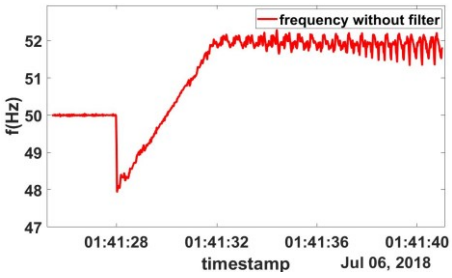

Figure 21. Frequency estimation (Test C)

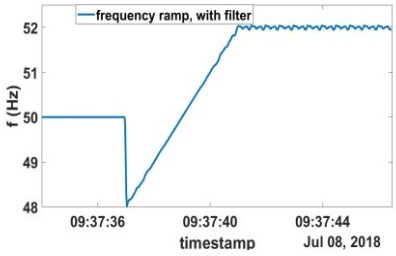

Figure 22. Frequency estimation, with filter (Test C)

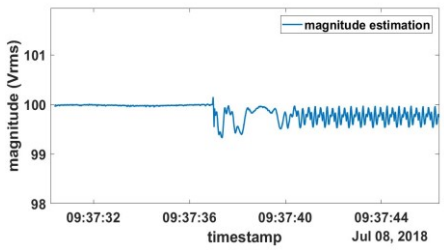

Figure 23. Magnitude estimation (Test C)

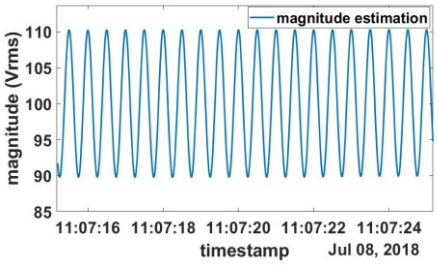

Figure 24. Magnitude estimation (Test D)

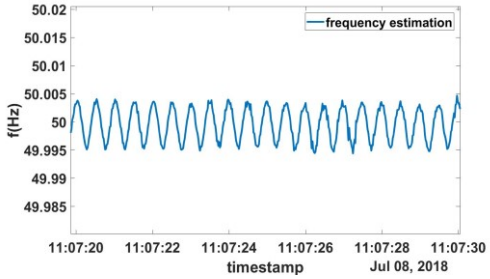

Figure 25. Frequency estimation (Test D)

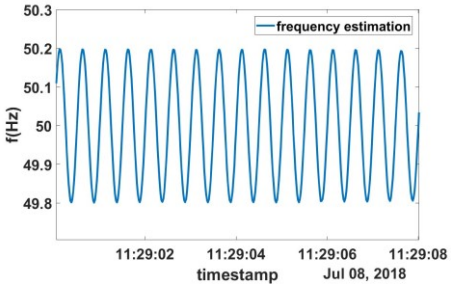

Figure 26. Frequency estimation (Test D, phase modulation)

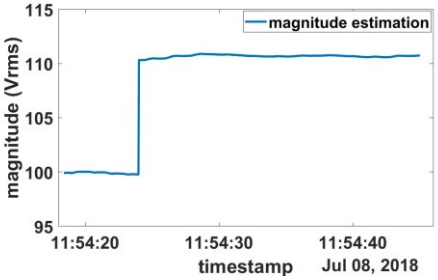

Figure 27. Magnitude estimation (Test E)

Similarly as for the magnitude modulation, in the frequency estimation during the phase modulation (+/- 10 %), oscillations occur around the referent value of 50 Hz. However, this time, oscillations are more significant, reaching a maximum value of 0.2 Hz (Fig. 26).

E. The results of the synchrophasor estimation algorithm under the step in the magnitude of the input signal

The final test represents the step in the magnitude of the input signal. The test starts with applying the magnitude step of 10% and tracking the algorithm response. Fig. 27 shows the result of estimation and the maximum error occurs at around 11:54:28, when the magnitude error reaches 0.9 Vrms.

V. CONCLUSIONS

This paper presents the optimization and implementation of the SDFT algorithm for the synchrophasor estimation on the cost-efficient processing platform.

The utilization of the mean filter and increased number of non-recursive DFTs in the implementation of the enhanced SDFT (e-SDFT) result in significantly higher accuracy of the results compared to the basic SDFT (b-SDFT).

In the future work, besides the processing platform for the purpose of voltage sampling, the developed code will be implemented in the processing platform for the current sampling. Furthermore, the complete testing of the prototype performance according to IEEE Standard C37.118.1a-2014 will be performed with PMU Utility Tool in RTDS and further algorithm improvements will be suggested.

Additionally, other higher performance programming languages and/or parallel processing will be considered and the code will be further optimized.

ACKNOWLEDGMENT

This study was financially supported by the Dutch Scientific Council NWO-STW, under the project 408-13-025 within the program of Uncertainty Reduction of Smart Energy Systems (URSES)

REFERENCES

[1] Teach, Learn, and Make with Raspberry Pi: http://www.raspberrypi.org/.

[2] D. Hart, D. Novosel, Yi Hu, B. Smith and M. Egolf, "A new frequency tracking and phasor estimation algorithm for generator protection," in *IEEE Transactions on Power Delivery*, vol. 12, no. 3, pp. 1064-1073, July 1997.

[3] I. Carugati, C. Orallo, P. Donato, S. Maestri and D. Carrica, "Three-phase harmonics measurement method based on mSDFT," in IEEE Latin America Transactions, vol. 12, no. 7, pp. 1250-1257, Oct. 2014.

[4] P. Romano and M. Paolone, "Enhanced Interpolated-DFT for Synchrophasor Estimation in FPGAs: Theory, Implementation, and Validation of a PMU Prototype," in IEEE Transactions on Instrumentation and Measurement, vol. 63, no. 12, pp. 2824-2836, Dec. 2014.

[5] Jun-Zhe Yang and Chih-Wen Liu, "A precise calculation of power system frequency," in IEEE Transactions on Power Delivery, vol. 16, no. 3, pp. 361-366, July 2001.

[6] D. R. Gurusinghe, D. Ouellette, A. D. Rajapakse, Implementation of Smart DFT-based PMU Model in the Real-Time Digital Simulator, in the Proceedings of the International Conference on Power Systems Transients, Seoul, Republic of Korea June 26-29, 2017.

[7] IEEE Standard for Synchrophasor Measurements for Power Systems, IEEE Std., C37.118.1-2011, December 2011.

[8] I. Radević: SDFT Based PMU Prototype, MSc thesis, July 2018, available on: https://repository.tudelft.nl

[9] S. Šandi, B. Krstajić and T. Popović, "pyPMU - Open source python package for synchrophasor data transfer," 24th Telecommunications Forum (TELFOR), Belgrade, 2016, pp. 1-4.

[10] https://github.com/GridProtectionAlliance/PMUConnectionTester

[11] Solutions for intelligent distribution grids, Horizon 2020 project, European Commission, 2017-2021.

[12] https://www.smartstatetechnology.nl/

[13] http://www.orangepi.org/

[14] P. Hintjens: ZeroMQ - Messaging for Many Application, O'Reilly Media, March, 2013.

[15] https://www.rtds.com/about/downloads/