



Missouri University of Science and Technology
Scholars' Mine

Computer Science Faculty Research & Creative Works

Computer Science

01 Aug 2008

An Open Framework for Highly Concurrent Real-Time Hardware-in-the-Loop Simulation

Ryan C. Underwood


Bruce M. McMillin

Missouri University of Science and Technology, ff@mst.edu

Mariesa Crow

Missouri University of Science and Technology, crow@mst.edu

Follow this and additional works at: https://scholarsmine.mst.edu/comsci_facwork

 Part of the [Computer Sciences Commons](#), and the [Electrical and Computer Engineering Commons](#)

Recommended Citation

R. C. Underwood et al., "An Open Framework for Highly Concurrent Real-Time Hardware-in-the-Loop Simulation," *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference (2008, Turku)*, pp. 44-51, Institute of Electrical and Electronics Engineers (IEEE), Aug 2008. The definitive version is available at <https://doi.org/10.1109/COMPSAC.2008.165>

This Article - Conference proceedings is brought to you for free and open access by Scholars' Mine. It has been accepted for inclusion in Computer Science Faculty Research & Creative Works by an authorized administrator of Scholars' Mine. This work is protected by U. S. Copyright Law. Unauthorized use including reproduction for redistribution requires the permission of the copyright holder. For more information, please contact scholarsmine@mst.edu.

An Open Framework for Highly Concurrent Real-Time Hardware-In-the-Loop Simulation

Ryan C. Underwood * Bruce M. McMillin *
{rcuca4@mst.edu, ff@mst.edu}

Department of Computer Science

Intelligent Systems Center

Missouri University of Science & Technology, Rolla, MO 65409

M. L. Crow *

{crow@mst.edu}

Electrical and Computer Engineering

Abstract

Hardware-in-the-loop (HIL) real-time simulation is becoming a significant tool in prototyping complex, highly available systems. The HIL approach permits testing of hardware prototypes of components that would be extremely costly or difficult to test in the deployed environment. In power system simulation, key issues are the ability to wrap the systems of equations (such as Partial Differential Equations) describing the deployed environment into real-time software models, provide low synchronization overhead between the hardware and software, and reduce reliance on proprietary platforms. This paper introduces an open source HIL simulation framework that can be ported to any standard Unix-like system on any shared-memory multiprocessor computer, requires minimal operating system scheduler controls, enables an asynchronous user interface, and allows for an arbitrary number of secondary control components. The framework is implemented in a soft real-time HIL simulation of a power transmission network with physical Flexible AC Transmission System (FACTS) devices. Performance results are given that demonstrate a low synchronization overhead of the framework.

1. Introduction

Hardware-in-the-loop (HIL) is a technique in which portions of a given system are simulated and portions are implemented as hardware devices. The implemented hardware is connected via a digital interface to a computer-simulated system model. HIL is beneficial in testing prototypes of those devices which have complex internal algorithms, those which would cause catastrophe if failed in the

field, and those for which building a laboratory test environment is difficult or impossible [6].

The model problem considered in this paper consists of a prototype of the Advanced Electric Power Grid [9] whose goal is to provide a self-healing power grid. It consists of an electric power grid augmented with power electronics controllers under distributed control. The controllers used in this experiment are Flexible AC Transmission System (FACTS) devices [4]. A FACTS device is attached to a single line of an electric power grid and control its power flow. Working together, under distributed control, multiple FACTS devices can improve the stability of the grid and mitigate blackouts [5]. To test such a system requires that failures of the power system be injected (contingencies, such as line removals) and FACTS device response measured. Clearly this is infeasible to test on the nation's power grid, so a real-time simulation is needed. The FACTS devices then need to respond to the changes in the simulated power grid. However, simulating a FACTS device in detail is not feasible. Thus, the simulated power system, coupled with laboratory-scale FACTS devices [10] form the laboratory which uses HIL techniques.

In an HIL, an important goal is that the simulated system demonstrate dynamics approximating those of the real system as closely as possible, with respect to the hardware under test (HUT). The accuracy of these dynamics depends both on the mathematical model of the simulation and on the latency of the simulation's response to changes in the state of the rest of the HIL system. Latency in the simulation response can have many sources both internal and external to the simulation program. While the implementation of the simulation model by itself dominates the computational time consumed by the simulation, an efficient simulation framework with real-time bounded internal delays is required to provide a predictable and minimal latency to the rest of the HIL system.

This paper focuses on an implemented HIL simulation

*This work was supported in part by NSF MRI award CNS-0420869 and CSR award CCF-0614633, and in part by the Missouri S&T Intelligent Systems Center.

framework. The framework implements a graphical user interface using Unix-style inter-process communication and shared memory techniques, as well as vendor-supplied extensions to the Linux kernel to allow shielding processors from hardware interrupts. The framework should be easily adaptable to any Unix-like system which has support for System V IPC, support for process pinning, a sufficient number of processor cores, and in which CPUs can be selectively shielded from hardware interrupts.

2. Review of Literature

A characteristic feature of virtually all of the surveyed HIL simulation systems is that the computer-based simulation component, whether it is programmed to simulate the hardware under test or the rest of the system, is built upon a commercial, proprietary simulation platform such as RTDS [2]. This dependency on a proprietary software platform presents a barrier for independent scientists to reproduce the results that have been reported in those papers and causes the simulation software itself to depend on the commercial platform product continuing to exist and be supported by its sponsor, limiting the useful life of the otherwise independent simulation software. Unlike most previous efforts, this paper describes a simulation framework that meets the open source definition. As provided for in the open source definition, it can be employed without restriction in any application because of its open source license [3].

In the context of power system HIL simulation, several methods have been developed for interfacing the simulated (software-based) and hardware-based components of the HIL system. The simplest method is to couple the systems using low-voltage Digital-Analog Converter (D/A) and Analog-Digital Converter (A/D) interfaces [13]. Control signals are sent to the hardware in digital or analog form and the analog state of the hardware's outputs are sampled back into the simulation. A more complex scheme, and one that allows for better validation of the HUT when applied appropriately, is referred to as PHIL (Power-Hardware-In-the-Loop), or as a "Sim-Stim" (Simulation-Stimulation) interface. A PHIL method implies that real electric power is being exchanged at the interface boundary between the simulated system and the HUT, thus simulating as closely as possible the real environment in which the HUT will exist [18]. A MIL (Model-In-the-Loop) interface is very similar to PHIL, but instead of the simulation driving amplifiers directly to generate the power and sampling from transducers as in the Sim-Stim interface, in the MIL approach an external conversion black-box is implemented that converts the A/D and D/A signals on the simulation side to the real power flow on the device side using voltage-source or current-source converters [19]. One unique approach to

the real-time HIL interface question is to implement the interface across a USB (Universal Serial Bus) bus using the isochronous transfer mode of USB, which provides for real-time bounded transfers across the interface [17]. The interface used in this work most closely resembles the MIL approach because low-voltage A/D and D/A interfaces are utilized between the simulation and the HUT.

HIL has been used as a method for determining optimal control parameters for power system components such as STATCOM load banks [15]. The HIL technique has also been proposed as a way to refine the parameters of existing systems, so there are potential applications for the HIL simulation technique beyond new system design [15]. In the context of this work, a STATCOM is a type of FACTS device.

Previous work was done in creating an open source, freely redistributable HIL platform called RT-VTB (Real-Time Virtual Test Bed) [12]. The approach of RT-VTB is to implement a real-time process in the kernel side of RTAI that sends a periodic "tick" to the simulation userspace process, which in turn polls the kernel process using the RTLinux FIFO mechanism. One tick of the simulation userspace process updates the real world variables from A/D sampling, computes the next time step, and outputs any control signals via D/A.

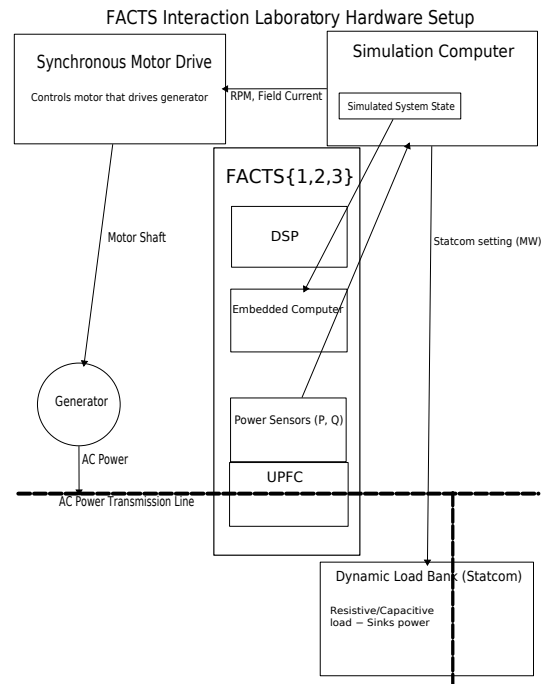


Figure 1: The laboratory setup.

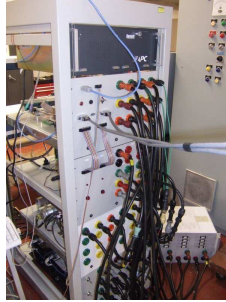


Figure 2: A FACTS device with embedded computer.

3. Experimental Setup - System Hardware/ Software

The overall experimental setup is to test the response of the Flexible AC Transmission System (FACTS) devices to changes in power flow on the electric power grid as well as determine frequency domain interactions with other FACTS devices. The power grid is simulated in real-time using a Simulation Computer. The Simulation Computer controls a physical generator attached to an AC Power Transmission Line and a Dynamic Load Bank such that the power flow over this line is governed by the simulation of the power system. The FACTS consists of several components, including power sensors, a DSP, an embedded computer, and power electronics (Figure 1, Figure 2, [10]). This type of FACTS device is a Unified Power Flow Controller (UPFC). The FACTS device is the HUT and resides on the AC Power Transmission Line. The FACTS devices responds to changes on the AC Power Transmission Line, which, in turn, influences the simulation. A/D hardware senses the changes on the AC Power Transmission Line and the Simulation Framework injects these readings into the simulation. This interaction forms a real-time computational loop in the system. The frequency response of the UPFC on the AC Transmission Line is capped at 300 HZ. This frequency response governs the sampling requirements of the Simulation Computer, which, in turn, governs the simulation time step of the computation.

The embedded computer implements a distributed control (Long-Term Control(LTC)) algorithm that controls the settings of the FACTS. The embedded computer communicates with the simulation computer via ethernet and receives operator input from the keyboard. The FACTS is implemented as a pair of compensators that control real and reactive power flow on the transmission line. The experiments conducted to exercise the Simulation Framework were carried out on a 4-processor Concurrent Computer iHawk rack-mountable system with attached A/D and D/A hardware.

4. Overview of HIL Simulation Framework

In this work, a framework has been produced that enables soft real-time simulation with no specialized hardware support and no specialized operating system support for real-time scheduling, meaning it can be run on a Commercial, Off-The-Shelf (COTS) Unix system such as Linux as long as it complies with the respective standards. The simulation algorithm itself has no requirement beyond the fact that it must run fast enough by itself on the given system's CPU and memory architecture to satisfy the real time constraints of the system. The framework imposes few additional constraints on the hardware and operating system beyond those imposed by the simulation core.

4.1. Requirements

A supervisor process such as an operating system delegates processing time to constituent processes based on algorithms that are not deterministic from the perspective of the constituent process in the general case. For any type of real-time computation running under a supervisor process, if it is at all possible to meet the real-time constraints, the most obvious starting point towards meeting those constraints is to maximize the proportion of time that the computation process is in "running" or "ready to run" state compared to the time that it is in other states, such as "waiting on hardware" or "waiting on lock". Thus, the primary goals of this framework were to decouple the management of A/D and D/A hardware from the simulation process itself and to use inter-process communication techniques that cause minimal blocking in the simulation process.

4.2. Asynchronous Hardware Management

Unix drivers typically include a userspace library that abstracts the kernel driver interface and which is called directly by the application wishing to use the hardware instead of the application calling the kernel interface directly. The API of the library thus insulates the application from changes in the implementation of the driver, which can be quite volatile.

The purpose of decoupling the management of A/D and D/A hardware from the simulation process is to eliminate the overhead caused by peripheral device I/O. This overhead comes from calling the userspace library, transitioning to kernel mode, and communicating with the peripheral through its registers. Sometimes the kernel mode transition can be eliminated if the device supports memory mapped I/O, but in all cases the sequence of device communication must be repeated every time analog data is to be captured or sent. Placing the burden of hardware communication on

the simulation process is unreasonable in two ways. First, it adds to the baseline latency of a simulation time step due to the layers of driver code described above, ensuring that one iteration of the simulation loop can never be completed in less real time than the hardware I/O requires. It also requires making a difficult decision about the location of the code performing the analog data updates relative to the code implementing the system solver loop. If this code is placed outside the system solver loop, the most recent updates to the sensor readings are unavailable to the solver until a time step has passed, and a latency between the solver updating the system state and the real world outputs reflecting the updated state occurs. However, placing the update code inside the system solver loop requires more indirection of memory access, slowing the solver; the complexity of the inner solver code itself is also increased, potentially causing instruction cache misses. Decoupling the A/D and D/A and having a separate process asynchronously merge the simulated system state with the real world state avoids these problems and allows as fine a granularity of A/D and D/A sweeps as the application requires. For the remainder of this paper, interface variables refer to the variables that are sampled externally from A/D hardware and then injected back into the system.

4.3. Interprocess Communication Alternatives

Available inter-process communication (IPC) mechanisms are signals, pipes, sockets, message queues, semaphores (POSIX and SysV implementations), and shared memory (either shared pages external to the process as in POSIX shared memory or a shared program area and heap memory as in POSIX threads).

Many of these IPC primitives have conditions under which they block (suspend) the process utilizing them. This would be unacceptable for the simulation process since its computation is under a real-time constraint. Some sort of shared memory approach was thus required, so that IPC could occur without a system call. A thread-based approach was rejected as the available operating system did not have an explicit real-time scheduler. Instead, processes were scheduled on multiple cores of the simulation platform. System V semaphores were implemented as relatively simple DOWN() and UP() macros that are used throughout the codebase. To utilize System V semaphores there is no choice but to use the semop() system call. To avoid blocking, a set of “new data” flags was implemented in an approach similar to a general double-checked locking approach. To be consistent with the implemented framework code, these flags will hereafter be referred to as “stale”

flags. (A “stale” flag can simply be regarded as an active-low “new data” flag.) Each stale flag X_f for interface variable X exists as an atomically-updatable data type and corresponds to the data item or set that is protected by a lock $P(X)$. When that data set is updated by a writer holding the corresponding semaphore, the stale flag is also reset before releasing the semaphore. An interested reader can check the stale flag before attempting to take the lock $P(X)$. If the stale flag X_f is set, there is no reason to take the lock $P(X)$, since the data set X has not yet been updated since the last time it was read, and in this case a system call is avoided. If the stale flag X_f is reset (not set), then the computational process takes the lock $P(X)$, reads the interface variable X , sets the stale flag X_f , and finally releases the lock $P(X)$ (Figure 3).

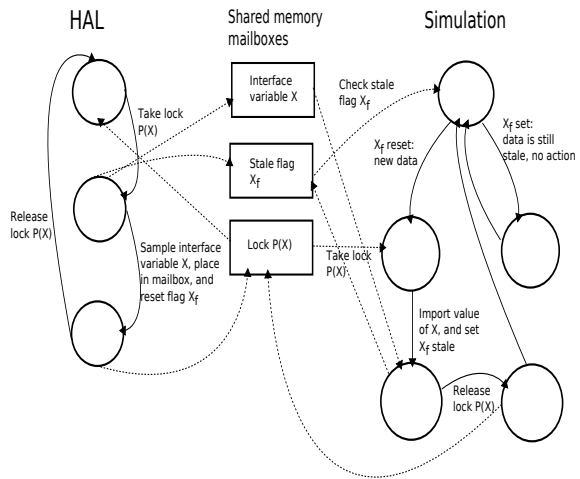


Figure 3: The locking scheme, with stale flags.

In this paper, individual slots in shared memory will be referred to as “mailboxes” (Figure 4).

5. Discussion of HIL Simulation Framework

The simulation framework contains many components. Each component of the framework along with some details about its implementation will be described. Discussion about the implementation of the simulation itself will be limited to those aspects which affect its integration into the framework; the simulation’s algorithmic implementation will remain a “black box” to assure the applicability of the framework to any simulation satisfying the requirements.

5.1. System Block Diagram

In Figure 4, the framework is decomposed into several principal components. The Simulation Engine is comprised

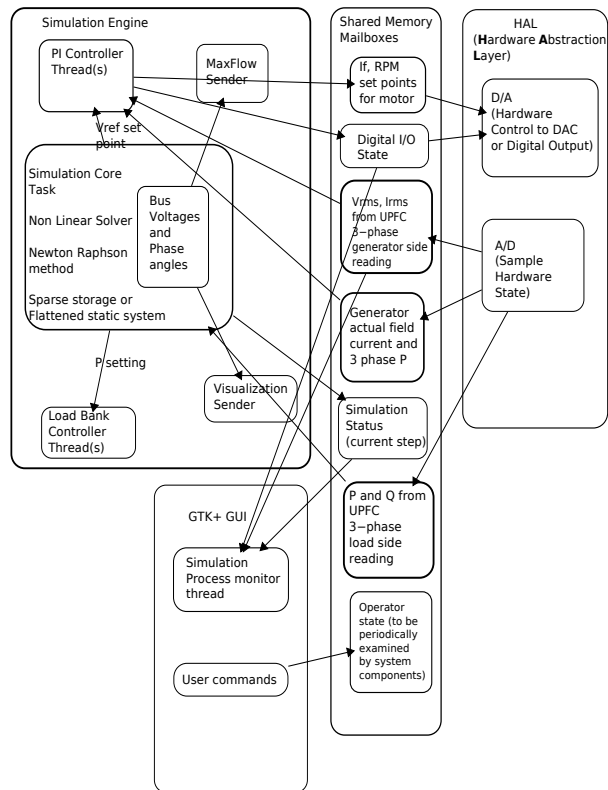


Figure 4: High level diagram of the simulation system.

of the PI (Proportional-Integral) controller that controls part of the AC Power Transmission Line (the HIL line), a Simulation Core that contains the numeric solver, a Load Bank Controller that controls the load of the HIL line, and a Visualization and Long Term Control sender. Latter two elements reflect sending state information from the simulated power system. These interact with the Hardware Abstraction Layer (HAL) through Shared Memory regions.

The Simulation Framework uses a multi-process design. Challenges are coordination of access to shared data and avoiding race conditions through the use of locks. Misuse of locks or the accidental introduction of a lock ordering bug could lead to seemingly-random deadlock – a partial or total halt of the system. Misuse of locks could also foil the real-time performance of the system by stalling an important process for longer than otherwise necessary.

To reduce complex scheduling, and to take advantage of the available multi-core architecture, a multi-process design was used where there are at least $N+1$ processor cores in the system, where N is the number of concurrently running processes that hold a lock. Those lock-holding processes are pinned to distinct CPUs so that no other process can be scheduled on the CPU that a lock-holding process owns.

Along with shielding critical processors from hardware interrupts and disallowing calls to non-realtime-safe code inside critical sections, this scheme guarantees that the time that any process holds the lock is bounded, ensuring that the simulation process time step – which takes a lock itself in order to read the asynchronously updated values of its real world variables – is bounded. The framework described in this paper utilizes a shared memory architecture, where the A/D is sampled and the D/A state is updated in a process that runs concurrently with the simulation, and where that process shares the simulation state memory as in the approach of [17].

5.2. HAL (Hardware Abstraction Layer)

The approach of this framework is simply to sample the real world values asynchronously so that they can be sampled as frequently as necessary. Based on the Nyquist rate, a 1ms sampling rate provides the simulation with sufficient dynamics with respect to the 300 Hz filter in the FACTS [16]. HAL runs as a separate process and acts as a concentrator for all physical I/O.

Its main loop runs as often as the application requires. On a SMP (Symmetric Multi-Processor) system such as in this lab, one processor can be dedicated to running the HAL process. In this case, the main loop can run continuously and update the hardware state as quickly as the hardware allows because the CPU does not have to yield to other processes such as the simulation process or user interface.

The HAL main loop simply does the following in pseudocode:

```
SweepDtoA();
CollectAndConvertAD();
usleep(DELAY);
```

The function of SweepDtoA is to take the values that are in the mailboxes which correspond to what the current state of the DAC and Digital I/O outputs should be, and write out the entire state to the hardware. The state of the real hardware should thus consistently follow the state of these variables in the mailbox.

CollectAndConvertAD is a collection of steps taken to process the analog readings into an interface variable in the system. Several variables, such as voltage and current readings, are filtered through a first-order low-pass filter (LPF) to reduce sampling noise caused by sensor inaccuracy. The scale factors that are used to scale a voltage level to a real value to be placed in an interface variable are dependent both on the particular data item and the particular sensor that was used to gather its value, and thus are experimentally determined and then encoded as magic constants. The flow of data for one interface variable is depicted in Figure 5.

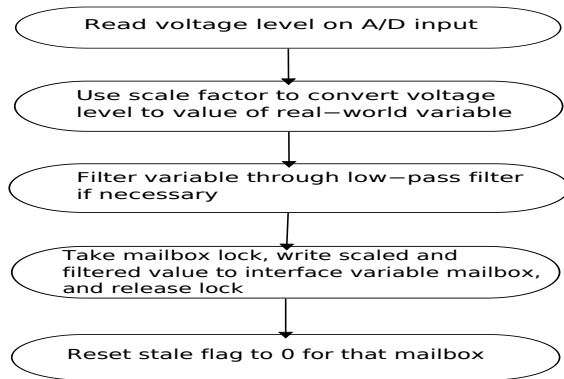


Figure 5: Data flow for an interface variable being sampled into the system.

It should be noted that only memory-mapped I/O should be utilized in the HAL and the interface card drivers during execution of the simulation. Using any form of system-call based I/O during the simulation besides that required to synchronize shared memory access would introduce additional real-time dependency on the operating system in order to ensure correct and safe hardware operation in the general case. Good practice would be to implement any timing-critical hardware control in separate controller threads that do not participate in the shared-memory locking such as in the PI Controller implementation described in Section 5.5.

5.3. Simulation Core (Flatten4)

The simulation core is loosely coupled to the HAL framework. It consists of a library that implements the solution of the non-linear power system at a particular time step. The simulation core takes the physical connections, bus voltages, phase angles, power generation, and admittances of the power system as inputs, and calculates the new bus voltages and phase angles using a non-linear system solver, specifically Newton-Raphson (NR) [8, pp 52-69]. The NR method involves repeatedly updating a Jacobian matrix that describes the partial derivatives of the state variables with respect to each other and directly solving the resulting linear system, until the error term variables (ΔP and ΔQ) are driven close enough to zero. This should take two to three iterations in the average case, but may take many more depending on how far away the system is from the steady state.

A new approach was developed to symbolically perform the LU decomposition at compile time, avoiding the expensive indexing and multiplications associated with the LU decomposition in each NR solution. This was accomplished by writing a MATLAB program that generated the C source code files comprising the solver [14]. Memory accesses performed by the C program were “flattened” so that memory

was accessed through a single pointer for each matrix instead of through multiple levels of indirection.

This paper’s approach is to enable a soft-real-time HIL simulation to be built by first relaxing the real-time constraints on the simulation algorithm itself. It cannot always be guaranteed that a non-linear system with an arbitrary set of values for the system’s real world variables will have a bounded number of steps to convergence with an iterative solver such as NR [7]. Once the power system has been allowed to converge to a relatively steady state, convergence of future time steps will be very fast (2 or 3 iterations); small changes in the real world values will not impact this fast convergence. However, when the system encounters a contingency such as the removal of a line, a change in generator voltage, or a change in power flow through the FACTS device under test, it initially violates the real time constraint, but then catches up within several time steps. It is this average behavior that leads to a soft real-time constraint on the simulation system. Since the system is simulating, and sampling, a continuous phenomena, the experimental setup is able to tolerate missed simulation time outputs with minimal loss of accuracy, although we do not have a quantification of this particular facet of the system.

For the power system dynamics, a 1 ms simulation time step is tied to the numerical analysis of the system, the sample rate, and, ultimately, the frequency response of the HUT.

5.4. Simulation Driver

This program wraps the simulation core and takes care of minimal locking to update the simulated system state from the sampled hardware state. The simulation task also updates the state of the controller threads for the external hardware and processes such as the generator and loadbank controllers. It either takes the shared memory region passed to it by the controller process, or sets up its own shared memory region. It then spawns the HAL process and passes it the handle for the shared memory region.

5.5. PI (Proportional-Integral) Controller(s)

The PI controller program is run as a thread within the simulation process. The PI controller thread is in charge of controlling the motor hardware and ensuring that it is left in a safe condition and that all physical constraints are met. It also controls the output voltage of the generator by varying the field current (I_f).

The PI controller uses the mailbox system like any other simulation component. It sets the RPM and field current through the mailbox interface for the D/A card and reads back the output voltage and field current through the mailbox interface for the A/D card. Separate mailboxes are supplied for up to three PI controllers.

The PI controller's only input is a voltage set point for the bus to which the FACTS generator side is connected. This set point is determined by the output of the simulation for that time step. When the set point is updated, the PI algorithm steps the output field current to approach the new V set point asymptotically. The rate of stepping is controlled by the constants K_i and K_p . The further away the present V value is from the set point, the faster the new value will be approached.

If at any point the generator voltage, output current, or field current go out of range, the PI controller shuts down the motor.

5.6. System State Transmitter

The system state transmitter component transmits the current values of voltage and phase angles at each bus of the power system in response to incoming network requests. This state data is used by the algorithm in the Long Term Control (LTC) process of the FACTS and by a power system visualization console (not shown). This state data is analogous to that which would be collected from the readings of the physical electric power grid.

The transmitter is implemented as a thread which shares the system state with the simulation. The thread implements a minimal server which receives incoming connections and then enters a request loop. The request loop only supports a handful of commands, the most important of which is to initiate a state dump, at which point the system's "V" and "theta" variables are captured in a snapshot and then copied across the network. Since this is an infrequent operation, it might be possible for the LTC to cause a slowdown in the simulation by repeatedly requesting the state variables, causing an increase in lock contention for the snapshot copy and a possible missed deadline for the simulation.

There are several approaches to providing the snapshot requests, depending on how the snapshot data is to be used. If the snapshot is required to be consistent with respect to the simulation time step, one approach is to limit the snapshot requests to no more than a certain number of requests per time period. The snapshot would have to be first copied into a local buffer before being sent to the network layer for transmission so that the lock is not held for an unreasonable amount of time. If the snapshot does not necessarily have to be consistent with respect to a simulation time step, another approach is possible. Since all of the simulation state data are stored in atomic data types according to the system design, the snapshot mechanism could be implemented so that the simulation simply writes the state data as usual and the state is read "as-is" from the buffer, possibly cutting into the middle of a time step, but requiring no lock at all.

6. Results and Discussion

Experiments were performed with several different settings for the HAL update delay: $100 \mu s$, $50 \mu s$, $5 \mu s$, and no delay. The time the simulation spent acquiring the lock, as well as the time spent in each critical section, were then examined.

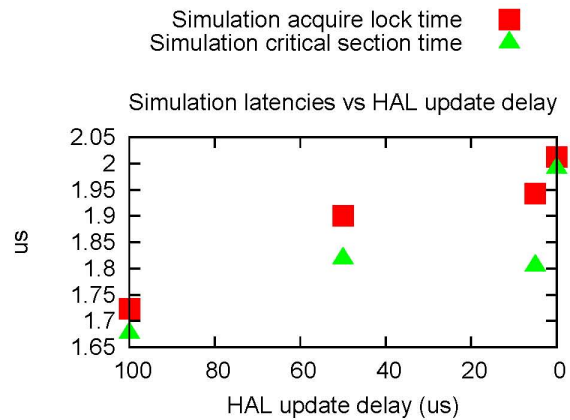


Figure 6: Simulation latencies versus HAL update latency.

The results as measured in several full HIL simulations are given in Figure 6, and show the lock acquisition latency to be less than $20 \mu s$ which is less than 5% of a 1 ms simulation time step. The figure also shows that the average time to acquire a lock increases as expected when the HAL update delay decreases (with significance as shown by a single tailed t-test with $\alpha=.01, n=36772-66912, \sigma=1.18-2.14$). This is because the time the simulation spends acquiring the lock primarily depends on how frequently the HAL takes the lock to update the interface variables; when the delay is large, the lock is taken less often. While at the limit the hardware can be updated constantly while remaining useful with this particular simulation core, there remains a balancing act between having too many locks, which adds constant runtime overhead even when lock contention is low, and too few locks, which increases lock acquisition latency due to waiting.

With a power system simulation implemented in this framework, the simulation is able to respond to changes in the power flow through the FACTS device in soft real time with update latency determined by the polling frequency of the HAL component. A system based on this framework has been in use since March 2007 in the FACTS Interaction Laboratory [1] at the Missouri University of Science and Technology.

7. Conclusions

In this paper, the goal was to create a framework capable of soft real-time support for an HIL power system simulation. This goal has been accomplished by enabling soft real-time support at a 1 ms time step. While a 1 ms real-time response is not unprecedented in the field, the goal of a generalized open-source framework has been realized. This framework can be used with an arbitrary real-time simulation core and on any Unix-like operating system with the appropriate scheduling controls. There is no requirement that the simulation core be a power system simulation, so any HIL application requiring a low average latency can benefit from this framework.

8. Future Directions

Conceptually it is possible to integrate the entire simulation system into a single process with several POSIX threads. An external semaphore region will still be necessary unless the synchronization macros are ported to POSIX semaphores. A drawback to this approach, as noted above, is that a threaded system presents more problems with managing CPU affinity and scheduling priority.

With respect to the real-time simulation, another avenue for exploration could be the implementation of the techniques described in [11], in which an interpolation technique is combined with a variable time step. This method attempts to account for events that happen during a time step. It would make the simulation more accurate, but at the cost of computational complexity, and it is unclear if the implementation of such a technique would be feasible in the presence of real-time constraints. Additionally, a rigorous method for the claim that "occasionally missing a real-time constraint does not impact simulation correctness" could be developed along the lines of [16].

References

- [1] Fault-Tolerant and Secure Power Grid Systems using FACTS Devices; FIL Homepage. Online. <http://filpower.umr.edu/>, fetched August 22, 2007.
- [2] RTDS Technologies, Inc. Homepage. Online. <http://www.rtds.com/>, fetched August 22, 2007.
- [3] The Open Source Definition. Online. <http://www.opensource.org/docs/osd>, fetched September 11, 2007.
- [4] IEEE Power Engineering Society FACTS Application Task Force, FACTS Applications. 1996.
- [5] A. Armbruster, M. Gosnell, B. McMillin, and M. Crow. The maximum flow algorithm applied to the placement and distributed steady-state control of FACTS devices. In *Proceedings of the 37th Annual North American Power Symposium (NAPS)*, pages 77–83, October 2005.
- [6] M. Bacic. On hardware-in-the-loop simulation. In *44th IEEE Conference on Decision and Control (CDC-ECC '05)*, number 44, pages 3194–3198, December 2005.
- [7] P. Baracos, G. Murere, C. Rabbath, and W. Jin. Enabling PC-based HIL simulation for automotive applications. In *IEEE International Electric Machines and Drives Conference, 2001 (IEMDC 2001)*, pages 721–729, 2001.
- [8] M. Crow. *Computational Methods for Electric Power Systems*. CRC Press, 2002.
- [9] M. Crow, C. Gill, F. Liu, B. McMillin, D. Niehaus, and D. Tauritz. Engineering the advanced power grid: Research challenges and tasks. In *RTAS 2006 Workshop on Research Directions for Security and Networking in Critical Real-Time and Embedded Systems (CRTES '06)*, San Jose, CA, USA, April 4 2006.
- [10] L. Dong, M. L. Crow, Z. Yang, C. Shen, L. Zhang, and S. Atcity. A reconfigurable FACTS system for university laboratories. *IEEE Transactions on Power Systems*, 19(1):120–128, February 2004.
- [11] M. O. Faruque, V. Dinavahi, and W. Xu. Algorithms for the accounting of multiple switching events in digital simulation of power-electronic systems. *IEEE Transactions on Power Delivery*, 20(2):1157–1167, April 2005.
- [12] B. Lu, X. Wu, H. Figueroa, and A. Monti. A low cost real-time hardware-in-the-loop testing approach of power electronics controls. *IEEE Transactions on Industrial Electronics*, 54(2):919–931, April 2007.
- [13] W. Ren, L. Qian, M. Steurer, and D. Cartes. Real time digital simulations augmenting the development of functional reconfiguration of PEBB and universal controller. In *Proceedings of the 2005 American Control Conference, 2005*, volume 3, pages 2005–2010, June 2005.
- [14] W. M. Siever. *Power Grid Flow Control Studies And High Speed Simulation*. PhD thesis, University of Missouri-Rolla, Rolla, MO, 2007.
- [15] M. Steurer, S. Woodruff, N. Brooks, J. Giesbrecht, H. Li, and T. Baldwin. Optimizing the transient response of voltage source converters used for mitigating voltage collapse problems by means of real time digital simulation. In *2003 IEEE Bologna Power Tech Conference Proceedings*, volume 1, page 6, June 2003.
- [16] Y. Sun, B. McMillin, X. F. Liu, and D. Cape. Verifying Noninterference in a Cyber-Physical System: The Advanced Electric Power Grid. In *Proceedings of the Seventh International Conference on Quality Software (QSIC)*, Portland, OR, October 2007.
- [17] R. B. Wells, J. Fisher, Y. Zhou, B. K. Johnson, and M. Kyte. Hardware and software considerations for implementing hardware-in-the-loop traffic simulation. In *The 27th Annual Conference of the IEEE Industrial Electronics Society, 2001 (IECON '01)*, volume 3, pages 1915–1919, Nov/Dec 2001.
- [18] X. Wu, S. Lentijo, and A. Monti. A novel interface for power-hardware-in-the-loop simulation. In *2004 IEEE Workshop on Computers in Power Electronics, 2004*, pages 178–182, August 2004.
- [19] W. Zhu, S. Pekarek, J. Jatskevich, O. Wasynczuk, and D. Delisle. A model-in-the-loop interface to emulate source dynamics in a zonal DC distribution system. *IEEE Transactions on Power Electronics*, 20(2):438–445, March 2005.