PTIDES Model on a Distributed Testbed Emulating Smart Grid Real-Time Applications

Slobodan Matic, Ilge Akkaya, Michael Zimmer, John C. Eidson and Edward A. Lee

***Abstract*—PTIDES, a programming model for distributed real-time systems, was proposed previously. The model captures both the functionality of the system and the desired timing of interactions with the environment. The PTIDES simulator supports simulation of both of these aspects. In this work, we focus on the PTIDES development environment in the context of applications drawn from the control of electric power systems. The evaluation is based on experiments on a system of distributed computing platforms emulating typical power system control and monitoring devices and an emulation of portions of the electric power grid based on conventional micro-controller instrumentation.**

I. Introduction

MOST real-time software is structured either as threads with priorities or as tasks with periods or deadlines. Zhao et al. proposed an alternative programming model called PTIDES [13] (Programming Temporally Integrated Distributed Embedded Systems) that structures real-time software as an interconnection of actors communicating with timestamped events. PTIDES is an extension to the Ptolemy II simulation environment [9] used as a coordination language for the design of distributed real-time embedded systems. PTIDES leverages network time synchronization [8], [4] to provide a coherent global temporal semantics in distributed systems. Zou et al. gave an execution strategy for PTIDES and introduced feasibility analysis in [14]. Eidson et al. further described Ptides in [5], which showed how PTIDES supports modal behaviors and described an application to power plant control.

To design a real-time embedded system using PTIDES, the designer first constructs a model of the system, and perhaps the controlled plant, using PTIDES actors in the Ptolemy II simulation environment. Once the designer is satisfied that the simulated design meets the application's temporal and functional requirements, the designer generates code for the target platform using the code generation facility of the Ptolemy II system. The next step is a schedulability and feasibility analysis to determine whether an execution schedule exists on the target platform that will preserve the run-time temporal specifications of the original PTIDES model.

The PTIDES environment has been used to model and design a number of industrial-based applications. In [6] we used power plant emergency detection and power supply shutdown as PTIDES application examples. The purpose of this work is to validate the PTIDES design environment for smart grid applications executing distributed platforms. Our goal is to deploy some of the synchrophasor-based real-time applications on conventional micro-controller hardware connected over a local area network. We believe that reliable smart grid asks for fault-tolerance at all networked devices including the low-level embedded ones. Note that software fault-tolerance, e.g. redundancy, requires deterministic programming models such as PTIDES.

II. Target Power Grid System

The validation of PTIDES is based on the simplified version of a power system as illustrated in Fig. 1. This system implements a version of the measurement and control devices and algorithms used by the electric power industry in monitoring and controlling the transmission grid. The experimental system is based on papers [11], [12]. In particular, this design is suggested for system integrity protection schemes that detect power swings and out-of-step conditions.

The transmission grid is emulated using conventional electronic instrumentation. This emulation provides pairs of analog signals in the range 0-5V representing the outputs of voltage and current monitors typically found in substations or at critical points in the transmission system. The magnitudes and phases (essentially the power factor) of each pair can be varied independently, as can the frequency. All frequencies and phases are relative to a single real-time clock which is synchronized to the real-time clocks in the monitoring and control devices. This allows us to emulate several generators, each with different load conditions and frequency, and to permit the generation of synchrophasor data. In addition, digital signals can be generated or accepted to model warning signals from equipment or breaker closure commands and the like.

The PMU and SVP platforms represent primary measurement units and synchrophasor vector processing units used to monitor and control the grid. Each unit executes at least two different algorithms requiring real-time deadlines to be met as well as network interaction with peer or supervisory units. Within each PMU quantities such as the power factor and load are monitored to detect trip points at which a signal is sent

The authors are with the Electrical Engineering and Computer Sciences Department, University of California, Berkeley. email: {matic, ilgea, mzimmer, eidson, eal}@eecs.berkeley.edu.

This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #0720882 (CSR-EHS: PRET), #1035672 (CPS: PTIDES) and #0931843 (ActionWebs)), the U. S. Army Research Office (ARO #W911NF-07-2-0019), the U. S. Air Force Office of Scientific Research (MURI #FA9550-06-0312), the Air Force Research Lab (AFRL), the Multiscale Systems Center (MySyC), one of six research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program, and the following companies: Bosch, National Instruments, Thales, and Toyota.

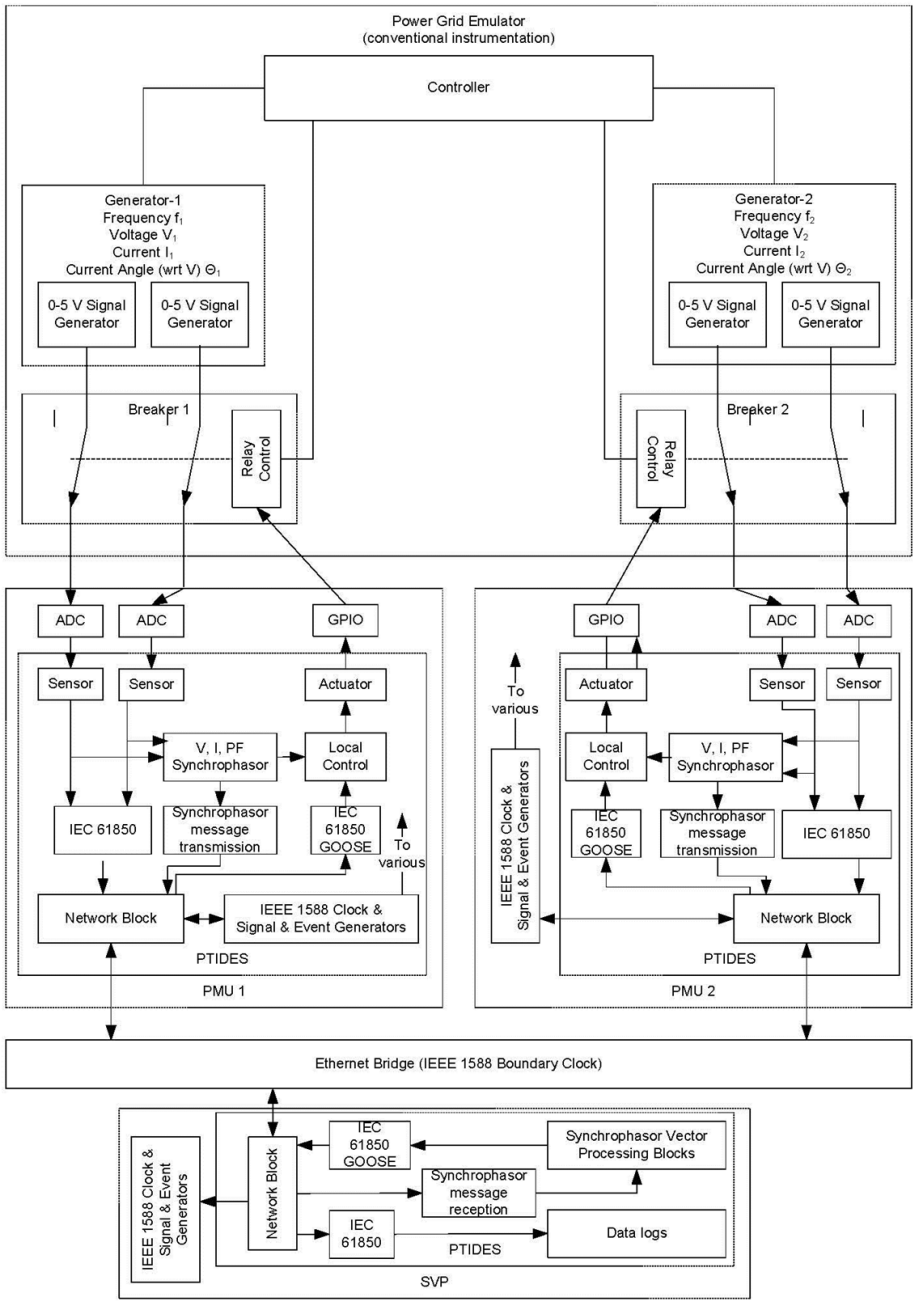

Fig. 1. Proposed validation system

to open the local breaker. Within the SVP the synchrophasors from the several PMUs are compared to detect discrepancies in the frequencies and phase and generate breaker signals to be sent to the appropriate PMU. The raw data is recorded in logs. Each unit contains a real-time clock synchronized to its peers using the recently standardized time synchronization protocol IEEE 1588.

Generation of synchrophasor data in each PMU is based on sampling the voltage and current signal inputs to the PMU and processing these relative to the local real-time clock. The raw data is forwarded to the SVP using the IEC 61850 process bus. The synchrophasor data is forwarded by the synchrophasor message transmission block. Today in practice this would use the C37.118 synchrophasor protocol since

standard committees are in the process of mapping this onto IEC 61850. Control messages use the IEC 61850 GOOSE protocol blocks.

III. PTIDES Basics

PTIDES is based on the semantics of discrete-event (DE) systems [1], [2], which provides a model of time and concurrency. PTIDES actors are concurrent components that exchange time-stamped events via input and output ports. The event time stamps are a formal part of the model referred to as model time. Model time may or may not bear any relationship to time in the physical world, i.e., real time. In typical DE semantics, each actor processes input events in time-stamp order but without constraints on the real time at which events are processed. PTIDES extends DE by establishing the correspondence between model time and real time at sensors, actuators, and network interfaces.

PTIDES assumes that each local platform contains a real-time clock synchronized with similar clocks in the other platforms. These clocks are used to ensure global time stamp order of execution and facilitate local scheduling without recourse to message passing between platforms. Note that with the exception of actual testing in a distributed environment, most features of PTIDES have been designed and simulated on numerous distributed applications. Likewise, generated code has been tested and timing verified in *single* platform situations.

The specific features of the PTIDES implementation that are validated in this project are as follows:

1) Safe-to-process semantics. DE semantics requires that events be presented to actors with state in global time stamp order. In an execution (as opposed to simulation) environment, each port of a multiport actor is assigned a safe-to-process attribute δ indicating that an event with time stamp τ appearing on one port will not be executed before real time $\tau + \delta$ which guarantees that no event with a time stamp earlier than τ will appear at another port of the same actor. This is enforced locally in each platform by observing the real-time clock on the local platform, but the attribute itself involves computation of maximum delays along all causal paths between the actor ports and the source of an event, usually a sensor.
2) Time stamp semantics at sensors, actuators, and network interfaces. We need to ensure that PTIDES temporal semantics are rich enough to handle the complexity of real sensor, actuator and network interfaces.
3) Timing support. We need to evaluate conditions for which support for enforcing correspondence between model and real-time require hardware support external to the microprocessor environment, e.g. support based on the National Semiconductor DP83640 IEEE 1588 enabled PHY chip.

An example of a PTIDES model is shown in Fig. 2. The model is rendered visually in the Ptolemy II environment, which allows for hierarchical composition of component models as depicted by dashed lines in the figure. This model represents a distributed application that runs on three processors PMU1, PMU2 and SVP. The power grid and network are modeled by components Power Grid and Network Model respectively.

The model illustrates a system that monitors relative phases of two power signals and outputs a control signal whenever the phase difference exceeds π/3 radians. Power grid signals are sampled at 500 samples/second. Processors PMU1 and PMU2 deliver phase values to SVP every 10 ms. The first power signal has a constant phase of π/18 radians. The second signal has the same phase initially, but at t=0.5 s, a phase disturbance starts occuring. The upper right corner of Fig. 2 shows the plots of input and output signals.

The lower left part of the figure is a PTIDES model for the code that executes on PMU2 (and PMU1). This illustrates the dataflow implementation of a phase-measurement unit. The input signal is multiplied by real and imaginary parts of a complex sinusoid at the same frequency and then filtered. The finite impulse response (FIR) filter coefficients for this implementation are designed for a sampling rate of 500 samples/second and a cut-off frequency of 30 Hz. The infinite impulse response (IIR) filter actors are used in the design to allow IIR implementations of the same functionality. An arctangent actor is used to calculate the angle between the two filter outputs, which represents the relative phase of the input signal with respect to the reference complex sinusoid.

The component model in the lower right part of the figure illustrates the code that executes on SVP, i.e., the code that calculates the phase difference between the phase values received from the two platforms. Whenever phase difference exceeds π/3 radians, the control output is set to high. The multiple mode operation is implemented through Ptolemy II multi-modal modeling features.

IV. Validation System Implementation

A simplified version of the target power grid application described in Sec. II has been built and deployed. The emulator to generate the needed application input signals is based on the National Instruments LabVIEW hardware platform. This platform contains reconfigurable I/O chip that enables custom timing, inline processing and control. The voltage accuracy of each 0-5V signal is better than 5mV, and the frequency is adjustable from 59.5 to 60.5 to an accuracy of 0.001Hz. The PMU and SVP units use the Renesas SH7216 Demonstration Kit as the execution platform. The embedded micro-controller runs at up to 200MHz clock, and has 1MB of Flash, 128MB of RAM and an integrated floating-point unit. This platform also uses the National Semiconductor DP83640 PHY as the Ethernet controller. Beside enabling precise time synchronization between processors, the DP83640 allows the evaluation of external timing support in enforcing correspondence between model time and real time. Note that in the first implementation of this project all message marshaling and demarshaling are based on simple data structures and UDP rather than the actual power industry protocols. In the future if we want to interact with actual commercial devices, these blocks will be converted to adhere to the relevant protocols.

This validation system allows us to evaluate supporting, in real-time, network and clock synchronization protocols in

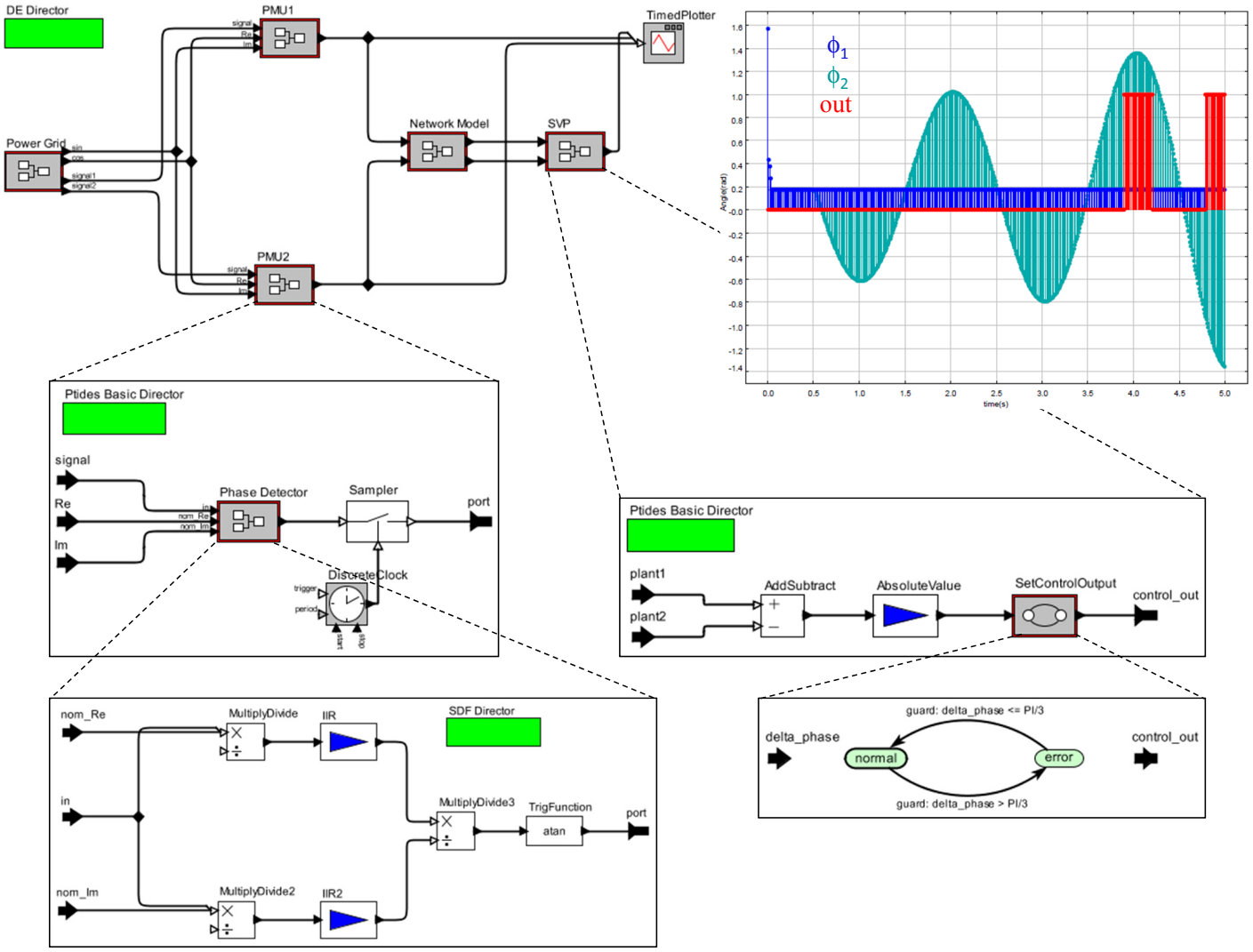

Fig. 2. Simplified PTIDES model of the target smart grid application. Hierarchical composition of model composite actors. Simulation plot (upper right corner).

parallel with the application code in a PTIDES generated environment. For the purposes of the validation we focus on the distributed code that monitors phase difference between two grid protection points.

A. Time Synchronization Protocol Implementation

The IEEE 1588-2008 [4] standard defines a Precision Time Protocol (PTP) over a network to synchronize distributed real-time clocks to the sub-microsecond range. The protocol was developed to meet requirements not achievable by existing common methods of clock synchronization, where the millisecond accuracy of Network Time Protocol (NTP) is not accurate enough and a Global Positioning System (GPS) receiver at each node is not feasible due to cost or signal availability.

The protocol operates on a hierarchical master-slave structure where all clocks synchronize their time to the grandmaster clock at the top of the hierarchy. In a basic network configuration, all clocks in the network exchange information about their accuracy and stability, and the best master clock (BMC) algorithm dynamically selects the master clock and slave clocks. Synchronization is primarily achieved by slave clocks adjusting their frequency based on the calculated time offset from the master clock. The offset is calculated by exchanging timestamped messages over the network.

The implementation of the protocol developed for this application can be viewed as an actor with discrete event semantics, allowing easier integration with PTIDES. Code is only executed when provided with an input event and it executes to completion without blocking. A modified version of the open-source network protocol stack μIP [3] is used to provide UDP support for both PTP and data communication. A controller is used to adjust the frequency of the slave clock to maintain a zero offset between the master and slave clock. One proportional-integral (PI) controller is used to adjust the frequency to match that of the master clock, and another PI controller is used to temporarily adjust the frequency to maintain a zero offset.

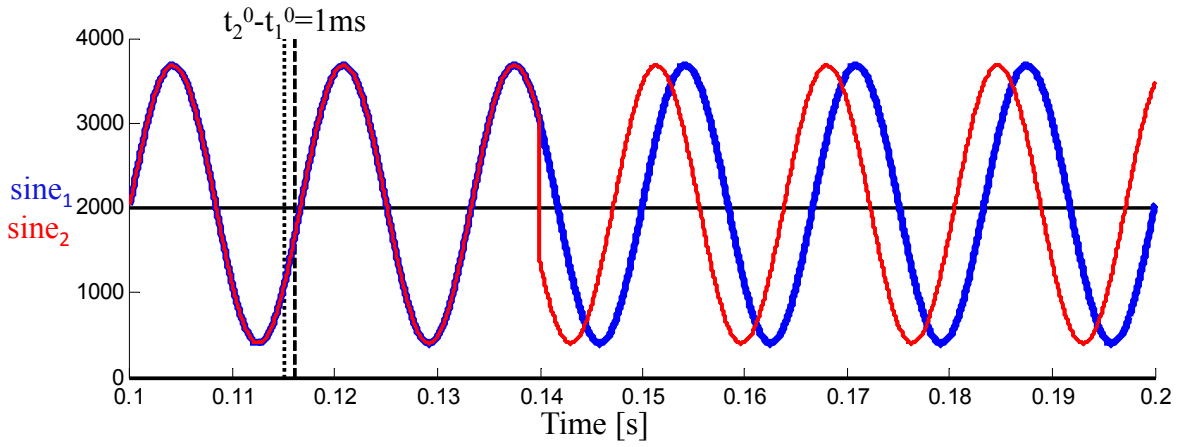

Fig. 3. Experiment 1. Emulated signals at two grid protection points. Two vertical lines represent zero time instants at two PMUs

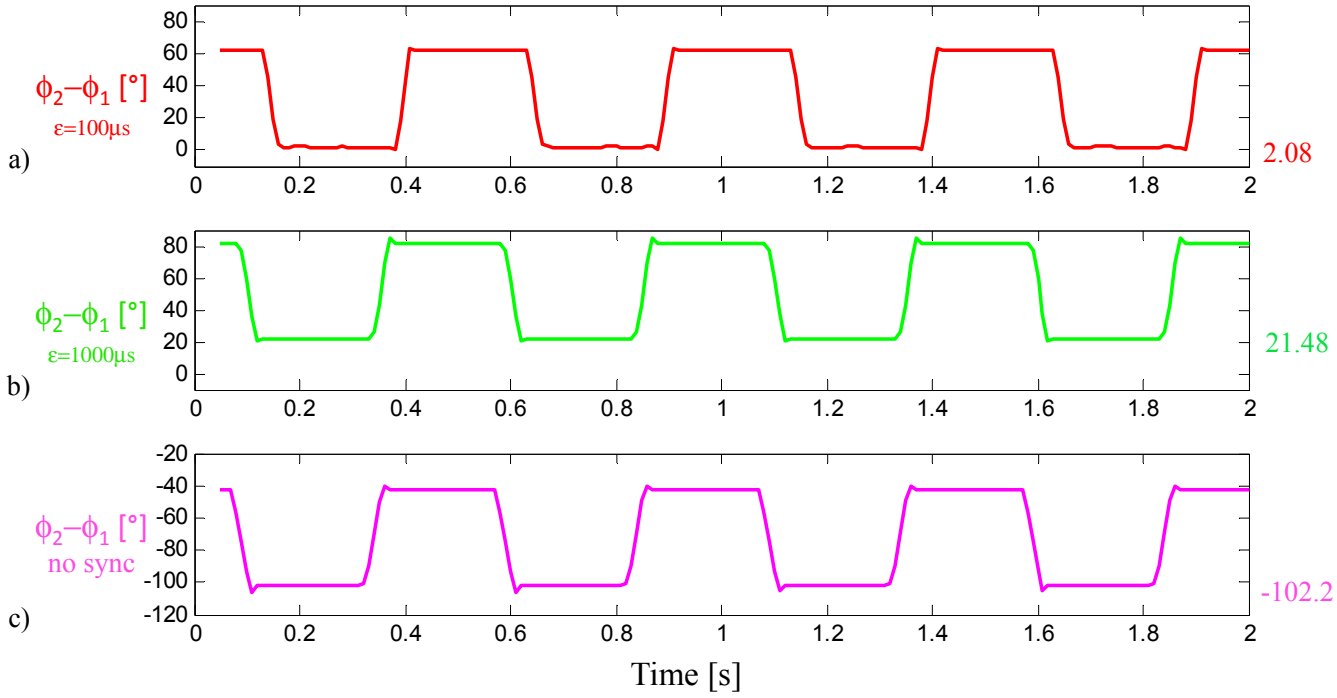

Fig. 4. Experiment 1. Effects of synchronization on phase difference detection

The Renesas 7216 Demonstration Kit utilizes a DP83640 Precision PHYTER from National Semiconductor. This PHYTER provides IEEE 1588 hardware support with a frequency adjustable real-time clock, packet detection and timestamping with $8ns$ precision, and event timestamping and triggering through GPIO ports. Hardware support allows for much better accuracy and precision, and reduces software execution time required for running the protocol. When deployed on a three platform system with one master clock and two slave clocks, the slave clocks were able to synchronize to within $100ns$. Variable packet delays through a typical router reduce synchronization accuracy, so using a router with IEEE 1588 support would allow for even better synchronization accuracy.

B. Phase Filter Implementation

The distributed PMU implementation in the emulated grid focuses on extracting precise-time phasor information from power signals. The emulated $60Hz$ power grid signals, sampled with a period of $2ms$, are converted into phasor form by quadrature mixing with a reference quadrature oscillator at the same sampling period. This step is followed by low-pass filtering to obtain the real and imaginary components of the grid signal (V,I). These components are then used to determine the phase with respect to the reference oscillators.

In the filtering stage, both FIR and Butterworth (IIR) implementations have been explored, both of which are efficient in terms of run-time and frequency response characteristics. For a low-ripple and accurate implementation, at the operating

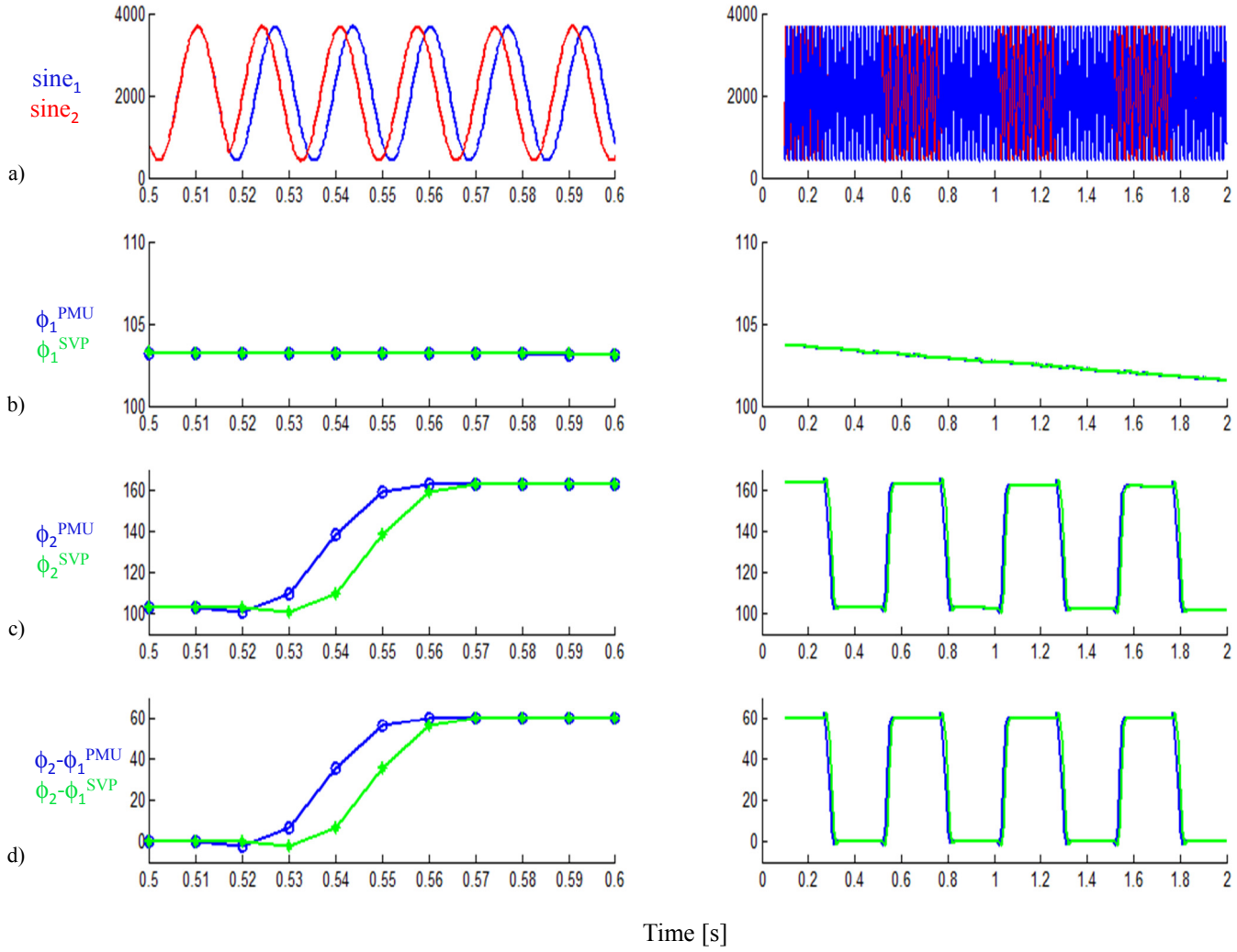

Fig. 5. Experiment 2. Phase change detection latency

sampling period, a 20-tap FIR or a third order IIR filter is sufficient. On the described validation platform these filters take less than 10μs of the processor time per sample.

Filtering is carried out at a sampling period of 2 ms, however the reporting rate varies. Typically, PMUs report the phase values at 10-60 frames/second to the Synchrophasor Vector Processor (SVP), which is the processing unit that receives phase values from the distributed PMU system and calculates phase gradients. Once the relative phasors are calculated in the distributed PMU system, SVP receives the individual phase values and calculates the phase difference between two platforms. Phase difference can be used to calculate other parameters such as slip frequency and acceleration, which are fundamental parameters for determining the power system operating state [11].

V. Experimental Evaluation

A. Experiment 1: Time synchronization

In the first performed experiment we demonstrate the need for time synchronization between the processors in detection of voltage phase difference between two grid protection points. Each of the two PMU processors samples a voltage input signal with period of $T_s = 200\mu s$, computes and stores its phase with respect to a local time reference. In an offline analysis, samples from the two processors that correspond to $T_c = 10ms$ time instants are used to compute phase difference. SVP processor is not used and there is no networking involved (besides PTP). Note that the 12-bit analog-to-digital converters are previously calibrated so that the same signal input value results in the same readings on the two processors. In the experiment, the voltage input signal on one of the processors has a constant phase. On the other processor every quarter of a second the input signal phase is alternatively increased or decreased for 60°. If the processors are perfectly synchronized, the phase difference will alternate between 0 and 60°.

The time synchronization error ϵ is emulated by running the PTP protocol and setting the initial zero time instants with offset ϵ on two processors. Fig. 3 shows an example for $\epsilon = 1ms$ with two vertical lines representing zero time instants at two PMUs. The same figure shows the moment when the phase of signal $sine_2$ at the input of processor PMU2 is increased for 60°. The experiment is repeated for

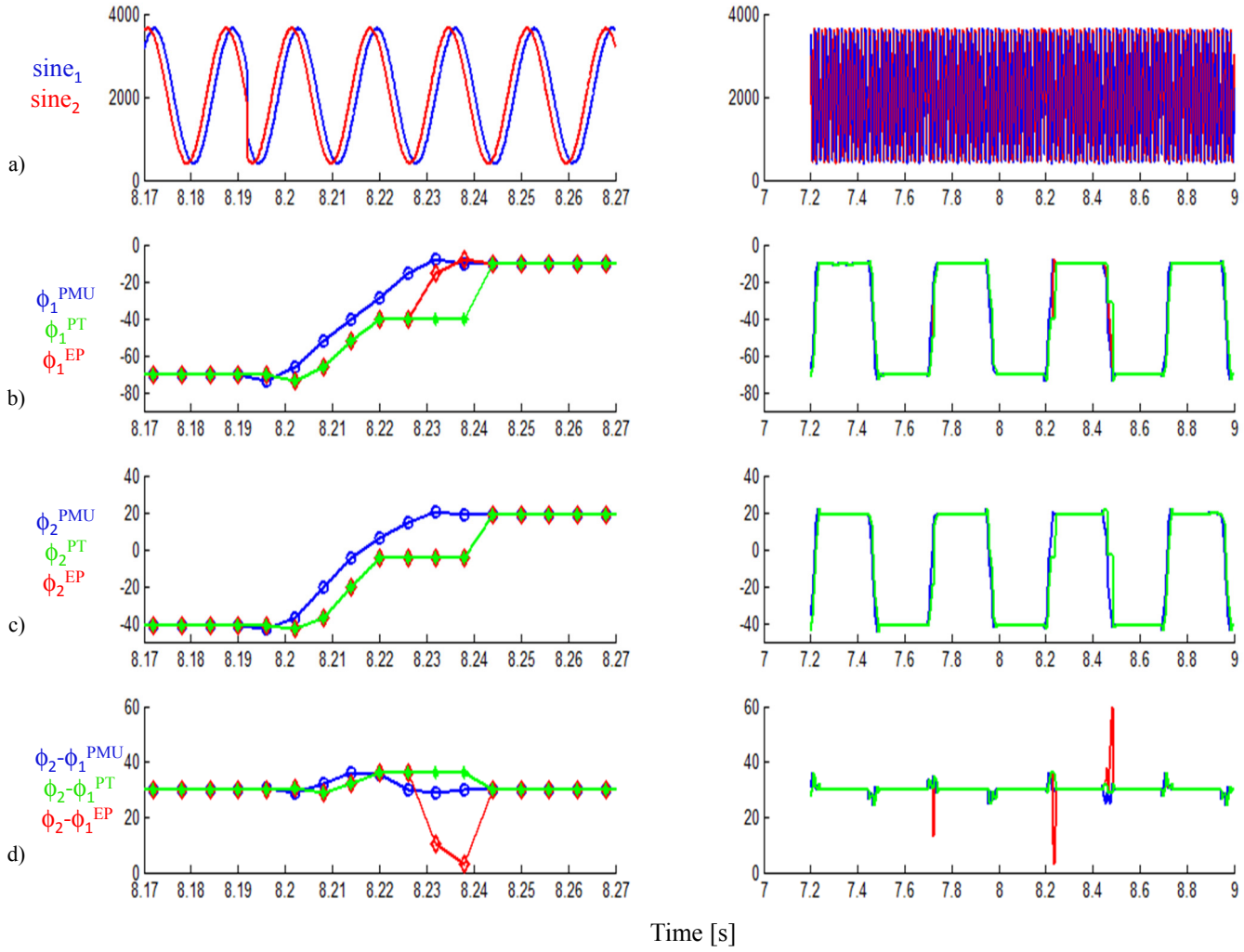

Fig. 6. Experiment 3. Phase change: PTIDES vs. traditional event programming

different values of the synchronization error ϵ and the phase difference plots are shown in Fig. 4. The numbers shown to the right of the plots denote the average estimated phase difference when the two applied signals are exactly in phase, i.e., when the actual phase difference is zero. For instance, as shown in Fig. 4b), when $\epsilon = 1ms$, the average estimated phase difference is 21.48°, whereas it should be 0. This value is close to the theoretical value of $360° \cdot f \cdot \epsilon = 360° \cdot 60Hz \cdot 10^{-3}s = 21.6°$, where f represents the frequency of the input signal. Fig. 4c) shows phase difference during the same experiment but when the synchronization protocol is not running. In the intervals when the two input signals are in phase the estimated phase difference of -102.2° is clearly an error. Overall, the experiment shows that time synchronization is necessary with required synchronization error in the sub-microsecond range.

B. Experiment 2: Latency

This experiment serves as a performance test for the PTIDES based code. In particular, it estimates latency of phase difference computation. The experiment is similar to the Experiment 1 in that each PMU processor samples a voltage input signal and computes its phase with the period $T_s = 200\mu s$. However, the two computed phases are transmitted to the SVP processor with the period $T_c = 10ms$, where the phase difference is computed. The network consists of the three processor boards directly connected through a standard router. An open source implementation of a UDP stack is used for phase data communication, and the PTP implementation described in the previous section was used for time synchronization. The total latency, including the protocol stack latency, of an individual packet transmission over such a network is measured to be in the range 1-5ms. This timing variability mostly comes from packet processing in the router and reception protocol stack. No packet loss has been noticed at the value of phase period communication of $T_c = 10ms$.

Similar to the Experiment 1, the phase of the voltage input signal on PMU1 is kept constant, whereas the phase of the input signal on PMU2 is increased or decreased for 60° every 250ms (Fig. 5a). Fig. 5 shows time diagrams from a typical run of the experiment, with the plots on the left side zoomed in to show system behavior around the phase transition time instant. On Fig. 5b), note that the detected phase PMU1 is

not constant, but has a small constant phase decline. This is explained by the fact that although real-time clocks in the three processors are synchronized using PTP protocol, the voltage input sinewaves are generated on the grid emulator that has its own independent clock. The plots on Fig. 5d) show that the stepwise change in the phase of one of the input signals is detected on the SVP processor with the maximum total latency of $50ms$ (green line). As shown with the blue line, the phase filtering algorithm described in the previous section introduces itself the latency of about $20ms$.

C. Experiment 3: PTIDES vs. traditional event programming

In the third experiment we show a benefit of the code generated using the PTIDES programming model over the code written in a traditional event programming style. The experiment is similar to the Experiment 2 in that each PMU processor samples a voltage input signal and computes its phase with the period $T_s = 200\mu s$. However, the two computed phases are sent to the SVP processor with a period $T_c = 6ms$. In this experiment the two voltage input signals always have phase difference of 30°, but every $250ms$ both phases are alternatively increased or decreased for 60° (Fig. 6a).

In the traditional event programming style each packet with the phase data is processed on the SVP processor as soon as it is received. Thus, the phase difference is computed as soon as a new phase data for one of the signals is received. In Fig. 6 the outcome of such code execution is shown in red color. As the network introduces different delays for different packets, and might even reorder the packets, the computed phase difference might not correspond to the actual phase difference between two signals. In addition, with the communication period of $T_c = 6ms$ up to 4% of packets might be lost due to the protocol stack overload. This results in more frequent phase difference errors. On the other hand, in the PTIDES programming approach the data timestamp is also communicated together with the phase data value. So, on the receiving SVP platform only the corresponding pairs of packets are matched and the difference between the corresponding phase data values is computed. In Fig. 6 the outcome of PTIDES code execution is shown in green color. As shown in Fig. 6d) the phase difference error for the PTIDES approach remains less than 5° which is about the same value as for the phase difference computation without communication (blue line). On the other hand, the error in the traditional approach can be as large as 30°. Note that in more complex applications such glitches may result in serious faults.

VI. Conclusion and Future Work

In this paper we applied PTIDES programming methodology in the context of smart grid real-time applications. Our evaluation of the methodology was based on experiments performed on a system of distributed micro-controllers time-synchronized over a local network. On an example application that computes phase difference between two grid points we first demonstrated the importance of time synchronization. For the application code generated using the PTIDES methodology we estimated latency in phase difference estimation and showed that the methodology results in better estimates compared to those obtained using a traditional programming methodology.

For our future work, we see the potential of the same setup being used for more complicated distributed applications such as those suggested for networked cooperative control in power systems. In such applications functional and timing correctness that PTIDES offers might be critical. Also, our efforts will be directed to automatic code generation from high-level Ptolemy II models such as the one discussed in Sec. III.

Acknowledgment

The authors would like to acknowledge the contribution of Casidhe Lee who helped us implement emulation of power grid signals. Veselin Skendzic of Schweitzer Engineering Laboratories introduced us to the synchrophasors and related power systems problems. National Instruments donated emulator equipment and their Berkeley office provided support. Renesas donated micro-controllers on which the validation system was deployed together with the corresponding programming environment.

References

[1] F. Baccelli, G. Cohen, G. J. Olster, and J. P. Quadrat. *Synchronization and Linearity*, An Algebra for Discrete Event Systems. Wiley, New York, 1992.

[2] C. G. Cassandras. *Discrete Event Systems*, Modeling and Performance Analysis. Irwin, 1993.

[3] A. Dunkels. *uIP - a TCP/IP stack for 8- and 16-bit microcontrollers*. http://dunkels.com/adam/uip/. 2002-10-14.

[4] J. C. Eidson. *Measurement, Control, and Communication Using IEEE 1588*. Springer, 2006.

[5] J. Eidson, E. A. Lee, S. Matic, S. A. Seshia, and J. Zou. *A Time-centric Model for Cyber-Physical Applications*. In Workshop on Model Based Architecting and Construction of Embedded Systems (ACES-MB), pages 2135, 2010.

[6] J. C. Eidson, E. A. Lee, S. Matic, S. A. Seshia, and J. Zou. *Time-centric Models for Designing Embedded Cyber-Physical Systems*. http://www.eecs.berkeley.edu/pubs/techrpts/2009/eecs-2009-135.pdf. Technical Report UCB/EECS-2009-135, EECS Department, University of California, Berkeley, October 9 2009.

[7] J. Eker, J.W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. *Taming Heterogeneitythe Ptolemy Approach*. Proceedings of the IEEE, 91(2):127144, 2003.

[8] S. Johannessen. *Time Synchronization in a Local Area Network*. IEEE Control Systems Magazine, pages 6169, 2004.

[9] E. Lee, S. Neuendorffer, and M. Wirthlin. *Actor-oriented design of embedded hardware and software systems*. Journal of Circuits, Systems, and Computers, 12(3):231260, 2003.

[10] J. Hill, R. Szewcyk, A. Woo, D. Culler, S. Hollar, and K. Pister. *System architecture directions for networked sensors*. In 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pages 93104, 2000.

[11] E. O. Schweitzer, D. Whitehead, A. Guzman, Y. Gong, and M. Donolo. *Advanced real-time synchrophasor applications*. http://www.selinc.com/workarea/downloadasset.aspx?id=3537. Technical report, Schweitzer Engineering Laboratories, Inc, 2008.

[12] V. Skendzic, I. Ender, and G. Zweigle. *IEC 61850-9-2 process bus and its impact on power system protection and control reliability*. https://www.selinc.com/workarea/downloadasset.aspx?id=3479. Technical report, Schweitzer Engineering Laboratories, Inc, 2007.

[13] Y. Zhao, E. A. Lee, and J. Liu. *A programming model for time-synchronized distributed real-time systems*. In Real-Time and Embedded Technology and Applications Symposium (RTAS), Bellevue, WA, USA, 2007. IEEE.

[14] J. Zou, S. Matic, E. A. Lee, T. H. Feng, and P. Derler. *Execution Strategies for Ptides, a Programming Model for Distributed Embedded Systems*. In Real-Time and Embedded Technology and Applications Symposium (RTAS), San Francisco, CA, 2009. IEEE.