Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

ORTiS solver codegen: C++ code generation tools for high performance, FPGA-based, real-time simulation of power electronic systems

Matthew Milton[a,*], Andrea Benigni[b,c]

[a] Department of Electrical Engineering, University of South Carolina, 301 Main Street, Columbia, SC, 29208, USA
[b] Institute of Energy and Climate Research: Energy Systems Engineering (IEK-10), Forschungszentrum Jülich, Germany
[c] Chair of Methods for Simulating Energy Systems, RWTH Aachen University, Germany

ARTICLE INFO

Article history:
Received 28 October 2020
Received in revised form 24 December 2020
Accepted 5 January 2021

Keywords:
Real-time simulation
Code generation
High performance computing
High level synthesis
Power electronics

ABSTRACT

The solver code generation tools of the Open Real-Time Simulation (ORTiS) framework are a C++ library and CLI tool designed to create real-time simulation solvers for power electronics systems. These C++ defined solvers – generated by the tools – support high level synthesis to HDL, enabling the implementation of FPGA solvers capable of nanosecond resolution in real-time. The ORTiS Solver Codegen tools support the creation of multi-FPGA solvers and the use of user-defined power electronic component models; system level models are described by netlists. These tools enable engineers to perform hardware-in-the-loop testing of power electronic systems with high frequency dynamics, using time steps as small as 35 nanoseconds.

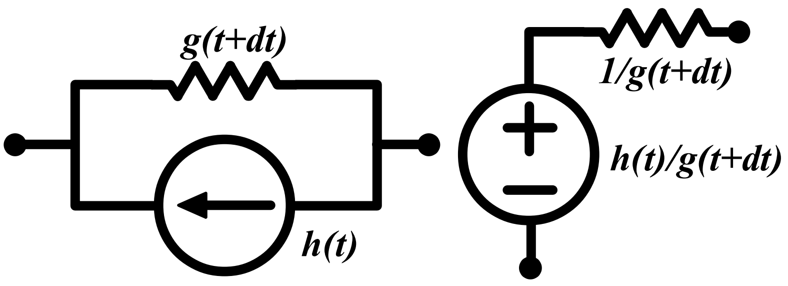
Code metadata

Current code version	v0.8
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-20-00073
Code Ocean compute capsule	N/A
Legal Code License	GNU General Public License (GPL) v3.0
Code versioning system used	git
Software code languages, tools, and services used	C++
Compilation requirements, operating environments & dependencies	For Windows and Linux, builds using GCC or MinGW-w64 with C++14 support; requires Eigen 3 (http://eigen.tuxfamily.org)
If available Link to developer documentation/manual	N/A
Support email for questions	mmilton@email.sc.edu

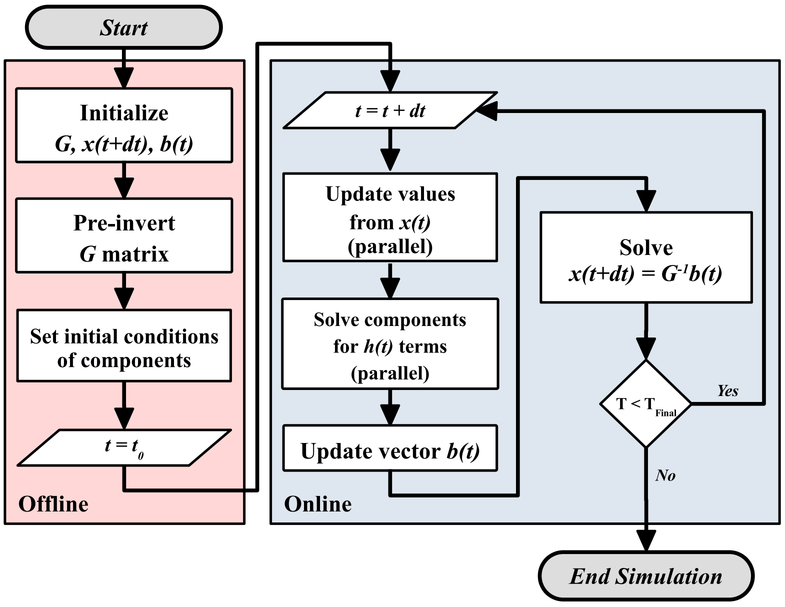
1. Motivation and significance

During the development of Power Electronic (PE) systems, Hardware-In-the-Loop (HIL) real-time simulation is often performed to reduce time to market of these products. From the rise of modern large scale PE systems with high switching frequency converters (100–200 kHz), fast control and fault protection mechanisms, and interplay of high harmonics in large systems, the need for Real-Time (RT) simulation tools able to model and simulate these systems with high fidelity has increased. To achieve such fidelity, time steps below 100 ns are required. In recent work [1–3], RT simulation solvers have utilized Field Programmable Gate Arrays (FPGA) for their low latency and high parallelism to solve PE systems with small time steps ≤ 1 µs. Though these particular RT solvers exist, there is a lack of tools available to rapidly custom-model and deploy PE system simulations using such solvers with FPGA execution. Commercial tools provide for PE systems RT simulation, but these tools are limited to CPU execution at time steps of 10 µs, with 200 ns being possible on FPGA hardware in exchange for limited modeling flexibility. Moreover, these commercial tools only provide for simulation under their respective proprietary and expensive hardware, and require purchased licenses.

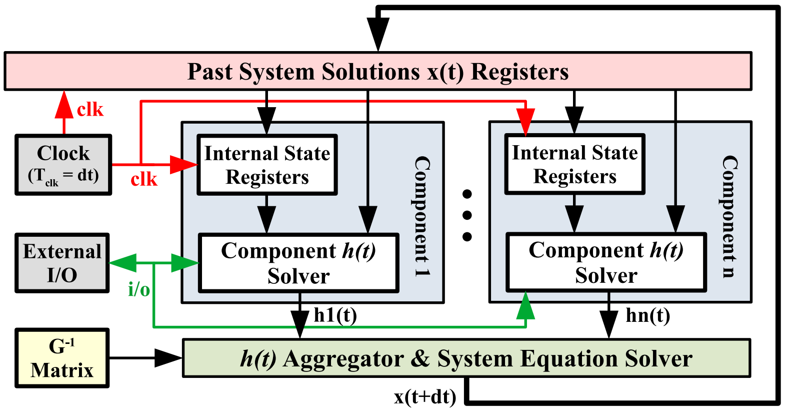
* Corresponding author.
E-mail addresses: mmilton@email.sc.edu (M. Milton), a.benigni@fz-juelich.de (A. Benigni).



(a)


Start
Initialize G, x(t+dt), b(t)
Pre-invert G matrix
Set initial conditions of components
t = t0
Offline
t = t + dt
Update values from x(t) (parallel)
Solve components for h(t) terms (parallel)
Update vector b(t)
Online
Solve x(t+dt) = G-1b(t)
T < TFinal
Yes
No
End Simulation

(b)



(c)

Fig. 1. LB-LMC modeling and solver design; (a) Resistive companion models; (b) Solution flow; (c) Solver FPGA structure.

In this paper, we present C++ solver code generation (codegen) tools, released as free, open source software under the Open Real-Time Simulation (ORTiS) framework, which can conveniently and rapidly generate RT simulation solvers of PE systems. Utilizing the Latency-Based Linear Multistep Compound (LB-LMC) solver method [4], the codegen tools produce C++ defined solvers of PE systems that can RT execute on a wide range of FPGA devices as High Level Synthesized (HLS) cores, with time steps as low as 35 nanoseconds on modern FPGAs. The solver codegen tools support defining PE system models as plain-text netlists indicating components and their network connections in the model. Individual component models can either be ones built into the tools, or be user defined ones added to the tools' source code. To allow multi-FPGA execution, the codegen tools support defining subsystems of a PE system, with a solver per subsystem created using the decomposition approach presented in [5]. This paper provides a summary on the codegen algorithms and their usage to create PE system solvers RT-executed on FPGAs.

2. Software description

2.1. Summary of LB-LMC method

The LB-LMC method [4] used in the solvers generated by the ORTiS codegen tools is a highly-parallelizable algorithm for

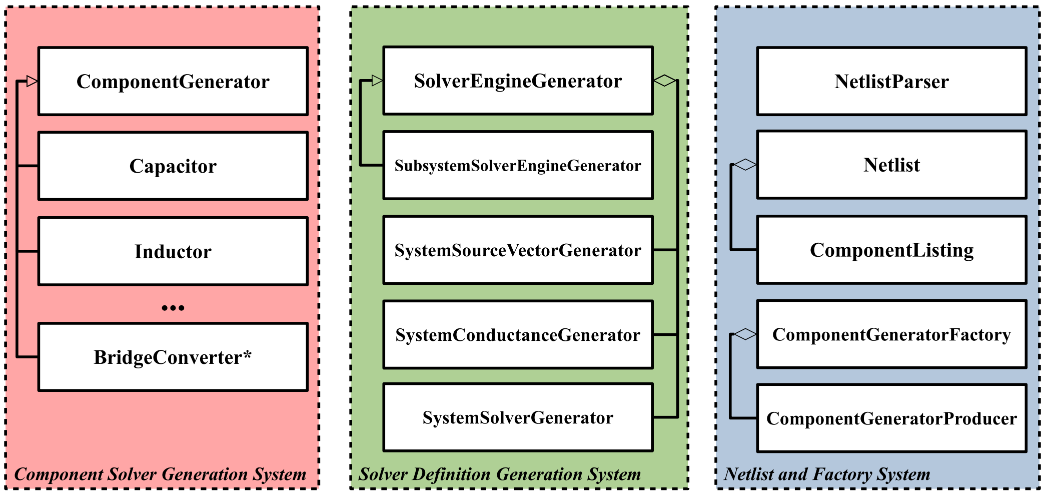

ComponentGenerator
Capacitor
Inductor
...
BridgeConverter*
Component Solver Generation System
SolverEngineGenerator
SubsystemSolverEngineGenerator
SystemSourceVectorGenerator
SystemConductanceGenerator
SystemSolverGenerator
Solver Definition Generation System
NetlistParser
Netlist
ComponentListing
ComponentGeneratorFactory
ComponentGeneratorProducer
Netlist and Factory System

Fig. 2. ORTiS LB-LMC solver codegen library structure.

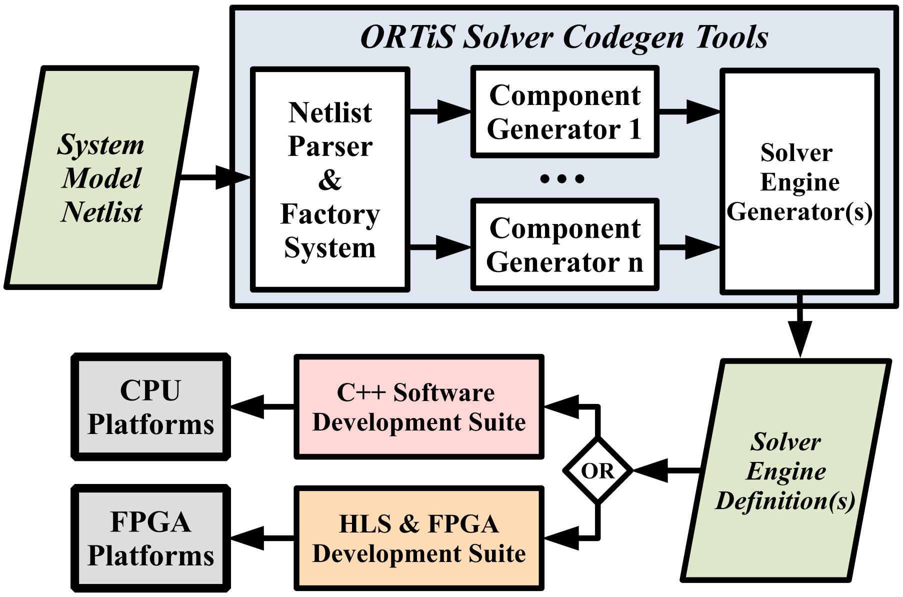

ORTiS Solver Codegen Tools
System Model Netlist
Netlist Parser & Factory System
Component Generator 1
...
Component Generator n
Solver Engine Generator(s)
Solver Engine Definition(s)
OR
C++ Software Development Suite
HLS & FPGA Development Suite
CPU Platforms
FPGA Platforms

Fig. 3. LB-LMC solver code generation flow.

solving nonlinear electrical/PE systems, similar to the Electro-Magnetic Transient Program (EMTP) and other Resistive Companion (RC) methods. Under the method, each component of a PE system is defined as a discretized state space (SS) model embodied as a collection of RC circuits, seen in Fig. 1(a). Each RC circuit consists of sources $h(t)$ representing past terms of the model, with conductances $g(t + dt)$ embodying present model terms. From the models, a set of equations (1) is produced to be solved each simulation time step of length dt for system solutions vector $x(t + dt)$. Matrix $G(t + dt)$ and vector $b(t)$ respectively aggregate the $g(t + dt)$ and $h(t)$ terms of each component model.

$$G(t + dt)x(t + dt) = b(t) \tag{1}$$

Unlike EMTP or other RC methods, all nonlinear components in a system are discretized with explicit integration, where nonlinear SS models consist of only past terms at time $\leq t$; linear components are discretized implicitly. As such, the $G(t + dt)$ term of (1) is held constant for all time and the equations for nonlinear systems are solvable without iterative process (Newton–Raphson), allowing nonlinear systems to be quickly solved using linear algebra approaches without loss of the nonlinearity. Moreover, the LB-LMC method solves models of all system components entirely in parallel, allowing for computational speedups to achieve small time steps in real-time, especially under FPGA execution. In [6,7], the method was demonstrated to RT simulate multi-converter switching PE systems with 50 ns time steps on Xilinx 7-series FPGAs.

To solve a PE system model for a time step under LB-LMC method, the flow of Fig. 1(b) is followed. Offline, the G, x, and b terms are initialized, G is pre-inverted, and component models are set to initial conditions. Then, during online simulation, past $x(t)$ is updated, component SS models are solved in parallel for their $h(t)$ terms, and $b(t)$ is updated from these terms. Finally, (1) is solved for $x(t + dt)$. If simulation continues, then the process repeats for another time step. For FPGA execution, the LB-LMC solver is implemented as in Fig. 1(c), where each component model of a PE system is given a solver core to solve for $h(t)$ and their solutions are passed to a system solver core that solves (1). The LB-LMC solver core is implementable to execute in dataflow manner to solve the system for single time step within a execution clock cycle, using fixed-point math. RT execution is achieved by setting clock period to dt; see [6,7] for details.

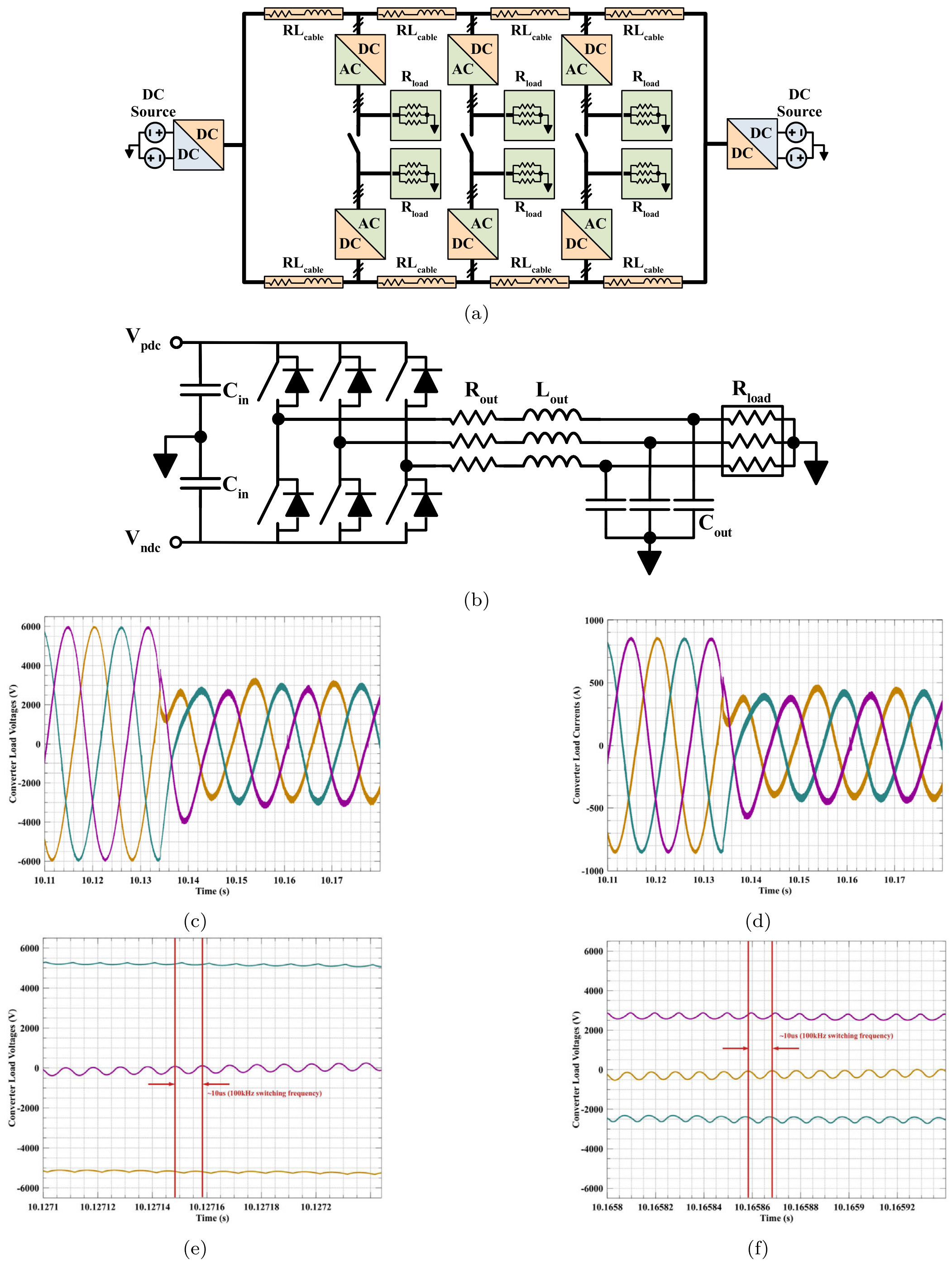


Fig. 4. Example models and their RT simulation results; (a) shipboard PE system; (b) DC/AC converter; (c)(d) RT simulation results of R_{load} voltages and currents; (e)(f) magnification of (c) before and after voltage change.

2.2. Software architecture

The ORTiS solver codegen tools is a C++ library consisting of code generation classes whose instances generate and aggregate C++ code for PE system LB-LMC solver definitions. As depicted in Fig. 2, the tools contain classes to generate component model $h(t)$ solver code; generate system solvers to solve (1); compute G of a PE system; and aggregate generated code into a custom C++ definition of the LB-LMC solver for a PE system. The tools also includes classes to load PE system models from plain-text netlist files. The generated LB-LMC solvers are defined as C++functions which can be compiled for RT CPU execution or be high level synthesized (HLS) for RT FPGA execution. To ease use of the solver codegen tools, a Command Line Interface (CLI) executable is provided as source code with the library, which when built can load netlist files and output solver definitions for given netlists.

2.3. Component solver generation

To generate model $h(t)$ solver code for components in a PE system, *ComponentGenerator* classes are provided by the solver codegen tools. For each component type included in the tools is a derived *ComponentGenerator* class, such as for capacitors, switching PE converters, etc. Each instance of a *ComponentGenerator* takes as input model parameters and the indices of the

```
%parameters of model

#name shipboard_zonal_system

#const DT 50.0e-9
#const CABLE_L 1.0e-6
#const CABLE_R 0.01
#const INV_CIN 0.001
#const INV_CFILT 1.0e-6
#const INV_LFILT 0.0001
#const INV_RFILT 0.0
#const LOAD_R 7.0

% rest of parameters...

% component listings of model

Inductor cable_l11(DT, CABLE_L) {15,16}
Inductor cable_l15(DT, CABLE_L) {31,32}
Resistor cable_r011(CABLE_R) {16,17}
Resistor cable_r015(CABLE_R) {32,33}
BridgeConverter3LegIdealSwitches inv6(DT, INV_CIN, INV_LFILT, INV_RFILT) {17,0,33,52,53,54}
Capacitor inv_c16(DT, INV_CFILT) {52,0}
Capacitor inv_c17(DT, INV_CFILT) {53,0}
Capacitor inv_c18(DT, INV_CFILT) {54,0}
Resistor rload16(LOAD_R) {52,0}
Resistor rload17(LOAD_R) {53,0}
Resistor rload18(LOAD_R) {54,0}

% rest of system model netlist...
```

Fig. 5. Netlist for portion of shipboard system.

component node connections in a PE system network. Based on the component model and given inputs, a *ComponentGenerator* will generate C++ code as a character string which is passed to a *SolverEngineGenerator* (SEG) instance that generates the actual PE system solver definition. This code will include the model solver expressions and external signal port definitions that act as inputs and outputs to the component solver, such as control and feedback signals for PE converters; these signal ports are realized as arguments to the system solver function. Moreover, the *ComponentGenerators* compute their constant conductances $g(t+dt)$ and stamps them into a G matrix stored by the SEG, along with providing information to the SEG on how component $h(t)$ terms map to $b(t)$ of the system. User defined components can be added to the codegen tools by extending the *ComponentGenerator* base class through inheritance.

2.4. Solver engine generation

To generate the PE system solver definition, a *SolverEngine-Generator* (SEG) class is provided. A SEG instance generates a function definition for the system solver whose arguments are the returned output $x(t+dt)$ and input/output signal ports of component solvers. For each time this solver function is called, it solves its system model for a single time step. Each SEG instance generates a solver function by taking the model code, signal port definitions, and RC model (h, g) information from each instanced component generator and aggregating the code together into the body and signature of the function. Then, the SEG computes and defines G^{-1} in the function, generates code to aggregate $h(t)$ terms of each component solver into a $b(t)$ vector, and finally produces code to solve (1) as (2); using internal *SystemConductanceGenerator*, *SystemSourceVectorGenerator*, and *SystemSolverGenerator* objects.

$$x(t+dt) = G^{-1}b(t) \tag{2}$$

During code generation, the SEG connects the inputs, outputs, and solutions of the system to the arguments of the function definition. After code generation, the SEG stores elsewhere the generated definition as either a character string to be further processed, or stores it into a C++ header file which is buildable for a PE system simulator platform.

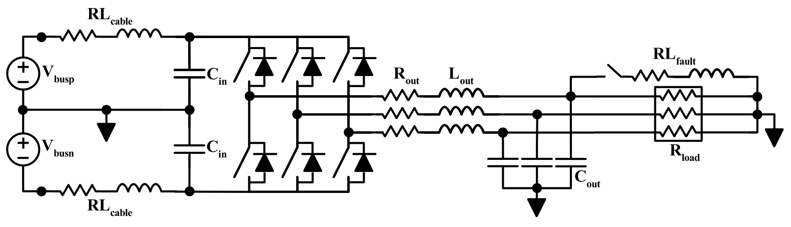
2.5. Netlist loader and component generator factory

As many users prefer to define PE system models as plain-text netlist files rather than write C++ code, the tools provide classes to parse netlists and generate solvers from these files; see Section 3 for example netlist. The netlist files are handled with the *NetlistLoader* and *Netlist* classes. *NetlistLoader* instances provide methods to read netlist files and return *Netlist* objects storing component listings of a PE system netlist. Each of these listing elements in a *Netlist* declares a component, including its type, model parameters, and node connections in a system.

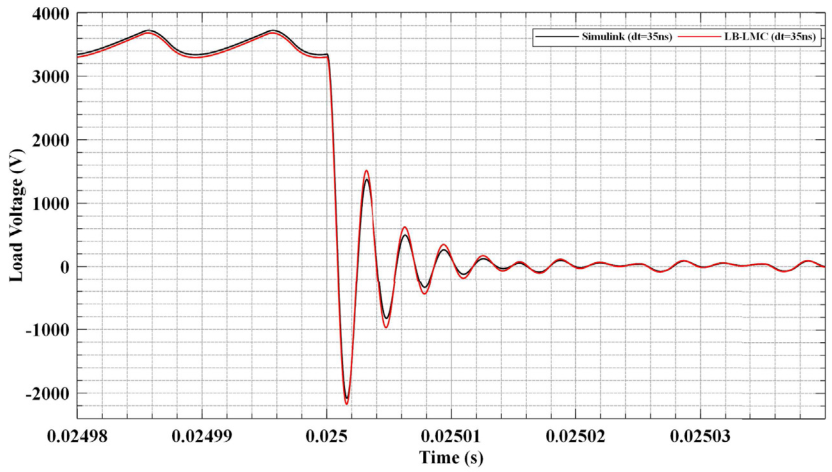
To instance the *ComponentGenerator* objects from the netlists, a set of producer and factory classes are included in the solver codegen tools. Instances of the *ComponentGeneratorFactory* class take listings from a *Netlist* and return a *ComponentGenerator* object via scoped pointer for each listing. The *ComponentGenerator* object is instanced by a *ComponentGeneratorProducer* object that is registered to a specific component type in the *ComponentGeneratorFactory*. Once the *ComponentGenerator* objects are created, they are systematically callable to generate their code and pass it along to a SEG to generate the system solver function definition. The CLI tool provided with the ORTiS solver codegen tools utilize the netlist and factory classes to perform its operation.

2.6. Decomposed system solver generation for multi-FPGA execution

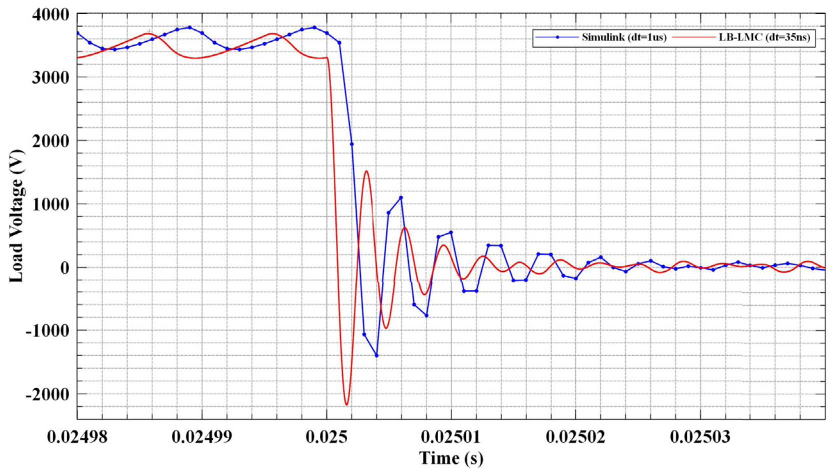
For large scale PE system models that need to be solved across multiple FPGAs, these models require decomposition into subsystems with associated solvers. These subsystem solvers are generated by instances of the *SubsystemSolverEngineGenerator* (SSEG) class, derived from the SEG class. These SSEG objects utilize the nodal decomposition extension to LB-LMC method, fully detailed in [5], to generate solvers for systems decomposed into subsystems. To create the subsystem solvers, the given netlist of a PE system is manually decomposed into subsystems across one or more 2-terminal network ports, with a netlist defined for each subsystem. Then for each subsystem netlist, *ComponentGenerator* objects are created for each component listing. These objects pass their code and other information to an associated SSEG object along with tuples indicating the decomposition port nodes and their port index. From the given data, the SSEG objects compute port models and exchange them between each other to complete their subsystem models and generated solvers. The resultant solver code can then be stored as strings or files to be utilized in multi-FPGA simulator platforms. Each subsystem solver function is similar to a monolithic solver created by SEG, but also has arguments to exchange source contributions between subsystem solvers to ensure subsystems are coupled during simulation to maintain monolithic solutions; see [5] for details.



(a)



(b)



(c)

Fig. 6. Comparison model and simulation results; (a) tested system; (b) comparison between LB-LMC and Simulink at 35 ns time step; (c) comparison between 35 ns and 1 μs time steps.

2.7. High level synthesis for FPGA execution

The general flow to create PE system solvers and deploy them on FPGA platforms with the ORTiS solver codegen tools is presented in Fig. 3. First, the PE system is defined as a netlist of its components. Then, this netlist is passed to the codegen tools (CLI tool or application utilizing solver codegen library) which generates a custom solver function definition based on the netlist. Next, the generated solver definition files are passed to FPGA C++ HLS tools, such as Xilinx Vivado HLx suite, to be translated to Hardware Description Language (HDL) definition which can be implemented on a FPGA target as an RT simulation core. Directives can be passed to the HLS tool so that the solver core is implemented with specified timing, latency, initiation interval, and resource usage. Alternatively, the solver C++ definition can be passed to a traditional C++compiler for implementation as a RT software solver to execute on CPUs.

3. Illustrative examples

An example usage of the ORTiS solver codegen tools is to generate a RT simulation solver for a 40 MW, 12 kV dual split-bus shipboard PE system depicted in Fig. 4(a) and presented in [6,7]. This system utilizes DC/AC converter components and resistive loads depicted in Fig. 4(b), all operating with 100 kHz switching frequency. The system model is discretized with 50 ns time step. Each component in the system is modeled in state space, applying switching functions for converter components, and has a corresponding code generator in the ORTiS tools. The C++solver of this system was generated with the codegen tools, HLS into a FPGA core using Xilinx Vivado HLx suite, and then RT executed at 50 ns time step on a Xilinx VC707 Virtex-7 FPGA kit, using a Texas Instruments DAC34H84EVM for analog output of RT results. RT simulation solutions during closed-loop controlled load voltage reduction of a PE converter in the system are depicted in Fig. 4(c) and 4(d). Magnification of 100 kHz switching ripple of these load voltages before and after change are shown in Fig. 4(e) and 4(f).

A portion of the ORTiS codegen netlist for the shipboard PE system centered around a DC/AC converter is presented in Fig. 5; the full netlist is provided as an example with the codegen tools [8]. In the netlist, the name of the system model is defined with *#name* command and parameter constants are defined with the *#const* command. Each component in the netlist is defined by a line starting with type of the component, followed by its label, parameters in parentheses, and node connection indices in curly brackets. A node index of 0 indicates connection to system common (ground). Comments are added to the netlist via a line starting with % character.

To highlight LB-LMC solver accuracy and the need for small time steps, the standalone 12 kV DC/DC converter of Fig. 6(a) was simulated under LB-LMC solver generated by the ORTiS codegen tools and again under the commercial simulator Mathworks Simulink. Under each simulator, the converter ran at 100 kHz switching frequency with 3500 V load output, using 35 ns time steps, and undergoing a fault scenario where a load is shorted to common by an inductive/resistive path. Fig. 6(b) presents the load voltage during the fault, showing the LB-LMC solver nearly matches the commercial simulator, indicating accuracy of the LB-LMC solver under this scenario. Within Simulink, the model was also simulated under 1 μs time step for same scenario, with load voltage plotted in Fig. 6(c) and compared to 35 ns time step result from LB-LMC solver. As seen in the figure, the higher frequency elements of the fault transients become distorted at larger time steps with lost fidelity and time shifts, introducing inaccuracies during simulation. These inaccuracies from larger steps, even at 1 μs, can adversely impact RT analysis and HIL testing of PE systems with high-frequency dynamics.

4. Impact

The ORTiS solver codegen tools provide a convenient and open source means to develop and deploy RT FPGA-based simulations of PE systems. Using these tools, engineers and researchers can quickly create FPGA-based switching PE system simulation solvers that can achieve 35 ns size time steps, enabling for RT/HIL testing of new PE systems with ever increasing high-frequency dynamics. In [5–7,9] we demonstrated how these solvers can be used for the RT/HIL analysis of microgrids and shipboard power systems with 50–100 kHz switching PE converters utilizing various Xilinx FPGA kits. Since the development of the ORTiS solver codegen tools, another research group has adopted these tools to analyze the development of DC PE systems fault protection with high time resolution of fault transients not possible on commercial RT simulators [10,11]. Thanks to the collaboration with this group we also demonstrated how the code generated solvers can be executed on other FPGA targets such as National Instruments platforms [10–12]. The most natural audience for the tools are research groups that focus their attention on PE system design and who have simulation time resolution needs not satisfied by existing commercial RT simulators. The open source nature of these codegen tools can also be attractive to educational groups, or to those that cannot afford the cost of commercial RT/HIL simulators.

5. Conclusions

In this paper, the ORTiS Solver codegen tools and their functionality have been presented. These tools were developed to enable PE engineers to perform high-fidelity RT-HIL testing of fast and complex PE systems that are arising in the modern times. By releasing these tools as open source, it is hoped that a community forms that can both expand the tools' functionality and spread its usage throughout academia and industry, furthering the progression of PE system development and testing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported in part by the Office of Naval Research (ONR), USA under Grant N00014-16-1-3042.

References

[1] Lin N, Dinavahi V. Detailed device-level electrothermal modeling of the proactive hybrid HVDC breaker for real-time hardware-in-the-loop simulation of DC grids. IEEE Trans Power Electron 2018;33:1118–34.
[2] Milton M, Benigni A. Latency insertion method based real-time simulation of power electronic systems. IEEE Trans Power Electron 2018;33:7166–77.
[3] Ould-Bachir T, Blanchette HF, Al-Haddad K. A network tearing technique for FPGA-based real-time simulation of power converters. IEEE Trans Ind Electron 2015;62:3409–18.
[4] Benigni A, Monti A. A parallel approach to real-time simulation of power electronic systems. IEEE Trans Power Electron 2015;30(9):5192–206.
[5] Milton M, Benigni A, Monti A. Real-time multi-FPGA simulation of energy conversion systems. IEEE Trans Energy Convers 2019;34(4):2198–208.
[6] Milton M, Benigni A, Bakos J. System-level, FPGA-based, real-time simulation of ship power systems. IEEE Trans Energy Convers 2017;32(2):737–47.
[7] Difronzo M, Milton M, Davidson M, Benigni A. Hardware–in–the–loop testing of high switching frequency power electronics converters. In: 2017 IEEE electric ship technologies symposium. Arlington VA: 2017. p. 299–04.
[8] Open real-time simulation (ORTiS) framework. 2020, https://github.com/OpenRealTimeSimulation. [online, Accessed 27 October 2020].
[9] Milton M, Benigni A. Software and synthesis development libraries for power electronic system real-time simulation. In: 2019 IEEE electric ship technologies symposium. Arlington VA: 2019. p. 368–76.
[10] Vygoder M, Milton M, Gudex J, Cuzner R, Benigni A. A hardware-in-the-loop platform for DC protection. IEEE J Emerg Sel Top Power Electron 2020. Early Access.
[11] Vygoder M, Gudex J, Cuzner R, Milton M, Benigni A. Real time simulation of transient overvoltage and common-mode during line-to-ground fault in DC ungrounded systems. In: IECON 2019–45th annual conference of the IEEE industrial electronics society. Lisbon, Portugal: 2019. p. 6451–56.
[12] Milton M, Vygoder M, Gudex J, Cuzner R, Benigni A. Power electronic system real-time simulation on national instruments FPGA platforms. In: 2019 IEEE electric ship technologies symposium. Arlington VA: 2019. p. 32–8.