Enhancing Large-Scale Power Systems Simulations through Functional Mock-up Unit-based Grid-Forming Inverter Models

Sagnik Basumallik[1] Luigi Vanfretti[2] Mohammad Ali Dashtaki[3] Ziang Zhang[3] Reza Pourramezan[1] Hossein Hooshyar[1]

[1]New York Power Authority, {sagnik.basumallik, reza.pourramezan, hossein.hooshyar}@nypa.gov
[2]Rensselaer Polytechnic Institute, vanfrl@rpi.edu
[3]Binghamton University, {mdashta1,zhangzia}@binghamton.edu

Abstract

New York State (NYS) faces significant challenges in meeting the Climate Act's bold goals of 70% renewable energy generation by 2030 and total decarbonization of the electric grid by 2040. Extensive simulations are required to assess the impact of numerous inverter-based resources (IBRs) deployed to the large-scale NYS power grid, aiming to evaluate their dynamic behavior and mitigate any negative interactions with their control schemes. However, the modeling efforts required are huge and the computational burden of large-scale simulations is extensive, and often limited by the capabilities of domain-specific tools. This work addresses these limitations by developing a Functional Mock-up Unit (FMU) of Grid-Forming (GFM) Inverters for IBR control and integrating them with an electromechanical phasor-domain power system solver. The proposed FMU facilitates the simulation and parametric studies needed to analyze large-scale IBR usage with significantly improved manual modeling and computational efforts. The paper details the process of developing and FMU model for GFM IBRs, including all relevant control loops implemented in the Modelica language and FMU integrated in OPAL-RT's ePHASORSIM^{TM} software. Our FMU models are used to successfully deploy and study the impacts of up to $6,200+$ MVA from IBRs on the 5000-bus NYS transmission system.

Keywords: Power Grid, Power Systems, Droop-based Control, Grid Forming Inverter, Inverter-based Resource, Modelica, Functional Mock-up Unit

1 Introduction

In accordance with the NYS Climate Leadership and Community Protection Act, 70% of New York State's electricity must be derived from renewable energy systems (RES) by 2030 (Senate 2023). Grid-forming (GFM) inverter-based technology stands out as a promising solution to integrate RES that are interfaced with the NYS grid via power electronics-based inverters, including wind, PV solar, and fuel cells. This is due to the ability of GFM control to independently provide voltage and frequency support to the grid with fault ride-through capabilities (Du et al. 2019). Comprehensive simulation studies are essential to evaluate the impacts of IBRs on the future NYS grid. However, modeling and simulating a large power system with GFM IBRs deployed at scale introduces substantial challenges including modeling efforts, interfacing resources, and computational requirements.

To address these challenges, this paper presents the development of a complete IBR model with a droop-based GFM control as a Functional Mock-up Unit (FMU). An FMU is a prepackaged, standardized container of user-defined models that adheres to the Functional Mock-up Interface (FMI) standard. The FMI standard is a free tool-independent standard that defines *"a container and an interface to exchange dynamic simulation models using a combination of* `.xml` *files, binaries and C code" (Modelica Association 2025).*

The main advantage of developing IBR models in Modelica is that the models can be re-utilized in different simulation platforms by exporting them as an FMU. This provides end-users with full control over model parameters and IBR control modes, making their reuse in simulation highly scalable and efficient. In addition, FMUs enhances interoperability between different simulation tools (Luigi Vanfretti, Li, et al. 2013). Common industry tools such as PSS/E, PSCAD, EMTP and PowerFactory have different ways of defining models and data, making it difficult to share them between tools (Laera et al. 2022). In contrast, using FMUs offers a *"build once, deploy anywhere"* advantage, allowing models to be easily shared and reused across different simulation platforms without the need for extensive reconfiguration. This minimizes dependencies to the C compiler and the Modelica tool (Luigi Vanfretti, Laughman, and Chakrabarty 2024). Some of the power system simulation tools that currently support FMUs include ePHASORSIM, EMTP, Pandapower, and Powerfactory. Users can also create custom models and interface them as FMUs when existing native software libraries do not offer the required models. Flexibility is key when performing large-scale power system simulation and assessing the impact of integrating IBRs, as required in NYS grid studies, and FMUs offer such advantage.

The FMU developed for the GFM IBR used in this paper is implemented in the Modelica language. Modelica is

an open-access standard and a free, object-oriented mathematical modeling language that is generally used to describe large-scale physical systems (Fritzson and Engelson 1998) and build control applications that interact with their environment (Thiele et al. 2017; Hellerer, Bellmann, and Schlegel 2014). Modelica supports component-based modeling and allows one to mathematically describe a model using differential, algebraic, and discrete equations. In addition, the Modelica Standard Library provides a plethora of components that can be reused, for example, as done in this paper to build several control loops of the GFM IBR. Using object-oriented modeling constructs, such as inheritance (Fachini, Bhattacharjee, et al. 2023), the GFM inverter model is created hierarchically. Beginning with a simple voltage source, we progressively build a controllable voltage source that is the fundamental backbone of the GFM inverter model. The generated FMU is set up to be independently tested and verified inside a Modelica-compliant tool, Dymola (Brück et al. 2002), prior to external deployment. This is one of the main advantages of using the Modelica language and Dymola, which allows to test each of the individual component of the model in isolation, a feature currently not supported by power system simulation software (Laera et al. 2022).

1.1 Literature Review

The Modelica language has been widely used for various power system applications. One notable development is the Open Instance Power System Library (OpenIPSL) (L. Vanfretti et al. 2016; Baudette et al. 2018; de Castro et al. 2023) which has power system component models written in the Modelica language for power system dynamic studies, such as phasor time-domain simulations, while allowing one to extend the modeling scope of conventional power system simulators. The authors in (Luigi Vanfretti, Mukherjee, et al. 2019; Gomez et al. 2018) develop an automatic re-synchronization controller for islanded networks within a multi-domain gas turbine and power system model using Modelica. Specifically, they used Modelica libraries such as ThermoPower (Casella and Leva 2005) to model the thermomechanical dynamics of the gas turbine and OpenIPSL for the components of the power system. The authors in (Mukherjee and Luigi Vanfretti 2018) implemented a frequency controller in Modelica for island operation in power distribution networks. This work integrated multiple Modelica blocks that include a synchronous generator, a gas turbine model, and an excitation system, Modelica noise library for stochastic load modeling and zero-order hold, and `fixedDelay` blocks for PMU reporting rates and delays.

Modelica breaks through the limits of conventional power system simulators. It allows one to conduct specialized studies that usually need various tools or separate programs for different analyses. This is beneficial, for example, in studies related to the stability of island power systems, such as those presented in (Winkler 2018). Here, different models of an island power are connected through a single transmission line to the Icelandic power grid, and their stability is analyzed. The entire transmission and generator models were built in Modelica using the OpenIPSL library, and multiple scenarios are studied for cases where connection to the national grid is lost. Another example in (Segerstrom et al. 2023) develops a Modelica-based approach to analyze subsynchronous oscillations for a lumped mass torsional shaft model with 25 masses. The Modelica implementation was shown to have two unique advantages. In addition to including shaft torsional dynamics, turbine, boiler, and governor dynamics were taken into account using components from OpenIPSL. In addition, the authors emphasize that the `Modelica_LinearSystems2` library (Baur, Otter, and Thiele 2009) allows the analysis of modal-based eigenvalues to be performed directly, eliminating the need to develop a separate model for linear analysis or the use of additional tools for this purpose (Nikolaev et al. 2020).

Microgrid modeling and simulation often exclude phasor simulations, yet they offer benefits by being computationally efficient, as they simplify the switching behavior of power electronics components. For example, the authors of (Fachini, Bhattacharjee, et al. 2023; Fachini, Pigott, et al. 2023; Fachini, Bogodorova, et al. 2024) have implemented a microgrid model using Modelica standard libraries and the OpenIPSL library (Fachini, Bhattacharjee, et al. 2023; Fachini, Pigott, et al. 2023). In (Fachini, Bhattacharjee, et al. 2023), the microgrid was developed with two combustion turbo generators and four steam turbo generators using OpenIPSL machine, prime mover, and control system models, and analyses were carried out to study contingency and modal analysis. This study was extended in (Fachini, Pigott, et al. 2023), where distributed energy resources such as PV were integrated, and fault studies were carried out. These microgrid models were also extended to perform Model Predictive Control (MPC) in (Fachini, Bogodorova, et al. 2024), where the linearized model was used to obtain the MPC solution, and time-domain simulations were performed to apply the MPC actions for safe island operation and re-synchronization.

Developing power system models with Modelica provides additional benefits brought through the FMI ecosystem of tools for new and emerging needs. An example is the `ModelicaGridData` tool which was developed and interfaced with OpenModelica and/or Dymola through their respective Python APIs to generate large data sets of different operating conditions and disturbances, which were then used to train machine learning-based stability assessment approaches (Dorado-Rojas et al. 2023). Another example in (Castro et al. 2022) illustrates how power system models built using the OpenIPSL library can be exported as FMUs to be tested on real-time simulators such as dSPACE. Other power system applications using Modelica include control coordination for HVAC/HVDC power systems (Bakhos et al. 2017; Babaeifar, Barsali, and Ceraolo 2023), implementing and validating variable-speed drive-based induction mo-

DOI 10.3384/ecp218713

tor models (Fachini, Castro, et al. 2024), building energy simulation (Luigi Vanfretti, Laughman, and Chakrabarty 2024), and special protection schemes (Jakobsen et al. 2022).

Beyond power systems, Modelica has proven effective and extensively used in other domains such as thermofluid systems (Steinmann, Herold, and Schirmer 2024), aerospace (Reiner 2022), robotics and mechatronics (Reiser and Reiner 2023), and other industrial applications (Weber, Cartignij, and Zimmer 2023).

1.2 Contributions and Paper Organization

The three specific requirements of this study include: (a) the development of GFM inverter models suitable for large-scale power system simulation, (b) parametrization of GFM inverters, and (c) minimization of both manual and computational effort required for deployment and parameterization. This paper makes the following novel contributions to address the requirements:

1. Provides a guideline for modeling droop-controlled GFM IBRs using Modelica.

2. Develops a systematic approach to validate various subsystems of the GFM IBR model inside Modelica prior to full-scale deployment.

3. Develop interfaces to generate FMUs for the GFM inverter for integration with a specific power system simulator.

4. Discusses the benefits of using FMUs for large-scale power system simulation, such as 'easy-to-build' and 'easy-to-manage' advantages that involve minimal manual intervention and improved computational performance.

5. Examining scaling up of GFM IBR deployment via FMU across the New York State 5000-bus power grid.

The remainder of this paper is organized as follows: Section 2 presents the overview of the Modelica GFM library, Section 3 discusses the development of the GFM inverter model in Dymola, followed by FMU generation in Section 4 and interfacing with OPAL-RT's ePHASORSIM in Section 5. Results from simulation studies are discussed in Section 6 followed by the conclusions from this work in Section 7.

2 Modelica GFM Library Overview

This section presents the GFM IBR library in detail, including various dependencies, packages, and subpackages. In addition, it describes how the object-oriented approach is utilized to construct the GFM IBR model.

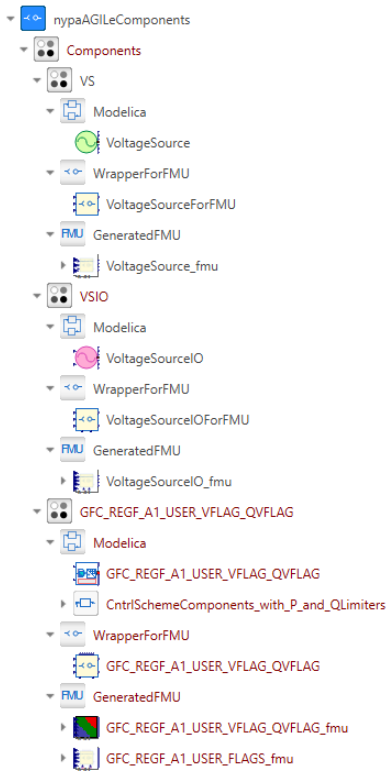

Figure 1. GFM library organization in Dymola

2.1 Organization of GFM Inverter Library

The GFM inverter components are built in the widely popular and proprietary Modelica-compliant language tool called Dymola (Brück et al. 2002). Dymola offers an intuitive graphical user interface with features that allow users to easily navigate built-in libraries and organize their own user-defined models. The GFM IBR library, called `nypaAGILeComponents`, needs to be combined with other Modelica libraries. These include two of Opal-RT's Modelica libraries, the `Opal_RT` [1], a proprietary library that uses the `Opal_RT` library to build power plant sub-systems called `GenUnit`, the Modelica Standard Library, and OpenIPSL. A snapshot of the `nypaAGILeComponents` library where the GFM inverter is built is shown in Figure 1.

There are three main components: (a) voltage source (VS), (b) controllable voltage source with input/output (VSIO), and (c) GFM control model REGFM_A1 (Du 2023). The library is organized reflecting the incremental manner in which components were developed and added. Each component has (a) the Modelica model, (b) an FMU wrapper, and (c) the generated FMU. The wrapper is a Modelica block that encapsulates the Modelica model and causalizes it when the FMU is generated. It contains the mandatory connectors, interfaces and parameters required by ePHASORSIM (described later in Section 4). In addition, each specific wrapper includes an additional set

[1] Available online: `https://github.com/Opal-RT-Technologies/modelica-ephasor-components`

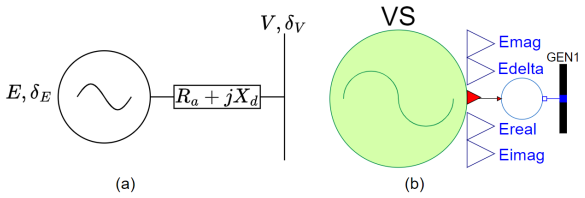

Figure 2. (a) Simple voltage source and (b) Modelica implementation

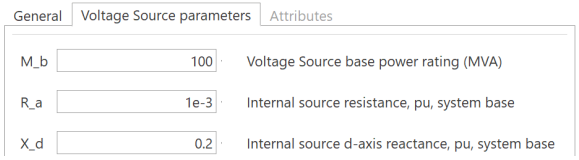

Figure 3. Simple voltage source parameters in Dymola

of parameters that the user can modify using the external tools, e.g., to select the control mode of the GFM inverter and parameterize it. The 'generated FMU' is a Modelica model that contains within the generated FMU the appropriate interfaces for use within the Modelica tool.

2.2 Model Building through Inheritance

Using inheritance, hierarchical and reusable component models were created for GFM modeling in Dymola. Each new component (such as the IBR model) inherits properties and behaviors from existing ones (such as the voltage source). This approach simplifies model development and maintenance by allowing modifications at higher levels of the hierarchy to propagate automatically to derived models. For example, the voltage source (VS) inside the GFM IBR extends from a "VS Droop model" that provides the required "measurement" outputs for the GFM controls. This component itself extends a simple VS whose voltage can be specified by external inputs, are applied to an internal impedance and couples with an acasual connector. An example of such inheritance is given through the statement: `extends VSIO.Modelica.VoltageSourceIO VS();`.

3 Droop-based GFM Components

The GFM inverter operates as a controllable VS behind a coupling reactance (Du et al. 2019). To understand the GFM inverter modeling in Dymola, we start with the design and validation of a simple VS, which forms the basis for the controllable VS that is needed for the GFM model. The controllable VS is followed by two droop-based control architectures (P-f and Q-V), overload limiters and a fault current limiter, as described below.

3.1 Simple Voltage Source

Consider a simple VS with an internal impedance of $R_a + jX_d$ connected to a bus shown in Figure 2. The objective is to find the internal voltage $E\angle\delta_E$ of the source from the active and reactive power (P and Q) measurements at the Gen1 terminal bus. The terminal voltage is denoted as $V\angle\delta_V = V_r + jV_i$. The current equation is given as:

$$I^* = \frac{P+jQ}{V_r+jV_i} = \frac{PV_r - jPVi + jQV_r + QV_i}{V_r^2+V_i^2} \tag{1}$$

The real part of the current is:

$$\Re\{I^*\} = I_r = \frac{PV_r + QV_i}{V_r^2+V_i^2} \tag{2}$$

The imaginary part of the current is:

$$\Im\{I^*\} = I_i = \frac{PVi - jQV}{V_r^2+V_i^2} \tag{3}$$

With internal source impedance of $R_a + jX_d$, the internal voltage ($E = E_{mag}\angle E_\delta$) is given as:

$$E = (I_r + jI_i)(R_a + jX_d) + (V_r + jV_i) \tag{4}$$

The real part of the internal voltage is:

$$\Re\{E\} = E_{real} = V_r - I_iX_d + I_rR_a \tag{5}$$

The imaginary part of the internal voltage is:

$$\Im\{E\} = E_{imag} = V_i + I_iR_a + I_rX_d \tag{6}$$

An excerpt of the VS model with parameters such as base and internal impedance is shown in Figure 3. The VS is implemented in Modelica, as shown in Listing 1. The terminal voltage values are interfaced with the mandatory (required by ePHASORSIM) connector of the `Opal_RT` library, `Powerpin (PwPin)`, through `p.vr` and `p.vi`. Details on PwPin and interfacing can be found later in Section 4.4.

3.1.1 VS Parameter Initialization

Using the VS parameters shown in Figure 4, the component is initialized as shown in the Listing 2. Note that we explicitly define variables with `start` values for initialization not only to provide a good starting guess value for the numerical solvers when simulating from different starting steady-state operating conditions, but also because these values need to be modified when the model is exported as an FMU to be used in a power system simulator.

Listing 1. Snippet of the Internal Voltage Source Equations

```
// Change from system (SB) to machine base (M_b)
parameter Real CoB=M_b/SB;
// Internal voltage source equations
Er = p.vr + CoB*R_a*p.ir - CoB*X_d*p.ii;
Ei = p.vi + CoB*R_a*p.ii + CoB*p.ir*X_d;
// Assing variables to outputs
Emag = E;
Edelta = delta;
Ereal = Er;
Eimag = Ei;
```

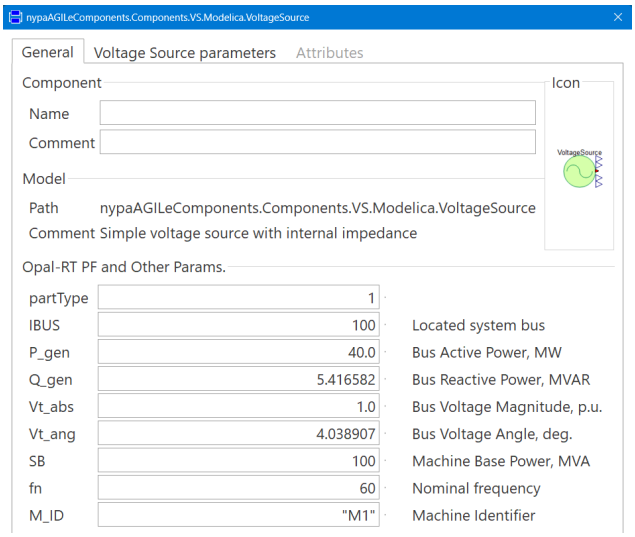

Figure 4. Parameters obtained from power flow used for voltage source initialization

Listing 2. Initialization of internal VS from power flow

```
protected
// Auxiliary parameters for initialization
v_0 = Vt_abs "Intial terminal Vmag from pf";
angle_0 =  Modelica.Units.Conversions.from_deg(
    Vt_ang) "Initial terminal V_angle from pf";
P_0 = P_gen "o/p power from pf";
Q_0 = Q_gen "o/p power from pf";

// Terminal Voltage variables
Real V(start=v_0) "Bus voltage magnitude";
Real anglev(start=angle_0) "Bus voltage angle";

// Public Auxiliary variables
Real P(start=P_gen/SB) "Active power";
Real Q(start=Q_gen/SB) "Reactive power";

// Public Internal voltage source variables
Real delta(start=delta0) "Internal VS angle";
Real E(start=E0) "Internal VS magnitude";

// Change of base
CoB=M_b/SB;
p0=P_0/M_b "Initial MW (machine base)";
q0=Q_0/M_b "Initial MVAR (machine base)";

// Initialization values
vr0=v_0*cos(angle_0);
vi0=v_0*sin(angle_0);
ir0=CoB*(p0*vr0 + q0*vi0)/(vr0^2 + vi0^2);
ii0=CoB*(p0*vi0 - q0*vr0)/(vr0^2 + vi0^2);

// Initialization of internal VS parameters
Er0 = vr0 + CoB*R_a*ir0 - CoB*X_d*ii0 "Initial
    value of E_real";
Ei0 = vi0 + CoB*R_a*ii0 + CoB*X_d*ir0 "Initial
    value of E_imag";
E0  = sqrt(Er0^2+Ei0^2);
delta0 = atan2(Ei0, Er0);

// Internal voltage source variables
Real Er(start=Er0);
Real Ei(start=Ei0);

equation
// Internal voltage source equations
delta = delta0 "assume Constant voltage angle";
E = E0 "assume Constant voltage magnitude";
Er = Er0;
Ei = Ei0
```

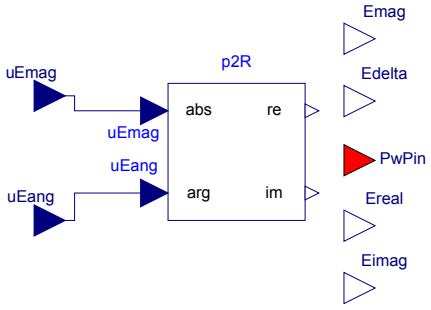

Figure 5. Controllable voltage source with two inputs

3.2 Controllable Voltage Source

The controllable VS is developed in Dymola based on the equations of the simple VS given in Equations (1) - (6). The controllable VS has two inputs `uEmag` and `uEang`, as shown in Figure 5, to vary the voltage magnitude and angle of the VS, respectively. To increase flexibility and facilitate testing, the input to the controllable voltage source can be provided from two sources: (a) Case 1: inject a voltage phasor to define the internal voltage of the source from an external tool (e.g., a power system simulator like ePHASOSRSIM), and (b) Case 2: inject a signal that is a deviation from the initial value of E0 and delta0 for internal testing. These two cases are shown in Listing 3.

Listing 3. Internal equations for controllable VS

```
if Case 1 then "input from external tool"
    E = uEmag;
    delta = uEang;
    Er = p2R.y_re "Real part of E";
    Ei = p2R.y_im "Imaginary part of E";
else "deviation from initial value"
    E = E0 + uEmag;
    delta = delta0;
    Er = Er0 + p2R.y_re;
    Ei = Ei0 + p2R.y_im;
```

3.3 Droop-based GFM Inverter Model

The controllable VS is used as a basic building block to build the GFM IBR model. Similarly to the controllable VS in Figure 5, the GFM inverter model has two inputs: `uEmag` and `uEang`. The input `uEang` that controls the voltage angle is determined by the P-f control loop, and `uEmag` that controls the voltage magnitude is determined by the Q-V control loop. The complete GFM IBR model, along with its control scheme, is shown in Figure 6.

3.3.1 Active Power-Frequency Droop-based Control

The active power-frequency (or P-f droop) droop-based control adjusts the output frequency of the inverter in response to changes in active power output. With $E\angle\delta_E$ as

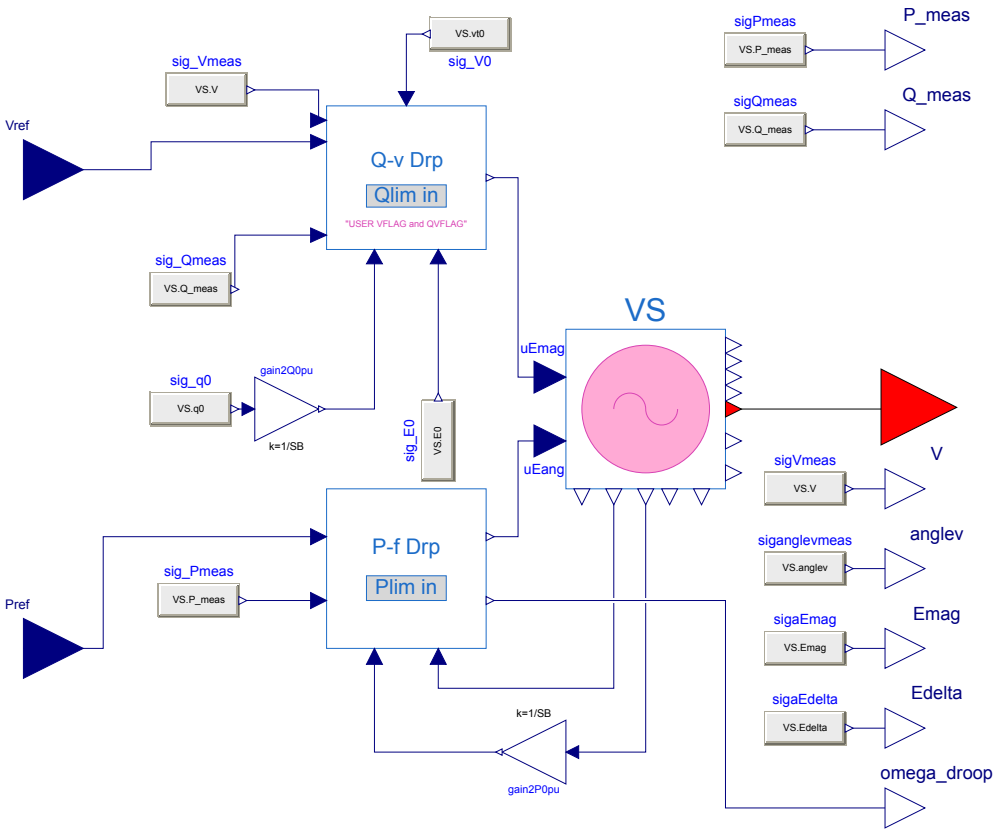

Figure 6. GFM Inverter Dymola Model Block Diagram

the internal voltage phasor and $V\angle\delta_v$ as the terminal voltage phasor, the active power output of the inverter P can be written as (Du 2023),

$$P \approx \frac{EV}{X_L} sin(\delta_E - \delta_v) \approx \frac{EV}{X_L}(\delta_E - \delta_v) \tag{7}$$

When a setpoint change or a disturbance causes the inverter active power to increase, the P-f droop-based control reduces the angular frequency ω of the internal voltage to reduce the inverter power output (Rocabert et al. 2012), which is modeled as a change in δ_E (Du 2023). The P-f droop-based control also ensures that multiple GFM inverters share power relative to their active power capacities under disturbances. The overall P-f droop-based control architecture is shown in 7, and the detailed P-f droop-based control loop is shown in Figure 8. The P-f droop loop takes the error between the reference active power `Pref` and the measured active power `Pfilt` (after passing through a low pass filter) as input, and provides the internal voltage phase angle and angular frequency of the VS as output. There is an auxillary input signal from the output power limiter block that prevents the active power output of the inverter from exceeding its limits. Note that the integrator in the control loop is initialized with the value `delta0` as discussed in Section 3.1.1 (see line 35 in Listing 2).

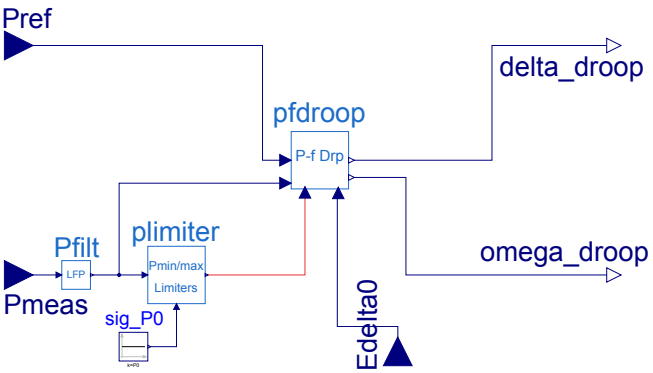

Figure 7. P-f Droop-based Control Architecture

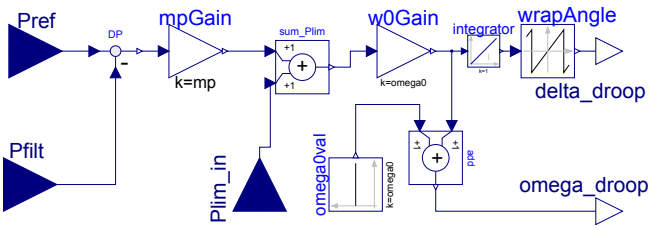

Figure 8. P-f Droop-based Control Function

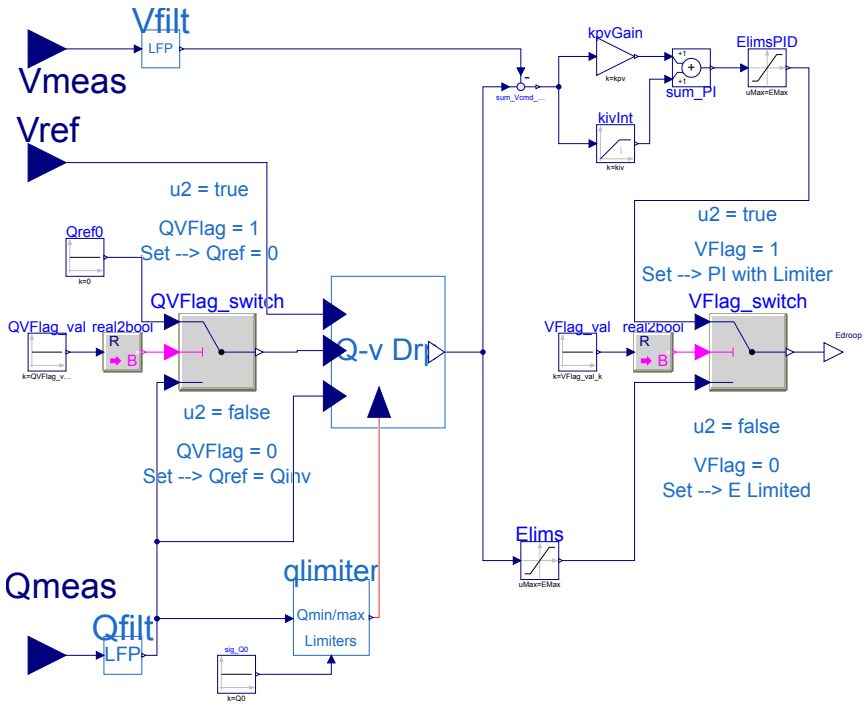

Figure 9. Q-V droop-based control architecture

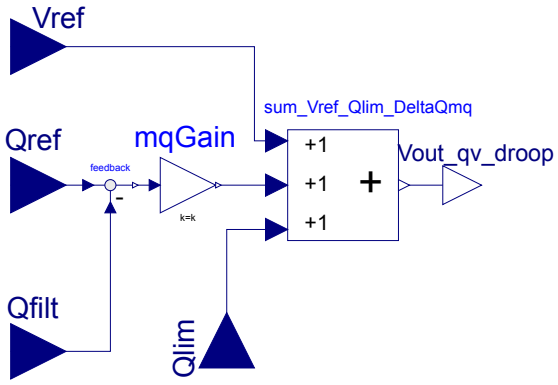

Figure 10. Q-V Droop-based Control Function

3.3.2 Reactive Power-Voltage Droop-based Control

The reactive power-voltage (or the Q-V) droop-based control adjusts the inverter reactive power output in response to changes in the voltage at its terminals. The reactive power output of the inverter can be written as (Du 2023),

$$Q \approx \frac{E^2 - EV\cos(\delta_E - \delta_v)}{X_L} \approx \frac{E(E-V)}{X_L} \tag{8}$$

When there is a step change in voltage setpoint or an event disturbs the terminal voltage, the Q-V droop-based control manipulates the magnitude of the internal voltage of the inverter to adjust the reactive power output accordingly. The Q-V droop has two voltage control modes; it can either regulate: (a) the internal voltage, $E\angle\delta_E$, or (b) terminal voltage, $V\angle\delta_V$, which is determined by the setting of `VFlag`. In addition, the Q-V droop has two reactive power control modes that are set by `QVFlag`, which can be set to represent the plant controller strategy of adjusting either the reactive power or voltage setpoint. When multiple GFM inverters are connected to the grid in parallel, the Q-V droop-based control prevents the circulation of the reactive power.

The input/output and control blocks of the Q-V droop scheme are shown in Figure 9. The Q-V droop model takes as input the reference voltage, the measured reactive power (after passing through the low-pass filter) and two flags, `VFlag` and `QVFlag`, and provides the internal voltage magnitude as output, as shown in Figure 10. In addition, the auxiliary signal from the output power limiter block prevents the reactive power output of the inverter from exceeding the converter's limits. Note that the integrator in the control loop is initialized with the value `v_0` as discussed in Section 3.1.1 (see line 9 in Listing 2).

3.4 Overload Limiters

For both control loops, there are corresponding maximum and minimum MW and MVAR limits to prevent the inverter from exceeding rated capacities. The block diagram for the limiters are shown in Figure 11 and Figure 12.

3.5 Fault Current Limiting Function

When there is a short-circuit and fault currents are very high (exceeding inverter rated current I_{maxF}), the GFM internal voltage $E\angle\delta_E$ will be calculated based on the inverter terminal voltage $V\angle\delta_v$, coupling reactance X_L and current phasor $I_{maxF}\angle\phi_{lim}$, which represents the limited current. The GFM internal voltage under fault conditions is given as:

$$E\angle\delta_E = V\angle\delta_v + jX_L I_{maxF}\angle\phi_{lim}. \tag{9}$$

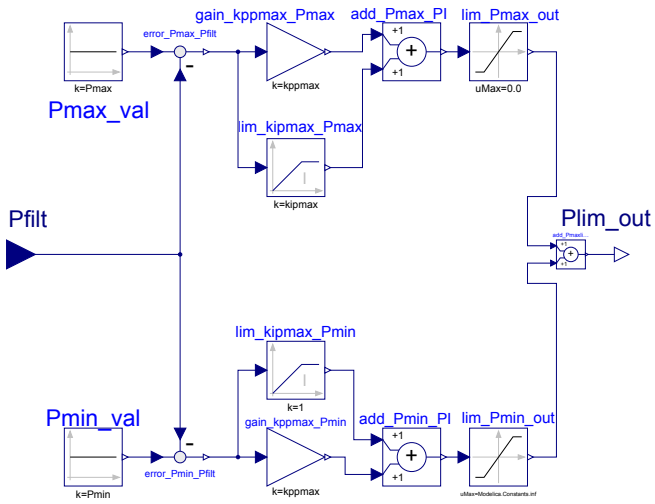

Figure 11. MW Limiter

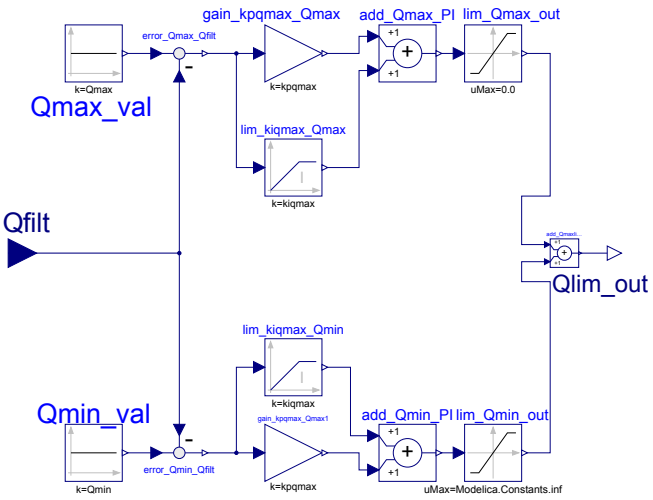

Figure 12. MVAR limiter

Due to space limitations, block diagrams for current limiters are not shown in this paper.

4 FMU Generation

In this section, we briefly discuss how Modelica model is setup to generate an FMU.

4.1 System Dependencies

To generate FMUs using Dymola, compatible with Windows and Linux (required by the ePHASORSIM simulator from Opal-RT), the following dependencies are necessary: Windows 10 or later versions, a suitable compiler for C++ (such as Visual Studio 2019 with the 'Desktop development with C++' workload enabled), and Windows Subsystem for Linux (WSL) 2 with Linux kernel update package.

4.2 Windows Subsystem for Linux (WSL)

The cross-compilation feature available in Dymola allows Dymola to compile models to be used on a Linux target. This is done through WSL, which enables a Linux system to run inside a Windows machine using WSL. FMUs generated with this feature can be executed by software tools native to Windows and Linux-based operating systems (OS). This is because the generated FMU will have specific target binaries for each of those OSs. ePHASORSIM crucially needs this feature since it relies on Windows for offline and Linux for real-time simulations.

4.3 FMU with and without Source Codes

FMUs can be generated with or without the C source code. With Dymola, this capability depends on whether or not the user has a license that enables code export. Regardless, both options are useful, since a user with a local "vanilla" Dymola license could regenerate the FMUs and execute off-line local simulations. Export of source code is only required when running the ePHASORSIM in real time. For FMUs without source code, the FMU package contains only the compiled binaries of the model, not the C source code used to build the binary (.dll or .so). With the source code export, users can recompile the FMU's code, offering greater flexibility.

ePHASORSIM has specific requirements for the FMUs it receives from Dymola, for example, it requires that internal variables of the FMU can be accessed, which is enabled in Dymola with the flag `Advanced.FMI.BlackBoxModelDescription := flase`. In this paper, we create the `FMUExportUtilities` feature that allows to create FMUs both with and without source code with the required flags for import in ePHASORSIM. Figure 13 shows the GUI of the function call used to generate the FMU with/without source code. Listing 4 shows the flags necessary to generate the FMU with source code from Dymola while taking into account the requirement of the availability of the source code generation license for code export. Note that OPAL-RT ePHASORSIM currently supports FMI version 1. While three types of interfaces are defined in FMI standard, we are interested in the FMI for Model Exchange (ME). ME implements only the model and not the required solution. It exposes the differential equations to an external solver that imports the model and performs numerical integration.

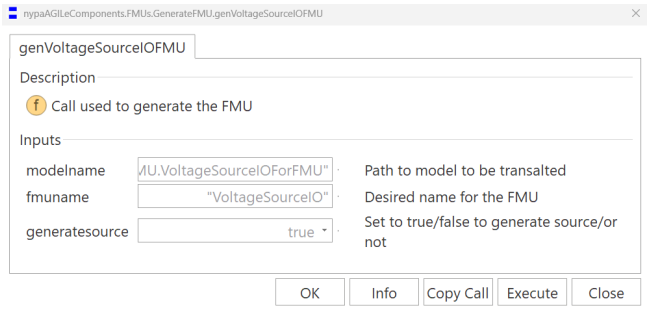

Figure 13. Custom function to generate FMUs with/without source code

Listing 4. Custom function to Generate FMU with Source Code

```
function generateFMUWithSourceCode
```

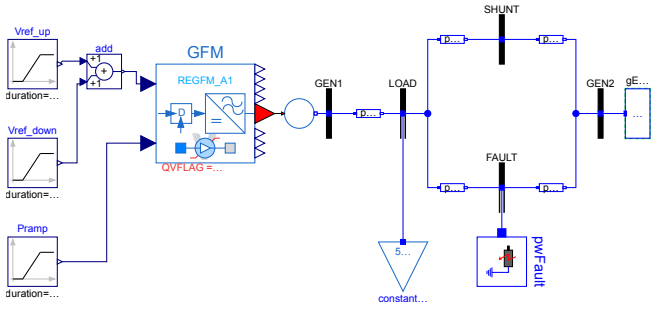

Figure 14. Testing GFM Model within a power system model built using OpenIPSL

```
  "Call used to generate the FMU with source
      code"
  extends Modelica.Icons.Function;
  input String modelname = "IBR_VS_Droop_FMU"
  input String fmuname = "GENSAL";
algorithm
  // Flags used in "..\FMUCreator\dymolaFlag.cfg
      "
  Advanced.CheckPackageRestriction :=false;
  Advanced.FMI.xmlIgnoreProtected :=false;
  Advanced.FMI.xmlIgnoreLocal :=false;
  Advanced.FMI.BlackBoxModelDescription :=false;
  Advanced.Define.AimForHighAccuracy :=false;
  Advanced.Define.NewJacobian :=false;
  Advanced.EnableCodeExport := true;
  Advanced.SourceCodeExportNormal :=true;
  Advanced.FMI.CrossExport :=true;
  Advanced.FMI.FMUIncludeSource :=true;
  Advanced.FMI.FMUSourceCodeUniqueNaming :=true;
  // Create FMU
  translateModelFMU(modelname, false, fmuname, "
      1", "all", true, 1);
end generateFMUWithSourceCode;
```

4.4 Testing of Models and FMUs in Dymola

The models and FMUs generated by Dymola are tested within the Dymola interface before interfacing with ePHASORSIM. In this section, we demonstrate both (a) testing the Modelica model within a power system model and (b) testing the FMU in Dymola within a power system model.

The GFM inverter model is tested within a power system modeled using OpenIPSL. A Single Machine Infinite Bus (SMIB) system with one load is extended from the OpenIPSL library using `extends OpenIPSL.Tests.BaseClasses.SMIB(SysData(fn=60), pwFault(R = 1e-6, X = 1e-3, t1 = 32.0, t2 = 32.15))` with a fault duration of 0.15 s. The setup is shown in Figure 14.

To connect the GFM model with the OpenIPSL SMIB within Dymola, the `Powerpin (PwPin)` connector is used by instantiating it from the `OPAL_RT` class `OpalRT.NonElectrical.Connector.PwPin`. The `PwPin` is required by ePHASORSIM, as when the FMUs are loaded, ePHASORSIM will determine if `PwPin` has been used or not. This is done by the software to force conformance to its internal numerical solver interface. The `PwPin` has two flow variables (`ir` and `ii`) and two potential variables (vr and vi). The implementation of `PwPin` is given in Listing 5.

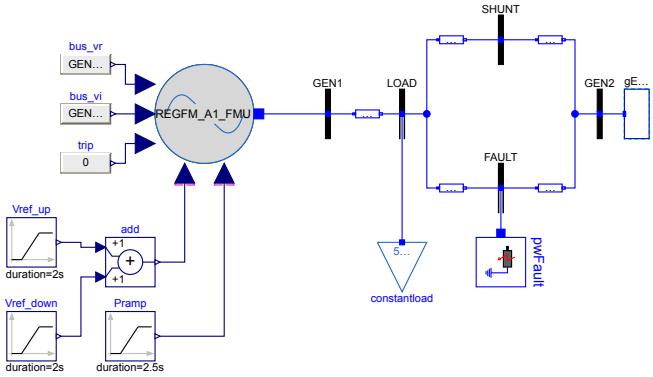

Figure 15. Testing GFM FMU in Dymola within a power system model built using OpenIPSL.

Listing 5. `PwPin` Connector from `Opal_RT`'s library

```
connector PwPin
    Real vr "real part of the voltage";
    Real vi "imaginary part of the voltage";
    flow Real ir "real part of the current";
    flow Real ii "imaginary part of the current"
        ;
end PwPin;
```

To interface the GFM model developed according to ePHASORSIM's requirements (with `PwPin`) with electrical components from the OpenIPSL library inside Dymola, an additional connector is required. In this connector, the variables of `PwPin` (instantiated as `pwPinA`) are assigned to the corresponding variables of the OpenIPSL electrical connector (instantiated as `pwPinB`). The interface between the power pins of both libraries is shown in Listing 6. This setup is used to test all models and submodels inside Dymola.

Listing 6. Code Excerpt of the Connector Interfacing the `Opal_RT` and OpenIPSL Libraries

```
model Opal2OpenIPSL "Interface between power
    pins of both libraries"
equation
  pwPinB.vr = pwPinA.vr;
  pwPinB.vi = pwPinA.vi;
  pwPinB.ir = pwPinA.ir;
  pwPinB.ii = pwPinA.ii;
end Opal2OpenIPSL;
```

The generated FMU model is also tested within a power system modeled in Dymola. A similar setup is shown in Figure 15.

Note that the `nypaAGILecomponents` library was developed and tested with Dymola 2024X and compatibility with other Modelica-compliant tools has not been verified. Meanwhile, the Opal-RT library is compatible with both `OpenModelica` and Dymola, while the `OpenIPSL` is compatible with Dymola, OpenModelica, Modelon Impact, and Wolfram SystemModeler.

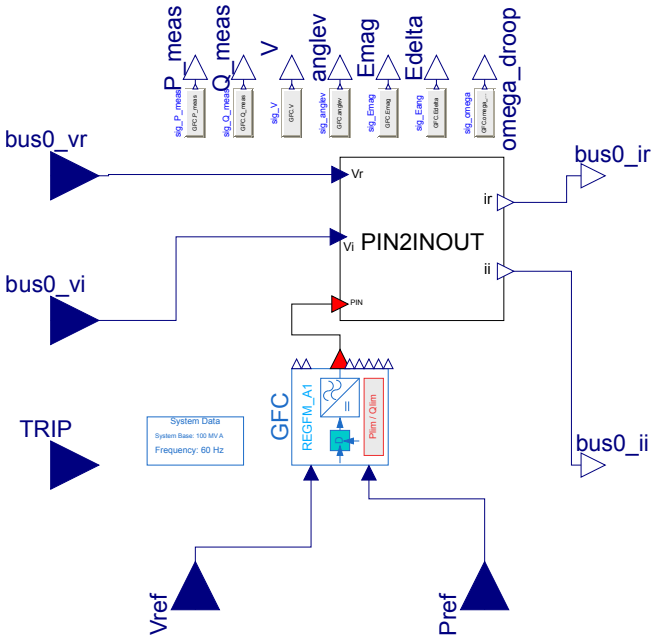

Figure 16. Wrapped GFM Model with Required Interfaces by ePHASORSIM for Export as FMU

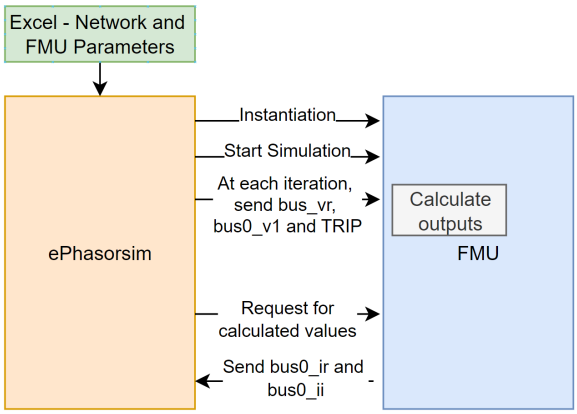

Figure 17. Flowchart of communication between OPAL-RT ePHASORSIM and FMU

5 Interfacing FMUs with ePHASORSIM

5.1 FMU - ePHASORSIM `PIN2INOUT` Interface

To interface the FMU with ePHASORSIM, an FMU wrapper is created. As discussed in Section 2.1, the wrapper includes the mandatory connectors, interfaces, and parameters required by ePHASORSIM and the set of parameters that the user can modify using external tools. The FMU wrapper is shown in Figure 16.

A `PIN2INOUT` block is required by ePHASORSIM to be used as an interface with FMU, which enforces the input (voltage) and generates (current) outputs as expected by ePHASORSIM. OPAL-RT's ePHASORSIM requires the mandatory `PwPin`. The block `PIN2INOUT` converts the acasual `PwPin` variables to casualized signals, defining the voltages as inputs and currents as output with the current flow direction required by ePHASORSIM. Listing 7 an excerpt of the `PIN2INOUT` block source code.

Listing 7. Excerpt of the `PIN2INOUT` block

```
block PIN2INOUT
equation
  p.ir = -ir;
  p.ii = -ii;
  p.vr = vr;
  p.vi = vi;
end PIN2INOUT;
```

5.2 Data Exchange between ePHASORSIM and FMU

Figure 17 shows the data exchange between OPAL-RT ePHASORSIM and the FMU generated from Dymola. In ePHASORSIM, all parameters for a power system model are defined in an Excel file. Once ePHASORSIM reads the network and FMU parameters from the Excel file, it instantiates the FMU and passes the parameter data to it. ePHASORSIM uses a fixed time-step solve, hence, at each simulation step, it sends the terminal bus voltage real and imaginary values, `bus0_vr` and `bus0_vi`, and any inputs defined in the wrapper to the FMU. The FMU equations are evaluated with the received values, and the results are sent back to ePHASORSIM in the form of real and imaginary current injection values `bus0_ir` and `bus0_ii`, and any of the output values included in the FMU wrapper. ePHASORSIM then solves for all variables within its internal models and assembles the overall network solution.

6 Simulation Results

6.1 Controllable Voltage Source

As discussed in Section 4.4, the models are interfaced to the OpenIPSL SMIB for testing within the Dymola environment. Testing within Dymola itself offers the advantage of efficient debugging, validation, and iterative development of models and FMUs without the need for external tools. The setup for testing the controllable VS is shown in Figure 18.

Figure 19 shows the responses of the internal magnitude and angle of the controllable voltage source to the ramp functions within Dymola. For `Emag`, the ramp was activated at 4 s., changing the value from 1.01439 p.u. to 1.03439 p.u. within 1 s. For `Eang`, the ramp was activated at 6s changing the value from 0.149386 rad to −0.243364 rad within 3 s. Meanwhile, Figure 20 shows the response of the active and reactive power, P and Q. It becomes apparent from the plot how a change in the internal voltage source δ_E has a direct impact on P according to Equation (7), while changes in E have a direct impact on Q as per Equation (8).

DOI 10.3384/ecp218713

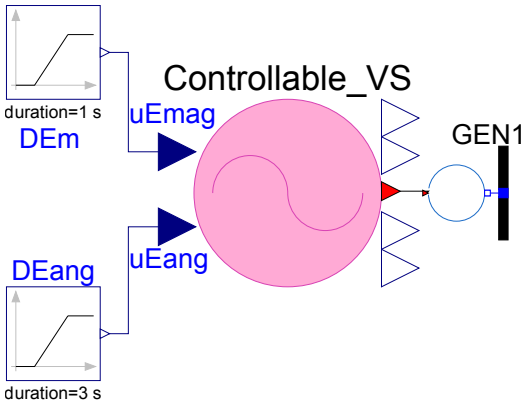

Figure 18. Controllable voltage source connected to a bus with inputs to change voltage magnitude and angle

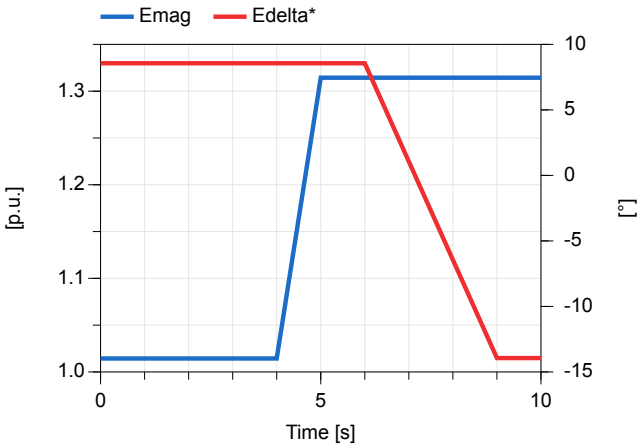

Figure 19. Controllable voltage source internal magnitude and angle ($E \angle \delta_E$) response to ramp function in Dymola

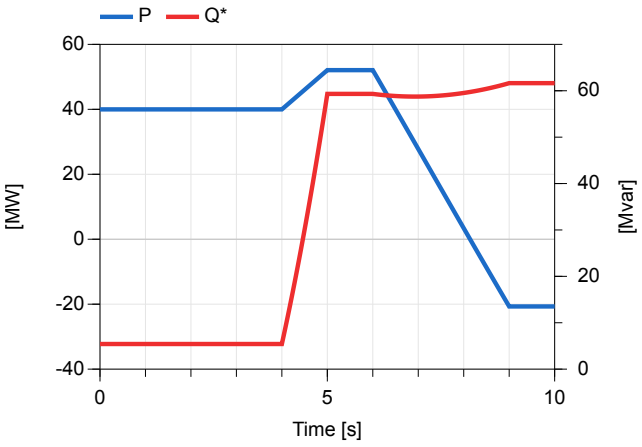

Figure 20. P and Q output of the controllable voltage source due to a ramp applied to its reference inputs in Dymola

6.2 Implementation of GFM inverter FMU on NYS Transmission

GFM IBRs, modeled as FMU, were integrated into the NYS 5000-bus, 800-machine transmission system in ePHASORSIM. In addition to the GFM components, models of synchronous generators (SG), exciters, turbines, wind generators, and power system stabilizers, developed using Modelica language and exported as FMU, were interfaced with ePHASORSIM. A total of 9 GFM IBR with 6,272.5 MVA capacity was configured to substitute traditional generators within the NYS transmission system. The impact on the transient stability was analyzed under different conditions, such as changes in GFM active power and voltage setpoints. For example, Figure 21 shows the response of when the MW set-point on one IBR increased from 187.46 MW to 300 MW. Figure 22 shows the response of the IBRs when the voltage setpoint on target bus was changed from 1.019 p.u. to 1.017 p.u. This demonstrates that the FMU generated from Dymola is successfully integrated with a large-scale power system, highlighting the approach that is easy to deploy, scalable, and computationally efficient.

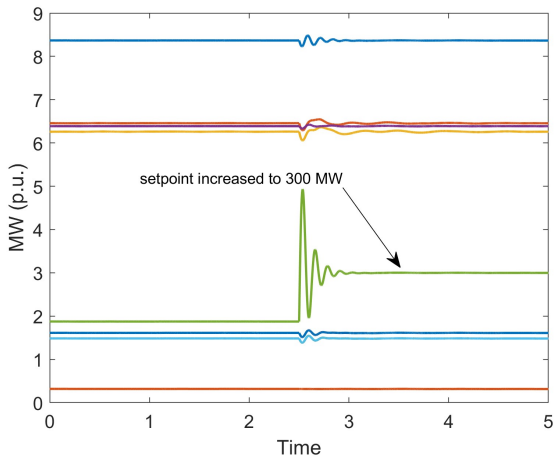

Figure 21. GFM response to changes in MW setpoint in OPAL-RT ePHASORSIM

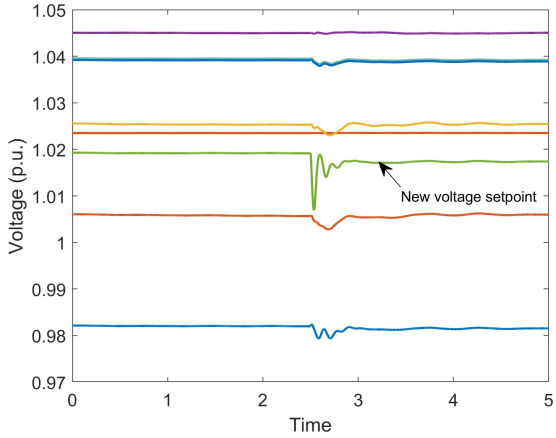

Figure 22. GFM response to changes in voltage setpoint in OPAL-RT ePHASORSIM

Table 1. Comparison between Simulink-Based and FMU-Based GFM IBR Deployment in ePHASORSIM

Aspect	**Simulink-Based Deployment**	**FMU-Based Deployment**
Controller Integration	Manual copy-paste of droop-based GFM controller block each time	Controller is pre-embedded in FMU; no need for repetition
Signal Wiring	Manual setup for each new GFM (all inputs/outputs to/from solver)	Minimal signal mapping (e.g., `Pset`, `Vset`) required
Model Scalability	Tedious and error-prone with more GFMs	Easily scalable to large numbers of GFMs (e.g., hundreds)
Transformer Setup	Details must be manually specified in Transformer sheet	Can be included directly in FMU or in the additional FMU-named sheet
Error Probability	High, especially with manual wiring and naming	Lower, due to fewer steps and more automation
Time Efficiency	Time-consuming due to repetitive steps and complex wiring	Very time-efficient and streamlined for high GFM penetration
Maintenance & Debugging	Difficult to trace and fix due to signal complexity	Easier maintenance

One significant advantage observed during the development process was the ease and efficiency of deploying FMUs with ePHASORSIM compared to using MATLAB/SIMULINK to build and deploy IBRs manually. A detailed comparison of both approaches is given in Table 1. These demonstrate that FMUs are the ideal candidates for efficient and scalable simulation.

7 Conclusion

This paper demonstrates the advantage of developing and deploying FMUs for large-scale power system simulations. A grid-forming inverter model was developed in Modelica with Dymola and integrated as an FMU with OPAL-RT's ePHASORSIM simulator to evaluate the impact of widespread renewable energy integration on the NYS transmission system. The development of the GFM FMU inverter was made easier through the object-oriented modeling features of Modelica. First, a generic voltage source model was implemented which was then extended to create a controllable voltage source and was further extended to develop the GFM-controlled inverter. With a 'build-once deploy-anywhere' approach, this paper shows that FMU offers significant time savings in modeling, debugging, and parametrization compared to other power system simulation approaches. The FMUs were tested inside Dymola and integrated with the NYS 5000−bus system in ePHASORSIM. In future, we will extend our work by (a) building FMU models for the virtual synchronous machine GFM model, (b) exploring FMU interoperability by deploying FMUs on different power system simulation tools and (c) investigating the impact of IBR parameter uncertainties on overall system stability.

References

Babaeifar, Mohammad, Stefano Barsali, and Massimo Ceraolo (2023). "Simulation of Sapei HVDC using modelica language". In: *2023 AEIT HVDC International Conference (AEIT HVDC)*. IEEE, pp. 1–6.

Bakhos, Gianni et al. (2017). "Hybrid AC/DC Power System Stability: An Attempt of Global Approach". In: *Available at SSRN 4704802*.

Baudette, Maxime et al. (2018). "OpenIPSL: Open-instance power system library—update 1.5 to "iTesla power systems library (iPSL): A modelica library for phasor time-domain simulations"". In: *SoftwareX* 7, pp. 34–36.

Baur, Marcus, Martin Otter, and Bernhard Thiele (2009). "Modelica libraries for linear control systems". In: *Proceedings 7th Modelica Conference*. DOI: 1, pp. 593–602.

Brück, Dag et al. (2002). "Dymola for multi-engineering modeling and simulation". In: *Proceedings of modelica*. Vol. 2002. Oberpfaffenhofen, pp. 1–6.

Casella, Francesco and Alberto Leva (2005). "Object-oriented modelling & simulation of power plants with modelica". In: *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE, pp. 7597–7602.

Castro, Marcelo de et al. (2022). "Power System Real-Time Simulation using Modelica and the FMI". In: *Modelica Conferences*, pp. 85–92.

de Castro, Marcelo et al. (2023-02). "Version [OpenIPSL 2.0.0] - [iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations]". In: *SoftwareX* 21, p. 101277. ISSN: 2352-7110. DOI: 10.1016/j.softx.2022.101277. URL: https://www.sciencedirect.com/science/article/pii/S2352711022001959 (visited on 2023-10-17).

Dorado-Rojas, Sergio A et al. (2023). "ModelicaGridData: Massive power system simulation data generation and labeling tool using Modelica and Python". In: *SoftwareX* 21, p. 101258.

Du, Wei (2023). *Model specification of droop-controlled, grid-forming inverters (REGFM_A1)*. Tech. rep. Pacific Northwest National Laboratory (PNNL), Richland, WA (United States).

Du, Wei et al. (2019). "A comparative study of two widely used grid-forming droop controls on microgrid small-signal stability". In: *IEEE Journal of Emerging and Selected Topics in Power Electronics*. Vol. 8. 2. IEEE, pp. 963–975.

Fachini, Fernando, Srijita Bhattacharjee, et al. (2023). "Exploiting Modelica and the OpenIPSL for University Campus Microgrid Model Development". In: *Modelica Conferences*, pp. 285–292.

Fachini, Fernando, Tetiana Bogodorova, et al. (2024). "A microgrid control scheme for islanded operation and re-synchronization utilizing Model Predictive Control". In: *Sustainable Energy, Grids and Networks* 39, p. 101464. ISSN: 2352-4677. DOI: https://doi.org/10.1016/j.segan.2024.101464. URL: https://www.sciencedirect.com/science/article/pii/S2352467724001930.

Fachini, Fernando, Marcelo de Castro, et al. (2024). "Modeling of Induction Motors and Variable Speed Drives for Multi-Domain System Simulations Using Modelica and the OpenIPSL Library". In: *Electronics* 13.9, p. 1614.

Fachini, Fernando, Aisling Pigott, et al. (2023). "Developing a campus microgrid model utilizing modelica and the OpenIPSL library". In: *2023 11th Workshop on Modelling and Simulation of Cyber-Physical Energy Systems (MSCPES)*. IEEE, pp. 1–6.

Fritzson, Peter and Vadim Engelson (1998). "Modelica—A unified object-oriented language for system modeling and simulation". In: *ECOOP'98—object-oriented programming: 12th European conference Brussels, Belgium, July 20–24, 1998 proceedings 12*. Springer, pp. 67–90.

Gomez, Francisco J et al. (2018). "Multi-domain semantic information and physical behavior modeling of power systems and gas turbines expanding the common information model". In: *IEEE access* 6, pp. 72663–72674.

Hellerer, Matthias, Tobias Bellmann, and Florian Schlegel (2014). "The DLR Visualization Library-recent development and applications". In.

Jakobsen, Sigurd Hofsmo et al. (2022). "Modelica-based parallel computing framework for power system adaptive special protection schemes". In: *2022 Open Source Modelling and Simulation of Energy Systems (OSMSES)*. IEEE, pp. 1–6.

Laera, Giuseppe et al. (2022). "Guidelines and Use Cases for Power Systems Dynamic Modeling and Model Verification using Modelica and OpenIPSL". In: *Modelica Conferences*, pp. 146–157.

Modelica Association (2025). *FMI 3.0 Implementers' Guide*. URL: https://modelica.github.io/fmi-guides/main/fmi-guide/ (visited on 2025-04-03).

Mukherjee, Biswarup and Luigi Vanfretti (2018). "Modeling of PMU-based islanded operation controls for power distribution networks using Modelica and openIPSL". In: *Proceedings of The American Modelica Conference*, pp. 9–10.

Nikolaev, Nikolay et al. (2020). "PSS/E Based Power System Stabilizer Tuning Tool". In: *2020 21st International Symposium on Electrical Apparatus & Technologies (SIELA)*. IEEE, pp. 1–6.

Reiner, Matthias J (2022). "Simulation of the on-orbit construction of structural variable modular spacecraft by robots". In: *Modelica Conferences*, pp. 38–46.

Reiser, Robert and Matthias J Reiner (2023). "Modeling and simulation of dynamically constrained objects for limited structurally variable systems in Modelica". In: *Modelica Conferences*, pp. 151–158.

Rocabert, Joan et al. (2012). "Control of power converters in AC microgrids". In: *IEEE transactions on power electronics* 27.11, pp. 4734–4749.

Segerstrom, Eric et al. (2023). "Modeling Components of a Turbine-Generator System for Sub-Synchronous Oscillation Studies with Modelica". In: *Modelica Conferences*, pp. 541–550.

Senate, NY State (2023-05). *N.Y. Pub. Svc. Law § 66-P – Establishment of a renewable energy program*. en. Section: Public Service Law. URL: https://newyork.public.law/laws/n.y._public_service_law_section_66-p (visited on 2025-05-01).

Steinmann, Christoph, Johannes Herold, and Jens Schirmer (2024). "Model-Based Design and Characterization of an Actuator with Low-Boiling Liquid". In: *Modelica Conferences*, pp. 7–14.

Thiele, Bernhard et al. (2017). "Towards a standard-conform, platform-generic and feature-rich Modelica device drivers library". In: *Proceedings of the 12th International Modelica Conference*. Linköping University Electronic Press, 2017, pp. 713–723.

Vanfretti, L. et al. (2016). "iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations". In: *SoftwareX* 5, pp. 84–88. ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2016.05.001. URL: https://www.sciencedirect.com/science/article/pii/S2352711016300097.

Vanfretti, Luigi, Christopher R Laughman, and Ankush Chakrabarty (2024). "Integrating Generative Machine Learning Models and Physics-Based Models for Building Energy Simulation". In: *Modelica Conferences*, pp. 178–188.

Vanfretti, Luigi, Wei Li, et al. (2013). "Unambiguous power system dynamic modeling and simulation using modelica tools". In: *2013 IEEE Power & Energy Society General Meeting*. IEEE, pp. 1–5.

Vanfretti, Luigi, Biswarup Mukherjee, et al. (2019). "Automatic re-synchronization controller analysis within a multi-domain gas turbine and power system model". In: *2019 7th Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*. IEEE, pp. 1–5.

Weber, Niels, Camiel Cartignij, and Dirk Zimmer (2023). "Using the DLR Thermofluid Stream Library for Thermal Management of Fuel Cell Systems in Aviation". In: *Modelica Conferences*, pp. 415–422.

Winkler, Dietmar (2018). "Analysing the stability of an Islanded hydro-electric power system". In: *Modelica Conferences*.