# Advanced Distribution and Control for Hybrid Intelligent Power Systems

Final Technical Report – October 16, 2011

Michael Lemmon, Dept. of Electrical Engineering, University of Notre Dame

Agreement No. OT-UWM-11012009-03

Prime Contract No. W9132T-10-C-0008

**Abstract:** This technical report documents Notre Dame’s simPower simulation model of a distributed control and management system for military microgrids. The control/management system uses power inverters to connect a variety of distributed generation sources to the microgrid. Multi-unit stability is assured through the use of a decentralized control system that mimics the P-frequency and Q-voltage droop control used for synchronous machines. Optimal management of this system is accomplished through the use of a novel distributed peer-to-peer algorithm. This distributed optimization algorithm solves the optimal power flow problem in a distributed manner, relying only on local communication between generators on the microgrid. A novel event-triggered message-passing scheme is used to reduce the communication bandwidth used by this algorithm, thereby reducing the cost of communication infrastructure and potentially improving communication security.

The work documented in this report was performed from July 1st 2010 to October 1st 2011. During this period, a simPower simulation of the three-phase microgrid testbed at the University of Wisconsin, Madison (UWM) was built and tested. Another single-phase simPower simulation was built for the bench scale testbed being built by Odyssian Technologies. These simulations were used to develop and test distributed power dispatch and load shedding algorithms.

# Table of Contents:

|  |  |  |
| --- | --- | --- |
| Chapter 1 | Introduction | 5 |
| 1.1 | Background | 5 |
| 1.2 | Objective | 6 |
| Chapter 2 | Distributed Power Dispatch in Microgrids | 7 |
| Chapter 3 | Event-triggered Power Dispatch | 12 |
| 3.1 | Algorithm development | 13 |
| 3.2 | Simulation Testing | 15 |
| Chapter 4 | UND Simulator for UWM Microgrid Testbed | 18 |
| 4.1 | simPower Model Description | 18 |
| 4.2 | Experimental Comparison of UWM and UND Simulators | 23 |
| Chapter 5 | UND Simulator for Odyssian Testbed | 26 |
| Chapter 6 | Load Shedding Algorithms | 30 |
| 6.1 | Line Frequency Estimator Design and Evaluation | 30 |
| 6.2 | Simplified Frequency-based Load-shedding | 32 |
| 6.3 | Automatic Load-shedding with adaptive reconnection | 34 |
| Chapter 7 | Distributed Dispatch Algorithms | 37 |
| 7.1 | Centralized Implementation of Dispatch Algorithm for UWM Testbed | 37 |
| 7.2 | Distributed Implementation of Dispatch Algorithm for UWM Testbed | 43 |
| Chapter 8 | Testing of Algorithms on UWM Simulation | 48 |
| 8.1 | Centralized Dispatcher with Automatic Load Shedding | 48 |
| 8.2 | Large Scale Simulation Results | 52 |
| Chapter 9 | Conclusion  References | 53  54 |
| Appendix | Matlab/Simulink/simPower Components | 55 |

List of Figures:

|  |  |  |
| --- | --- | --- |
| Figure 1 | 3-bus microgrid with attached agents | 12 |
| Figure 2 | 3-bus microgrid used in event-triggered simulations | 15 |
| Figure 3 | Simulation result showing time history of generator power | 16 |
| Figure 4 | Simulation result plotting the time since last broadcast for event-triggered simulation | 16 |
| Figure 5 | Top-level simPower model for 3-bus mesh microgrid used in event-triggered dispatch | 16 |
| Figure 6 | Generator simPower model for event-triggered simulation | 17 |
| Figure 7 | UWM controller logic (simulink model) | 17 |
| Figure 8 | UWM Mesh Microgrid | 18 |
| Figure 9 | Notre Dame simPower model of UWM mesh microgrid | 20 |
| Figure 10 | Idealized Microsource Generator (simPower) | 21 |
| Figure 11 | simPower model of ideal microsource generator with UWM power inverter control component | 21 |
| Figure 12 | Simulink model of UWM power inverter controller | 22 |
| Figure 13 | simPower model of diesel generator with synchronous machine using UWM power inverter component | 22 |
| Figure 14 | Comparison results for UND simulator, case 1 | 23 |
| Figure 15 | Original results for UWM simulator, case 1 | 23 |
| Figure 16 | Comparison results for UND simulator, case 2 | 24 |
| Figure 17 | Original results for UWM simulator, case 2 | 24 |
| Figure 18 | Comparison results for UND simulator, case 3 | 25 |
| Figure 19 | Original results for UWM simulator, case 3 | 25 |
| Figure 20 | Initial Odyssian bench scale system | 26 |
| Figure 21 | Single phase inverter (simPower) model | 26 |
| Figure 22 | Response of initial Odyssian bench scale simulation | 27 |
| Figure 23 | Odyssian system simulation with three sources | 28 |
| Figure 24 | Simulation results for Odyssian bench scale system with 3 sources | 29 |
| Figure 25 | Frequency response and requested power generated by dispatch agents | 29 |
| Figure 26 | Estimation performance of Odyssian’s original zero-crossing frequency estimator | 30 |
| Figure 27 | Simulink Model of Phase-Locked Loop Estimator | 30 |
| Figure 28 | Signal Flow Graph of Direct Form II Structure of Low Pass Filter | 31 |
| Figure 29 | Commanded and frequency estimate obtained from PLL estimator on UWM simulation | 31 |
| Figure 30 | Simulink model used in testing simplified (non-adaptive) load shedding component | 32 |
| Figure 31 | Load shedding module | 32 |
| Figure 32 | Triggering component of load shedding module | 33 |
| Figure 33 | Simulation results for simplified load shedding algorithm | 33 |
| Figure 34 | simPower Model for UWM simulator with centralized dispatch logic | 40 |
| Figure 35 | simPower Model for UWM simulator with distributed dispatch logic | 43 |
| Figure 36 | Interface between UWM controller and Odyssian Dispatch Agent | 46 |
| Figure 37 | Signals within UWM controller required by the dispatch agent | 47 |
| Figure 38 | simPower Model for UWM simulator (centralized dispatcher and load shedding modules) | 48 |
| Figure 39 | Simulation results for case 1 | 49 |
| Figure 40 | Simulation results for case 2 | 50 |
| Figure 41 | Simulation results for case 3 | 50 |
| Figure 42 | Simulation results for case 3 showing how the requested and generated power track each other | 51 |
| Figure 43 | Simulation results for case 4 | 51 |
| Figure 44 | Simulation results for case 4 showing how the requested and generated power track each other | 51 |
| Figure 45 | Large scale 3-phase microgrid and simulation results | 52 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Chapter 1: Introduction

## Background

Microgrids [1] are generation/distribution systems in which generation is relatively close to the loads. They can provide a higher level of local resilience to variations in the main power grid’s power quality. This means that microgrids are useful in applications where power reliability is a critical concern. Examples of such applications include hospitals, industrial parks, as well as forward military bases. Microgrids therefore play an important role in maintaining this nation’s military readiness.

Decentralized PQ control of distributed generation in microgrids has been previously demonstrated to maintain power quality during islanding and reconnection of the microgrid [2,3]. The equilibrium power levels achieved by these controllers, however, are not necessarily optimal from the standpoint of minimizing overall operating cost. Optimal generation dispatch may be realized through centralized supervision of the entire microgrid. But this approach will be difficult to maintain as the system grows and is therefore not seen as a *scalable* approach to microgrid energy management. In particular, for applications where the microgrid is expected to evolve over time, one needs to develop an approach to the management of microgrid generation that can quickly adapt to changes in system topology and application mission.

The approach being used to meet this need involves the distribution of dispatch and load-shedding decisions across the network [4]. In other words, rather than using a centralized top-down command structure, this project uses a distributed bottom-up approach in which computational agents located at the distributed generation source or load bus make *local* decisions conditioned on information received from neighboring agents. This *networked* approach to decision-making scales well with system size since the configuration data regarding the grid is stored in a distributed manner. It also provides greater security and resilience to abrupt changes in application mission or grid topology since decision-making and grid configuration data are not handled in a centralized manner.

The problem addressed in this project is the development and simulation of a scalable distributed scheme for dispatch and load management in military microgrids. This work is being done as part of a DoD phase II STTR project (Prime Contract No. W9132T-10-C-0008). The University of Notre Dame (technical contact: Michael Lemmon), as a subcontractor to the University of Wisconsin – Madison and Odyssian Technology, is performing this work. The technologies being developed under this project are intended for use in managing electrical generation and loads within microgrids used by military bases.

## 1.2 Objective

## The objectives of this project are to

## Develop a simulation of a three-phase microgrid that has been specified by the University of Wisconsin – Madison.

## Developed distributed algorithms for the dispatch of power and intelligent shedding of loads.

## Assist the prime contractor (Odyssian Technology) in the development of embedded software implementing the dispatch and load shedding algorithms.

## Assist the prime contractor (Odyssian Technology) in developing a wireless communication for implementing the proposed algorithms.

## The remainder of this report is organized as follows. Chapter 2 discusses the theoretical foundations behind the distributed dispatch algorithms being developed under this project. Chapter 3 describes an *event-triggered* approach to distributed dispatch that can greatly reduce the amount of communication needed by this approach. Chapter 4 describes the simPower simulation for the three-phase testbed at the University of Wisconsin Madison. Chapter 5 describes the simPower simulation for the single-phase bench scale system being built by Odyssian Technologies. The dispatch algorithm described in chapter 2 and 3 were modified for the UWM and Odyssian testbeds. Chapter 6 describes these modified dispatch algorithms and presents simulation results regarding their expected behavior. Chapter 7 describes the automatic load shedding algorithms that were developed for Odyssian Technologies and presents simulations results for both the UWM and Odyssian testbeds. Chapter 8 summarizes the University of Notre Dame’s main technical findings for this project.

**Chapter 2: Distributed Power Dispatch in Microgrids**

The power system is modeled as a directed graph, G=(V,E) where V={v1,v2, … ,vN} is a set of *nodes* representing the system buses, E⊆V×V, is a set of directed *edges*, representing the power distribution lines. An edge from node *I* to node *j* is denoted as *eij=(ve,vj)* with impedance *zij=rij+jxjj*. We assume that the line resistance *rij* is negligible compared to the reactances *xij*. Let *I* denote the *incidence* matrix of the graph G and let *D* be a diagonal matrix whose entries are the reactances of the distribution lines. We let *A=DI* denote the *weighted incidence matrix* of the graph G. The set of neighbors of node *I* is denoted as *N(i)* and the set of distribution lines leaving node *I* is denoted as *L(i).*

Let *Sij=Pij+jQij* denote the complex power flow from node *I* to node *j*, and *ui* denotes the generator voltage at node *i*. This voltage is represented in phasor form as *ui=|ui|exp(jθj)*. Under normal operating condition voltages the bus voltages are about equal. In a similar manner, the bus phases are about equal so that the phase difference, θi-θj , is typically small. In this case, the flow of active and reactive power are decoupled so the active power is mainly dependent on θi-θj and the reactive power flow is mainly dependent on |ui|-|ui|.

Let’s confine out attention to controlling the flow of active power, *Pij*, This assumption is reasonable provided the voltage magnitudes are nearly constant across the grid. Under this situation the real power flow between node *I* and node *j* is given by



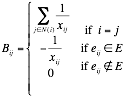
The total power flowing into bus (node) *I* is denoted as *Pi*. This must equal the power generated by generator *I* minus the power absorbed by the local load on the bus. This power, *Pi*, therefore must equal the sum of the power flowing away from bus *I* on all transmission lines. This means that



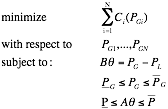
which can be expressed in matrix form as



where *P=[P1, … , PN]*, θ*=[θ1, … , θN],* and *B* is defined as



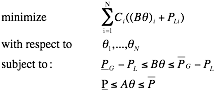
Let Ci(P)∈ R be a real-valued convex function representing the *cost* incurred in running generator *I* at power level P. We may then formula a general *optimal power flow* problem as follows



where PG is the vector of generated active powers for all generators and PL is the vector of total local loads for all buses. The matrices *A* and *B* were defined above. The vector and represent the lower and upper limits on generator power. These are the generation constraints. The other vectors, and , are lower and upper limits on the power flowing through the distribution lines. The objective function given above represents the total generation cost of all generators. The problem seeks to minimize this overall cost by selecting generating powers that satisfy three constraints. The first constraint is a power balance relation. The second constraint requires that the selected power levels stay within the limits specified by and . The third constraint requires that the power flowing over the distribution lines stay within the specified bounds, and .



In solving this problem it will be more convenient to represent the decision variables in terms of the phasor angles, θI , since these angles directly control real power flows. In addition to this, the power flow constraint must always be satisfied in the network. We may, therefore, recast the original optimal power flow problem as a *modified* problem in which the decision variables are the phase angles. This modified power flow problem is



where (Bθ)I is the *i*th element of *Bθ.* Note that the modified optimization problem is solved with respect to the phase angle θ, rather than the generator power set points.

The optimization problem given above is similar to *network utility maximization* problems that have appeared in the communication network community [5,6,7,8]. One unique feature of these problems is that they can be solved in a *distributed manner.* What this means a bus generator in the system can decide its own generation set point using only the information of those loads and generators on buses that are directly connected to it. In other words, decision-making can be distributed amongst the individual generators in the system and the communication required to support that decision-making only has to be between neighboring buses. This distributed approach to power dispatch may be referred to as *peer-to-peer* dispatch [4].

Peer-to-peer dispatch represents a novel distributed way of dispatching power is microgrids. This approach avoids the use of centralized command and control centers in managing power generation.

The distributed algorithm used to solve our modified power flow problem is based on the so-called *augmented Lagrangian method*. In this approach, the original constrained problem is converted into a sequence of unconstrained problems by adding to the cost function a penalty term that assigns a high cost to infeasible points. Take the constraints, for example. We introduce a slack variable *s∈ RM* and replace the inequalities for all *j in E* by



Here the vector is the *j*th row of the incidence matrix *A*. We then define a penalty function of the form,



where *w* is a penalty parameter associated with the distribution line. It is easy to show that



In a similar way we can define a penalty function for the constraint to obtain



Penalty functions for the other constraints can be defined in a similar way to obtain



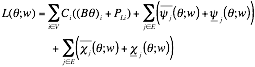
for constraint for all *k* in *V* and



for constraint for all *k* in *V.*



These penalty functions are used to *augment* the original cost function. The resulting augmented cost is



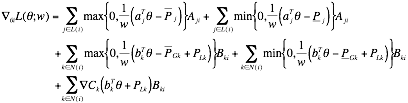
The function *L(θ;w)* is a continuous function of θ for fixed weights, *w*. We now define a sequence *w[k]* of weights that decrease monotonically to zero and let θ\*[k] denote the approximate minimizer of *L(θ;w[k]).* It has been shown that as *k* goes to infinity, the sequence θ\*[k] of approximate minimizers approaches the optimal solution to the modified power flow problem.

Rather than seeking the exact minimum solution, we seek an approximate solution for a given weighting parameter, *w.* If *w* is sufficiently small, then the approximate minimizer for this parameter will be a good approximate to the original power flow problem. We may search for the minimizer using a gradient descent algorithm in which



for each generator *I* in *V.* The derivative of the cost, *L(θ;w)* can be shown to be

We may simplify this expression by defining some variables that are representative of the *edge’s local state* and the *node’s* local state. In particular, for each distribution line define



In this case is simply the power flow on the line *j* at time *t*. The parameter *w* is a coefficient that penalizes the violation of the line flow limit. It is easy to see that is nonzero if and only if the flow on the *j*th line exceeds the flow limits. We can therefore see as summarizing the information about the *j*th line’s power flows at time *t*. In particular, we’ll find it convenient to refer to as the *j*th line’s *state.*



In a similar way we’ll find it convenient to define a state for the *k*th node (generator) in the grid. This state will be defined as



where *w* is a constant penalty coefficient that levies a cost for violating the generation limit constraints of the generator.

With the above definitions for the line state and generator state, we can now simplify our expression for the gradient of the augmented cost and obtain



and the gradient descent algorithm takes the form



Note that the *i*th generator computes the above equation only using information about its own local state, ϕI , the generator states of its nearest neighbors, and the line state, μk , of those lines leaving bus *i*. This means that the computation of the phase angles is done in a distributed manner because each node only needs local information to complete its computation.

Distributed computation has a number of potential advantages relative to centralized computation of the optimal dispatching vector. These advantages are itemized below

* **Greater resilience to faults**: Since a distributed algorithm distributes the decision-making and storage of information across the entire system, there is no single point of failure. Even if some of the information within the algorithm is lost or incorrect, the system can still compute a reasonable dispatching solution.
* **Lower Cost Communication Infrastructure**: Prior work in developing centralized traffic control schemes in municipal settings have suggested that the costs of communication equipment do not scale gracefully with system size. By forcing all information to be gathered by a single command and control center, one greatly increases the complexity and hence cost of the associated communication network.
* **Lower Cost Modeling Efforts:** By distributing the workload, one only needs to use local models of systems. Moreover, since information is only exchanged locally, it means that systems can more quickly see what their neighbors are doing, thereby providing faster response to faults. In other words, the improved communication speed results in lower sensitivity to errors in modeling, thereby reducing the overall cost of developing a model for such systems.
* **Easily expandable or Plug-and-Play Functionality:** Again, because information is stored locally, this means that new nodes can be added to the system without requiring a global recalibration of the entire system. In essence, one can simply “add” a new node, have that node broadcast its data to its nearest neighbor, and the system will again be able to dispatch generation in an optimal manner.

In spite of these benefits, there are some potential limitations of the proposed approach, which will be addressed in the next chapter.

Chapter 3: Event-triggered Dispatch

While distributed dispatch appears to promise many benefits with regard to greater fault-tolerance and lower communication and modeling costs, there are some potential issues that still need to be addressed. In particular, the idea of attaching a computational agent to each generator and then letting those agents communicate freely with each other suggests that we might want to make use of wireless communication networking technologies. There are potential issues in using wireless communication on critical electrical infrastructure with regard to security and reliability. This chapter presents one way of handling those issues using a so-called *event-triggered* approach to message passing [9,10,11].

We view the system as shown in figure 1. This shows a microgrid consisting of three buses in a mesh configuration. A generation source is attached to each bus. These sources are assumed to be controlled by computers called *agents*. The agents are equipped with wireless radios that form a multi-hop communication network. This network allows agents to exchange local information over a single hop. This information is used to solve the distributed optimization problem posed in the preceding chapter.

Figure 3-bus microgrid with attached agents

Wireless communication technologies appear to be a natural technology for this type of system. The communication network links are adaptive and can reform when new nodes enter the system. They do not require the construction of wired infrastructure whose installation costs can be unwanted. By avoiding the use of wired infrastructure, it becomes more difficult to physical compromise the communication network.

The use of wireless technologies, however, also raises issues that may negatively impact the system’s overall performance. The reliability of these links can be time varying. In other words, we may not be guaranteed that a given message reaches its destination. Secondly, the wireless channel is open in the sense that anyone can listen to it and potential interfere with it. This means that while there is no physical wire to break in this system, it is possible for an adversary to jam to transmission provided they know when a transmission is about to occur.

The gradient descent algorithm outlined in Chapter 2 may not work well in a wireless environment. The algorithm assumes that each node has direct access to its neighboring node states and line states. This means that each time a generator’s local state is updated; it must first access the state information from its neighbors. In general, these algorithms may require hundreds of updates before converging to the desired solution point, which means frequent requests for neighboring information. The bandwidth requirements for these algorithms, therefore, may quickly overwhelm the capacity of the wireless communication network.

**3.1 Algorithm Development**

One way around this issue is to dramatically reduce the amount of information that has to be exchanged between neighboring agents. Breaking the tight connection between communication and computation in these gradient descent algorithms does this. Recent work demonstrated that an *event-triggering formalism* could reduce the required message passing complexity of the algorithm by two orders of magnitude [12,13,14].

Event-triggered message passing has agents broadcast their local states only when some measure of the information novelty in that state exceeds a pre-specified threshold. In other words, these agents broadcast only when they expect their data will have a significant impact on the behavior of their neighbors. By adopting a transmission policy that only sends data when it is needed, we break the tight connection between communication and computation in a manner that greatly reduces the amount of transmitted data.

Another interesting feature of event-triggered message passing is that it usually generates *sporadic* message streams. In other words, the time between consecutive transmissions varies in a random manner that is difficult to predict by an outside observer. This has potential benefits with regard to securing wireless traffic. An easy way of disrupting a wireless network is to set up a narrowband transmitter that *jams* the transmitter’s broadcast. In cases where transmitters periodically transmit data, it becomes rather easy for an adversary to identify the frequencies and times at which such jamming should be done. Adopting an event-triggered message-passing scheme, however, results in a sporadic scheme in which 1) very few messages are passed and 2) the time between broadcasts is difficult to predict. This means that event-triggered message passing will make it difficult for adversaries to determine the best time to activate their jamming systems. Event-triggered message passing, therefore, may be able to improve the security of such wireless systems to outside interference.

The gradient descent algorithm assumes that generator *I* updates its state using information for its neighbors’ states. As noted above, this would require frequent message passing between agents. An *event-triggered* version of the update equation assumes that generator *I* only accesses a *sampled* version of its neighbor’s state. In particular, let’s associate a sequence of *sampling* instants, , with the *i*th generator. The time denotes the instant when the *i*th generator samples its state ϕI for the *l*th time and transmits that state to neighboring generators *k∈ N(i)*. We can see that at any time *t*, the sampled generator state is a piecewise constant function of time in which



for all and any time . In this regard the event-triggered version of the gradient descent update’s algorithm now takes the form,



for all and any time . The sequence represents the time instants when generator *I* transmits its “state” to its neighboring generators. Here we assume there is no transmission delay in each of the broadcasts.



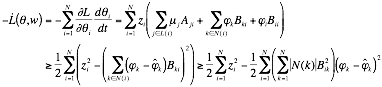
A systematic method must be used to select the sampling times . The main consideration is that these sampling times must be chosen to ensure that the “sampled” version of the gradient descent algorithm converges to the optimal dispatch decision. This selection is based on Lyapunov type arguments in which the Lagrangian, *L(θ;w)*, becomes a candidate Lyapunov function for the sampled system.



To ensure the algorithm’s convergence, we need to guarantee that the time rate of change in the Lagrangian is always negative. Let’s first define the local variable *zi*



to simplify the notation in the derivation. We now compute the derivative of Lagrangian. In particular, for all t > 0, we have



The preceding equation shows that is negative provided



for each generator *i.* Note that this requirement only requires information available to the *i*th generator. Moreover, we can recast this inequality as a thresholding condition of the form



where



is a constant. The thresholding condition given above requires that generator *I* transmit its local state when the difference (gap) between the current local state of the generator and the last *transmitted* state of the generator exceeds the state-dependent threshold . This ensures that the transmission time sequences are chosen so that is negative. So the Lagrangian becomes a Lyapunov function for the sampled gradient descent system and we can guarantee that this system converges to the optimal dispatch states.



As it turns out the proposed event-triggered dispatch algorithm can be easily integrated into the power inverter controller developed by UWM [2]. This is done by dynamically adjusting the requested power for each generator. In the microsource power inverters, each generator’s phase angle, θI, is adjusted by comparing the measured active power and the requested power so that the phase angle follows the differential equation



This suggests that if instead of fixing *Preq,I* , we can adjust it in a dynamic manner so that the θi(t) follows the sampled gradient update algorithm, then the requested power at each generator should converge to a value that globally minimizes the overall system’s operational costs subject to the generator/line power constraints inherent in the network. In particular, this is done by setting



where γ>0 is a constant that controls how fast we adjust the phase angle. This constant is needed because if *zi(t)* is adjusted too fast, then we may destabilize the entire system. Since generator *I* can compute both *PGi* and *z,I* locally, this means that *Preq,I* can be easily computed by generator *I* itself. This suggests that each generator only needs to adjust its power set point according to the above equation. It samples and then broadcasts its state ϕI to its neighboring generators when the event-triggering inequality is violated. If every generator follows this action, then our prior analysis guarantees that the generated power of all generators in the system should approach the solution to the optimal power dispatch problem.

**3.2 Simulation Testing**

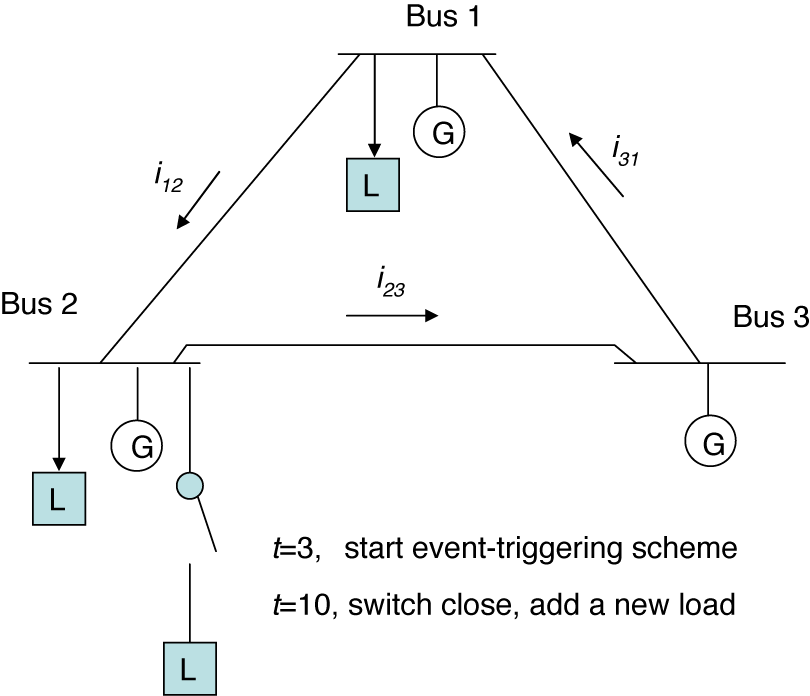


Figure 3-bus microgrid used in event-triggered simulations

A simple simPower simulation was built to test the correctness of the proposed event-triggered dispatch algorithm. The results in this chapter originally appeared in [4]. The simulation used here was a Matlab simPower simulation of the islanded microgrid shown in figure 2. The scenario chosen here assumes that the event-triggered dispatch algorithm is started at t=3 seconds and then an extra load is added at 10 seconds into the simulation. The cost functions were chosen so that generator 2 is the most expensive generator to operate and the distribution line between bus 3 and bus 2 is flow limited.

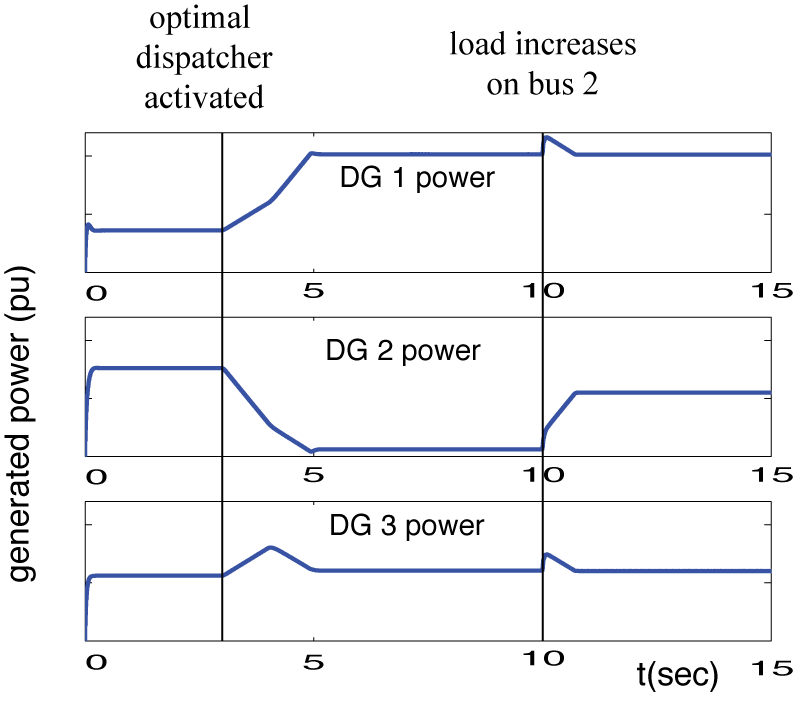
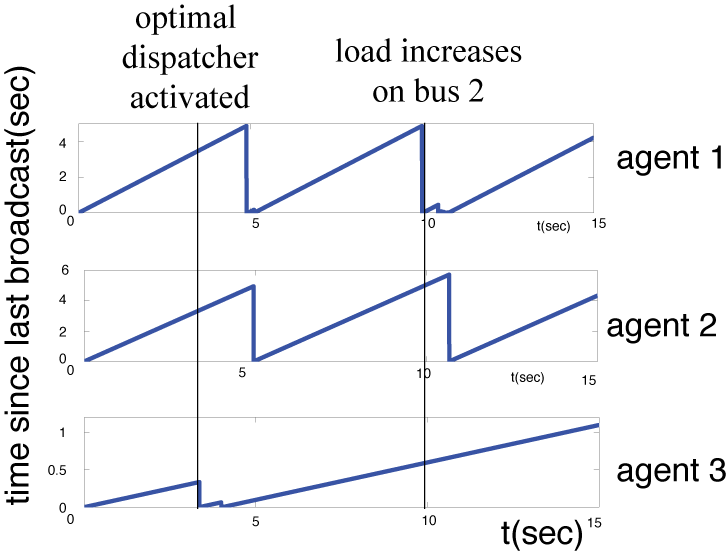
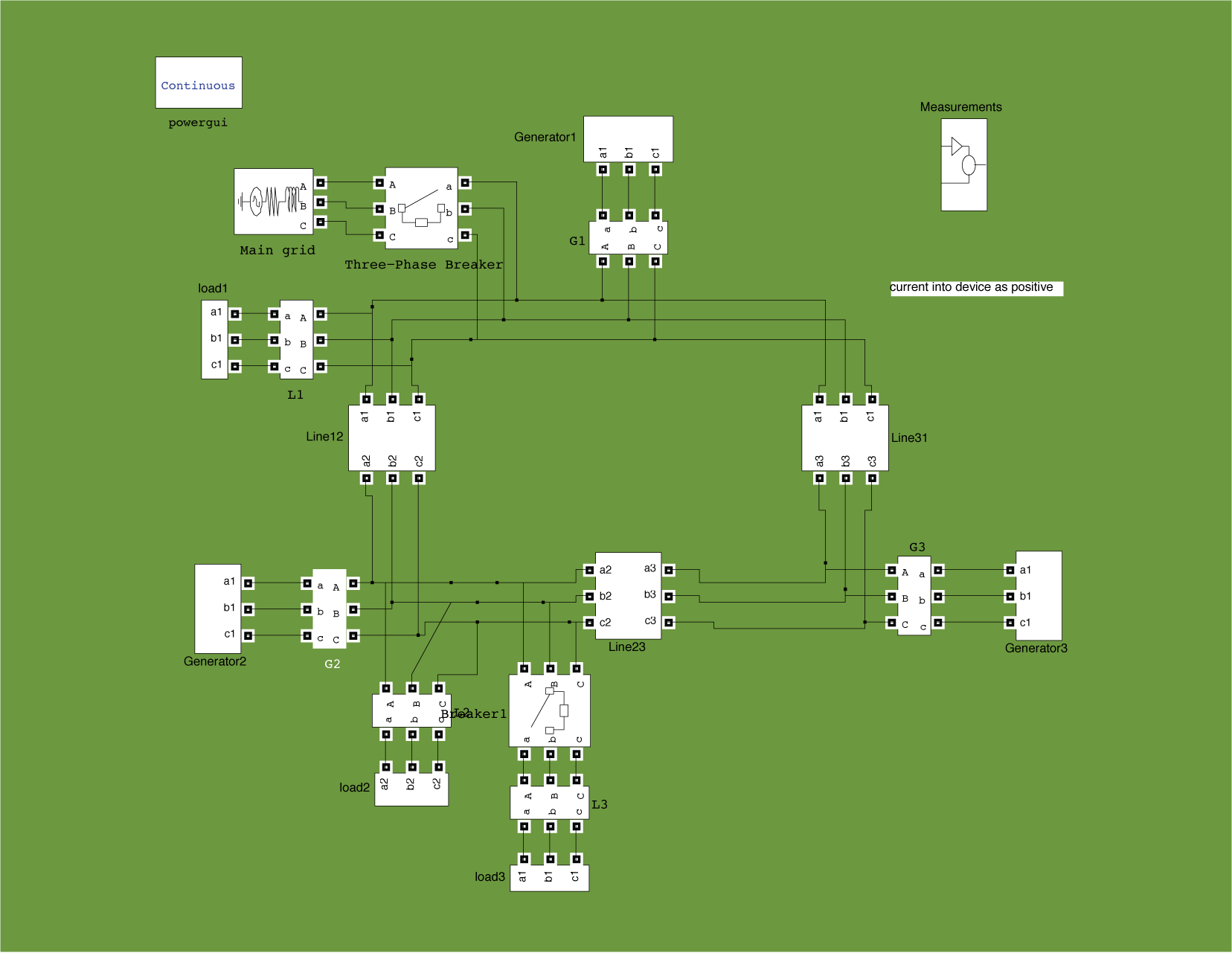
Results from this simulation are shown in figure 3. The power generated by each of the generators is shown in the plots as a function of time. As one can see, the optimal dispatcher is switched on at time t=3. Since generator 2 is the most expensive one to operate, we see generator 1 increase its power level and generator 2 reduce its dispatched power. At time t=10 seconds, an extra load is switched onto bus 2. In this case, however, the tie line between bus 3 and 2 is already fully loaded. Generator 1 cannot increase its output because it is already at its limit. Therefore generator 2 increases its power level, even though it is the most expensive generator to operate. These plots suggest that the proposed event-triggered dispatch algorithm is operating as expected.

Figure Simulation result showing time history of generator power

Figure : simulation result plotting the time since last broadcast for event-triggered simulation

The next plot to the right shows the communication cost associated with the event-triggered dispatch algorithm. This figure plots the time since last broadcast for each of the generators. The actual broadcast times occur at the discontinuous jumps in the curve. Notice that in all cases, the generators only broadcast when an event occurs. These events correspond to the activation of the optimal dispatcher and the abrupt change in load on bus 2. The important thing to note as that the time between successive broadcasts is usually very long. Indeed only a dozen messages were passed between the 3 generators during this simulation. So the approaches benefits with regard to reducing communication traffic are indeed realized.

The remainder of this chapter describes the simPower simulation that was used to obtain the preceding results. This simulation serves as the starting point for the microgrid simulation being developed under this project. So a description of this early simulation model will be useful in highlighting the new features of this project’s simulation.

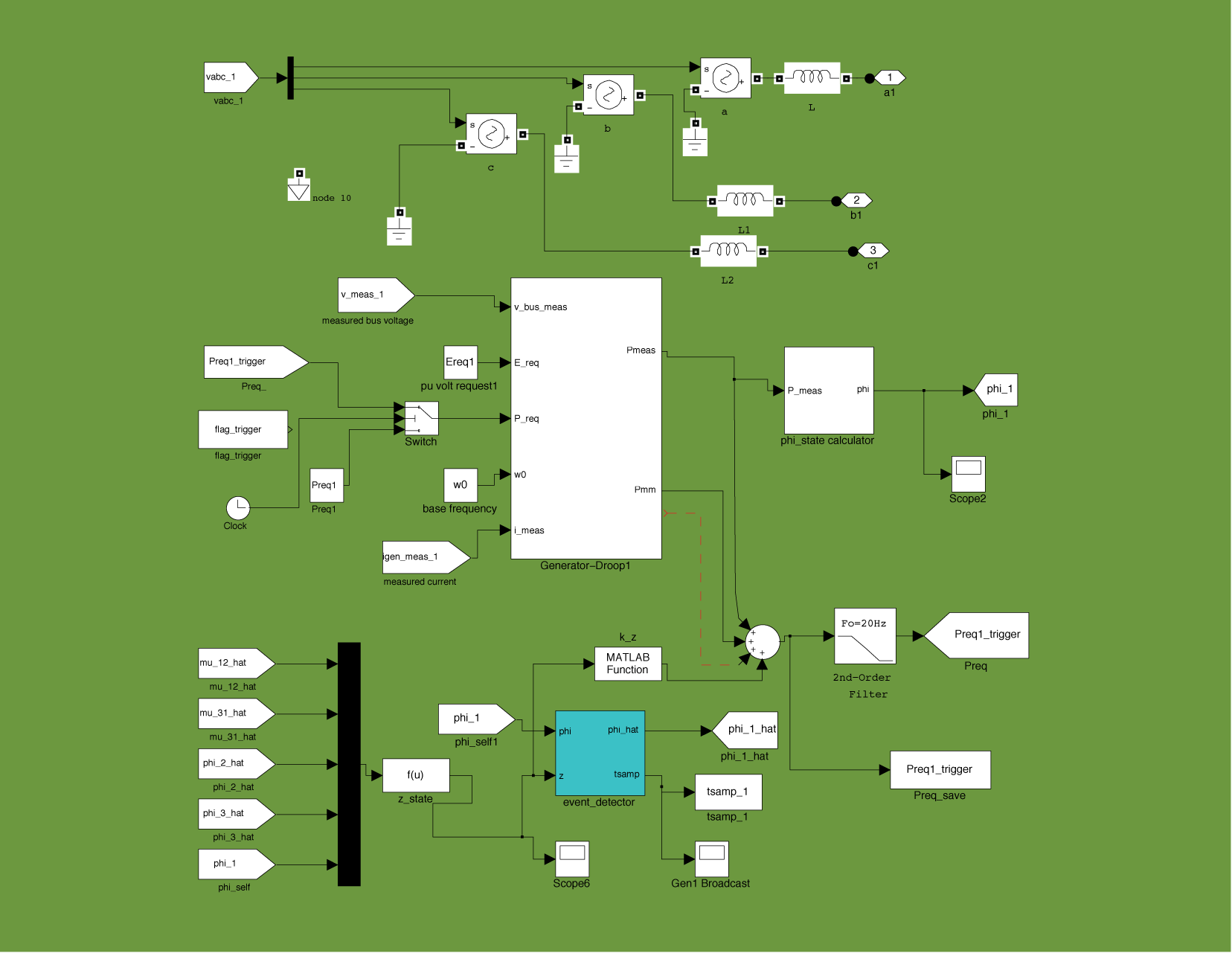
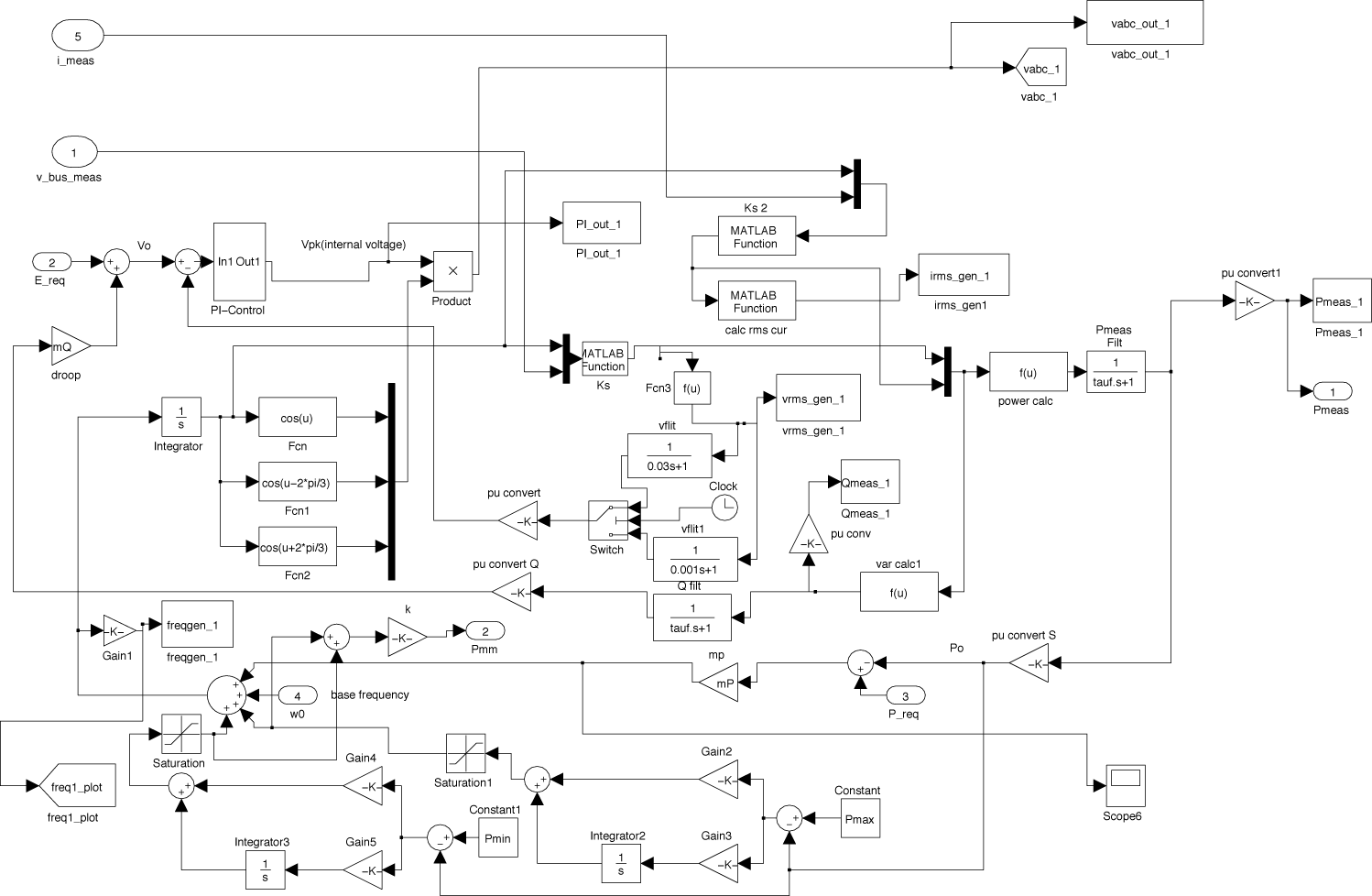
The top-level simPower model for this microgrid is shown to the left. The upper left hand corner shows the connection to the main grid. Bus 1 is shown on the top, bus 2 on the lower left and bus 3 on the lower right side of the figure. The main block of interest is the generator block.

Figure : top-level simPower model for 3-bus mesh microgrid used in event-triggered dispatch

Expanding out the generator block yields the simPower model shown on the right. Main blocks of interest include the central block, which contains the UWM controller logic, and the lower block for the event trigger. This event-triggering block realized the decision logic discussed in the preceding chapter. It was implemented as an S-function block.

Figure : generator simPower model for event-triggered simulation

The UWM control logic was realized as a Matlab Simulink model shown to the left. It is similar to the simulink model developed by UIUC under phase I of this project with some minor modifications that were needed to support the event-triggered dispatching system.

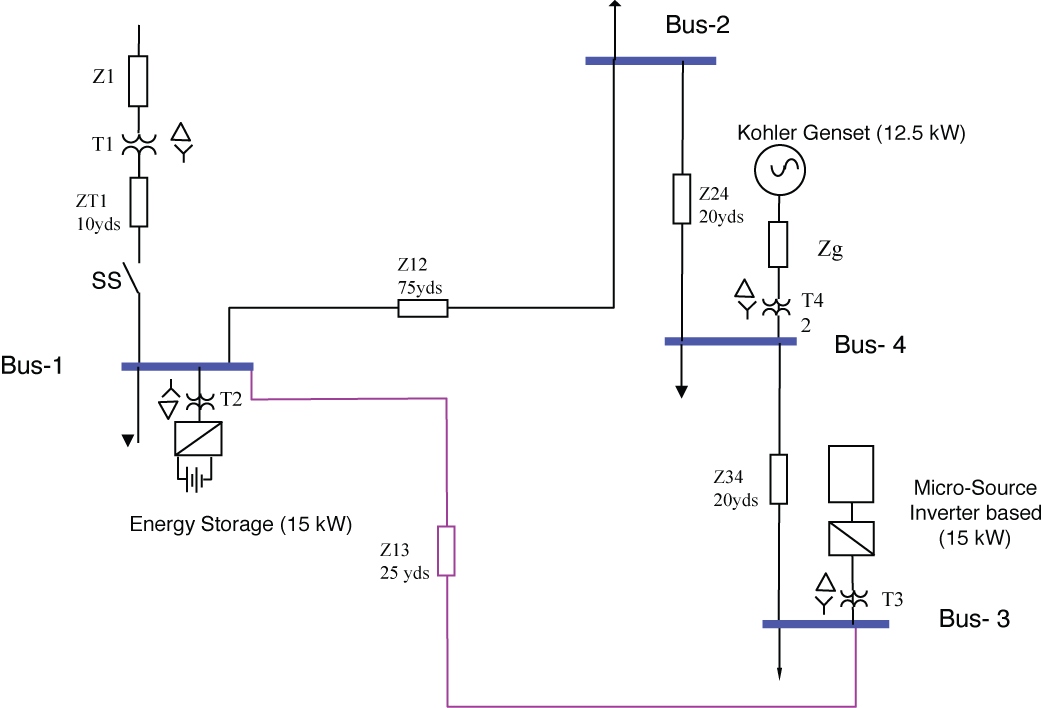
While this simulation worked well in demonstrating the effectiveness of the event-triggered dispatch algorithm, it was insufficient for the needs of this project. The preceding models lacked sufficient modularity to enable the straightforward integration of different generation sources. A more flexible and accurate simulation model for the UWM testbed will be described in chapter 4.Chapter 4: UND Simulator for UWM Microgrid Testbed

Figure : UWM controller logic (simulink model)

This chapter describes the simPower simulation that was built by the University of Notre Dame for the UWM Mesh Microgrid. This simulation represents one of the main deliverables of this project. The remainder of this section is organized as follows. Subsection 4.1 describes the simPower model components and subsection 4.2 describes the simulation experiments run to verify the correctness of the proposed models.

**4.1 simPower Model Description:** The microgrid simulation being developed under this project was proposed by University of Wisconsin – Madison (UWM) in a meeting that was held at UWM on February 19, 2010. This microgrid is a modification of the physical

Figure : UWM Mesh Microgrid



testbed at UWM. A schematic of the microgrid is shown in the figure above. This microgrid has four buses and three sources. The sources are an external storage (15 kW), a diesel engine with synchronous machine (12.5kW) and a microsource with inverter (15 kW). Each of these sources is connected to the distribution lines through a transformer. The parameters for the transformers and cables connecting the buses are shown below in table 1.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Cables | | Length yds | | R – ohms | | X – ohms | |
| Z1 | |  | | 0.0934 | | 0.0255 | |
| ZT1 | | 5 | | 0.0028 | | 0.00068 | |
| Z12 | | 50 | | 0.0274 | | 0.0066 | |
| Z24 | | 30 | | 0.0168 | | 0.0041 | |
| Z34 | | 30 | | 0.0168 | | 0.0041 | |
| Z13 | | 25 | | 0.0137 | | 0.0033 | |
| Zg | |  | | 0.0656 | | 0.0021 | |
| Transformer | voltage | | KVA | | Primary  R – pu | | Primary  X – pu | | Secondary R – pu | Secondary  X – pu |
| T1 | 480-208 | | 75 | | 0.0169 | | 0.0676 | | 0.0003 | 0.0127 |
| T2-T4 | 480-208 | | 45 | | 0.0269 | | 0.1075 | | 0.0050 | 0.0201 |

Table 1: UWM Mesh Microgrid Parameters

A simPower model of this microgrid was developed. The diagram for this model is shown below in figure 9. The main grid blocks are shown on the upper left hand side of the figure. The three other generators in this microgrid are on the left hand side of the figure and the loads are shown on the right hand side. This microgrid has two microsources (480 V and 15 kW) and one diesel generator (480 V and 12.5 kW). At the time of the writing of this report, a preliminary model of the external storage source had been developed. This external source model has yet to be integrated into the microgrid simulation. Note that after each coupling transformer, one finds an RC tank circuit that is connected to ground as a load absorbing 100 W and 10 kVAR. This tank circuit was used to reduce switching transients that occur when the system islands. The loads on bus 1 and bus 3 are 5 kW each. The load on bus 2 is 10 kW. Bus 4 has 10 kW load that increases to 13 kW at 2 seconds into the simulation.

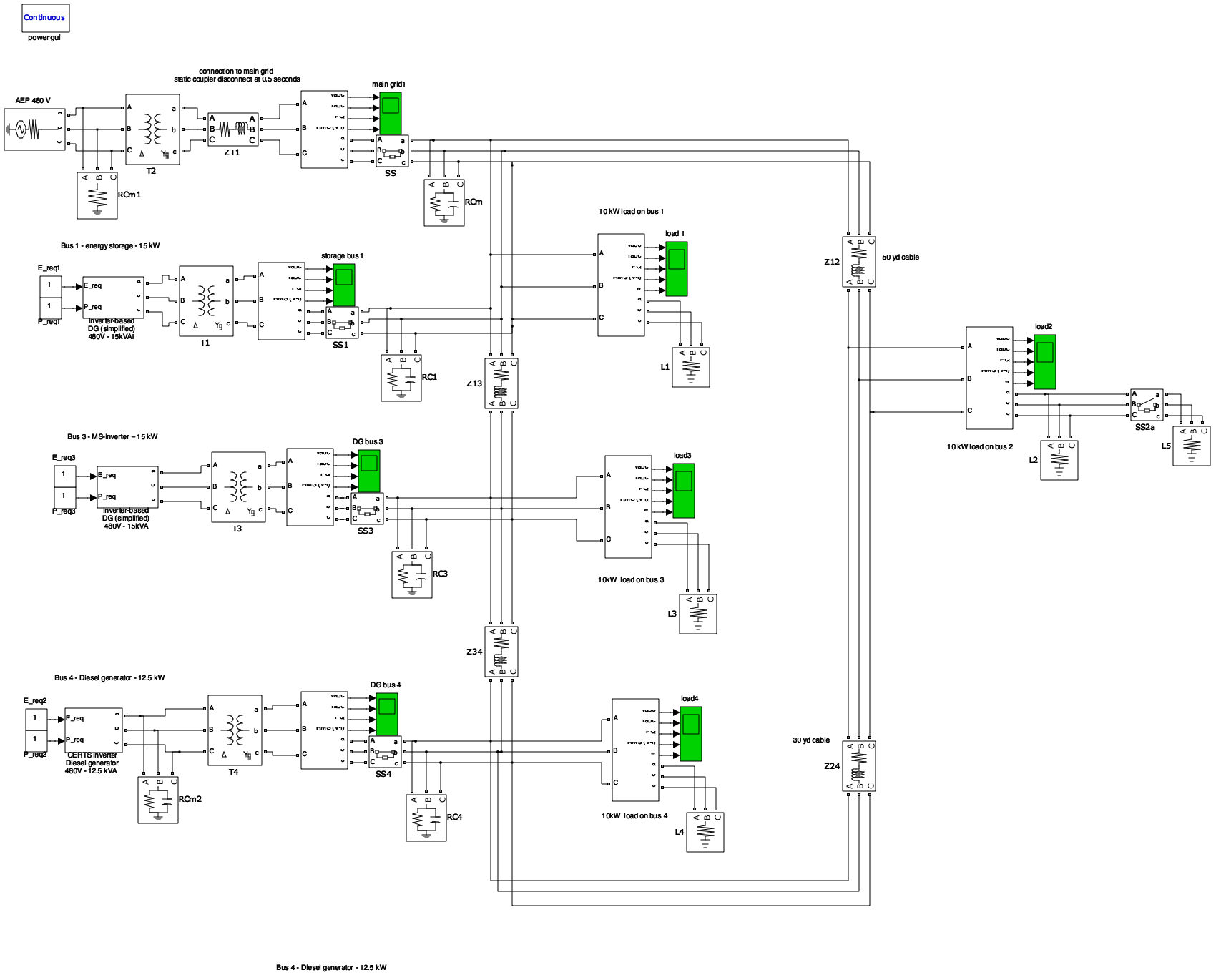


Figure : Notre Dame simPower model of UWM mesh microgrid

A number of simPower models were developed for this simulation. These blocks were specifically designed to allow easy modification of the microgrid simulation. For example, the UWM controller logic that was originally developed for the simulation in chapter 4 was rewritten so it could be interconnected in a modular manner with a wide range of sources. Each of these blocks will be described below.

The microsource model is shown below in figure 10. This is an “idealized” model of a microsource that treats the generator as a 3-phase voltage source whose magnitude and frequency can be set through external simulink inputs, *w* and Vpk. The outputs are simPower signals (Va,Vb, and Vc) representing the three voltage phases.

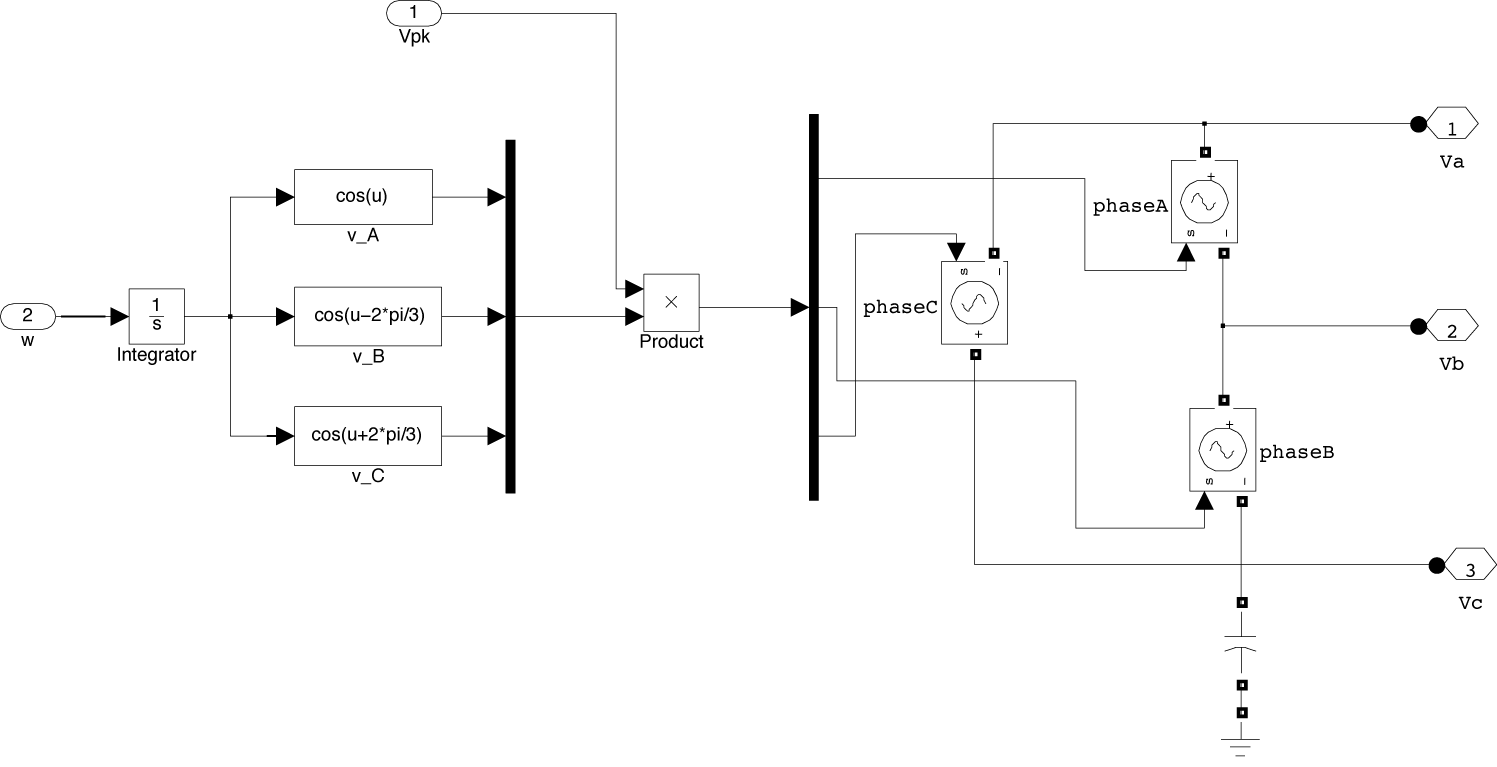


Figure : Idealized Microsource Generator (simPower)

This idealized generator model is connected to the UWM power inverter model in the manner shown below in figure 11. In this case, the UWM power inverter takes as inputs the measured terminal voltages (Vabc) and currents (Iabc) for the three phases. A terminal measurement unit obtains the signal Vabc and Iabc. The UWM power inverter also takes the requested voltage level (E\_req) and requested power level (P\_req) as inputs. These last two inputs are set points that determine the nominal active and reactive power that can be delivered by the source. In this simulation these two inputs are user constants. The output of the UWM power inverter is the peak voltage (Vpk) and frequency (w) that is input directly into the microsource block. The microsource is connected to the grid through a series inductor (used to adjust power factor) and a harmonic filter to remove power inverter switching harmonics.

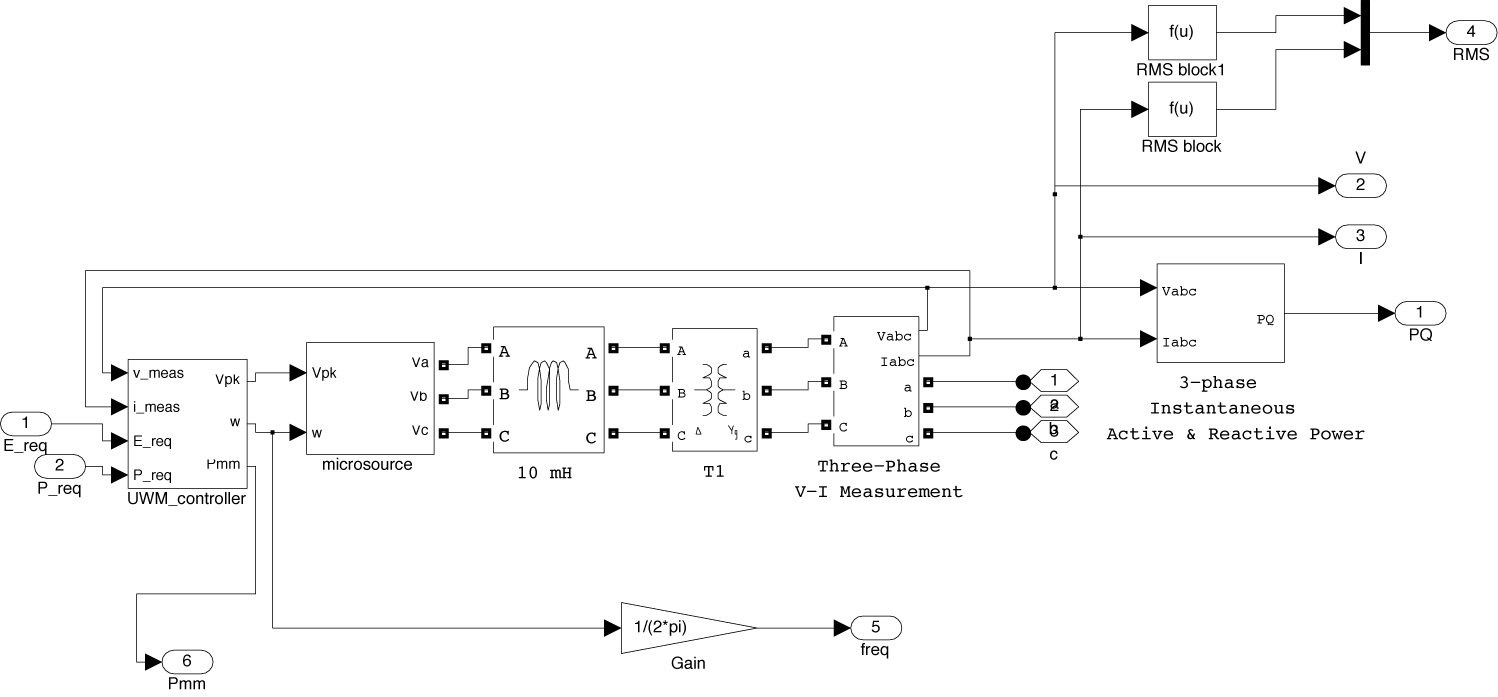


Figure : simPower model of ideal microsource generator with UWM power inverter control component

The UWM power inverter logic is shown in the figure 12. This logic was extracted from the earlier UWM controller logic developed for the simulation in chapter 5. The main difference lies in the reorganization of that model so that the UWM power inverter logic can be interfaced with other types of sources. The controller takes the terminal voltages and currents (Vabc and Iabc) and computes the real and reactive power. The real power is used in a P-freq droop controller to adjust the frequency, w. The reactive power is used in a Q-voltage droop controller to adjust the peak voltage requested (Vpk) requested of the microsource.

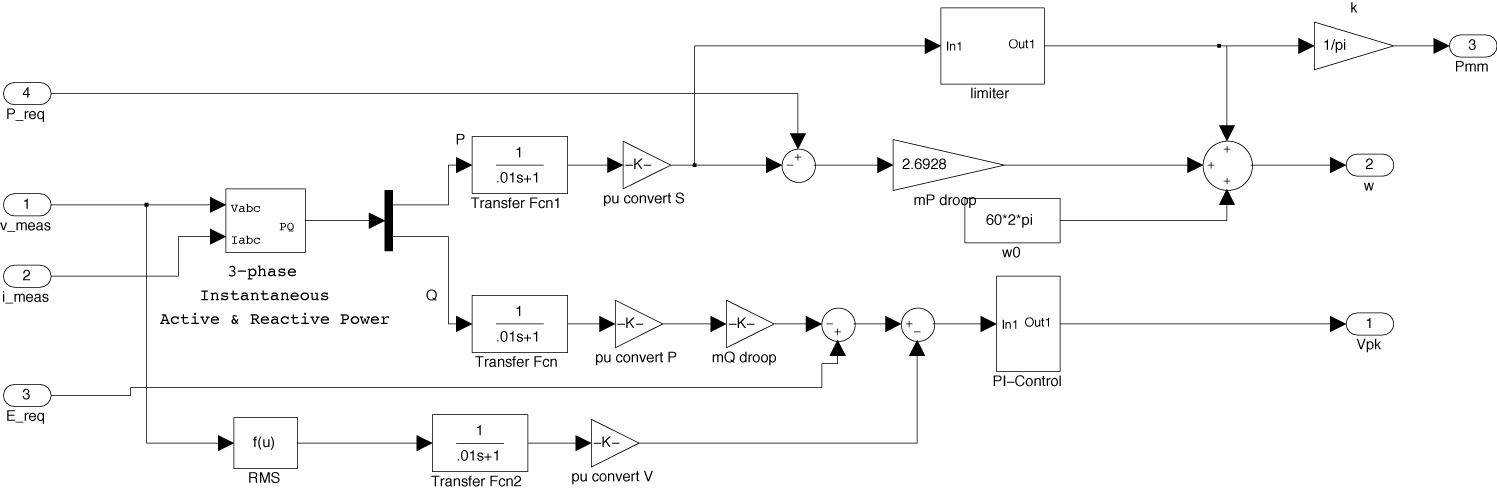


Figure : simulink model of UWM power inverter controller

The modularization of the UWM power inverter allowed us to easily control a variety of more realistic generators. For example, the integration of the UWM power inverter with a simple Matlab supplied diesel generator is shown in the following simPower model (figure 13). In this case, the generator consists of a diesel engine with speed and voltage controller connected to a synchronous motor (SM) in a feedback topology. We can treat the feedback connection of the diesel generator and SM as a block similar to the idealized microsource block. The UWM power controller is then connected around this Diesel-SM subsystem using the same interconnection topology used in constructing the microsource control system model.

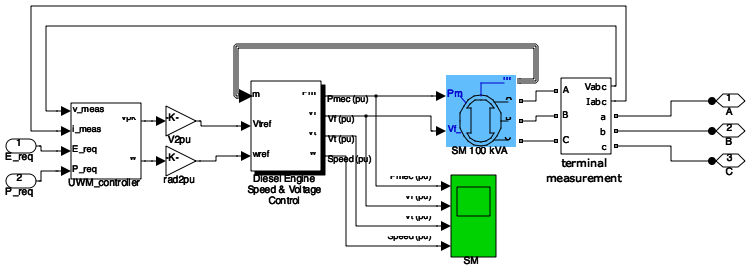


Figure : simPower model of diesel generator with synchronous machine using UWM power inverter component

A simple modification of the UWM controller for the microsource was developed to model an external storage device (battery). In this case, the controlled external storage (ES) device was modeled as a microsource whose minimum power, *Pmin*, was set to a negative value.

**4.2: Experimental Comparison of UWM and UND Simulators**

Results from the simulation models described in section 4.1 were compared against similar data obtained for UWM’s simulation of their system. Comparisons were made against the stand-alone microsource (ms), externals storage (es), and genset models.

Simulation results for stand-alone components closely followed the UWM test results. Test results from ND’s simulation are shown in the following plots. Simulation results from the simulated mesh microgrid closely followed that of the three test cases provided by UWM. These test cases are itemized below. With each case, we show the “original” results from UWM’s hardware validated simulation against the “comparison” results for UND’s simulation.

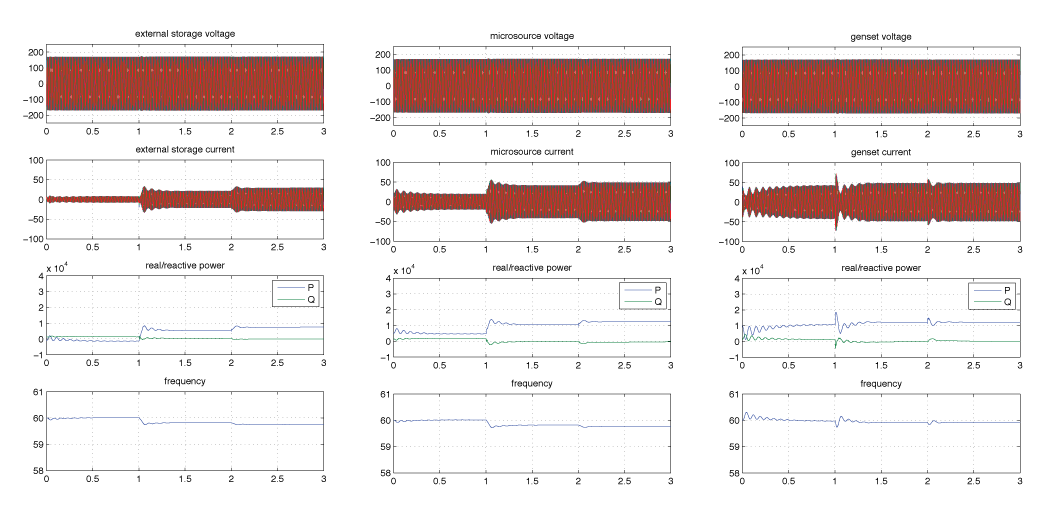
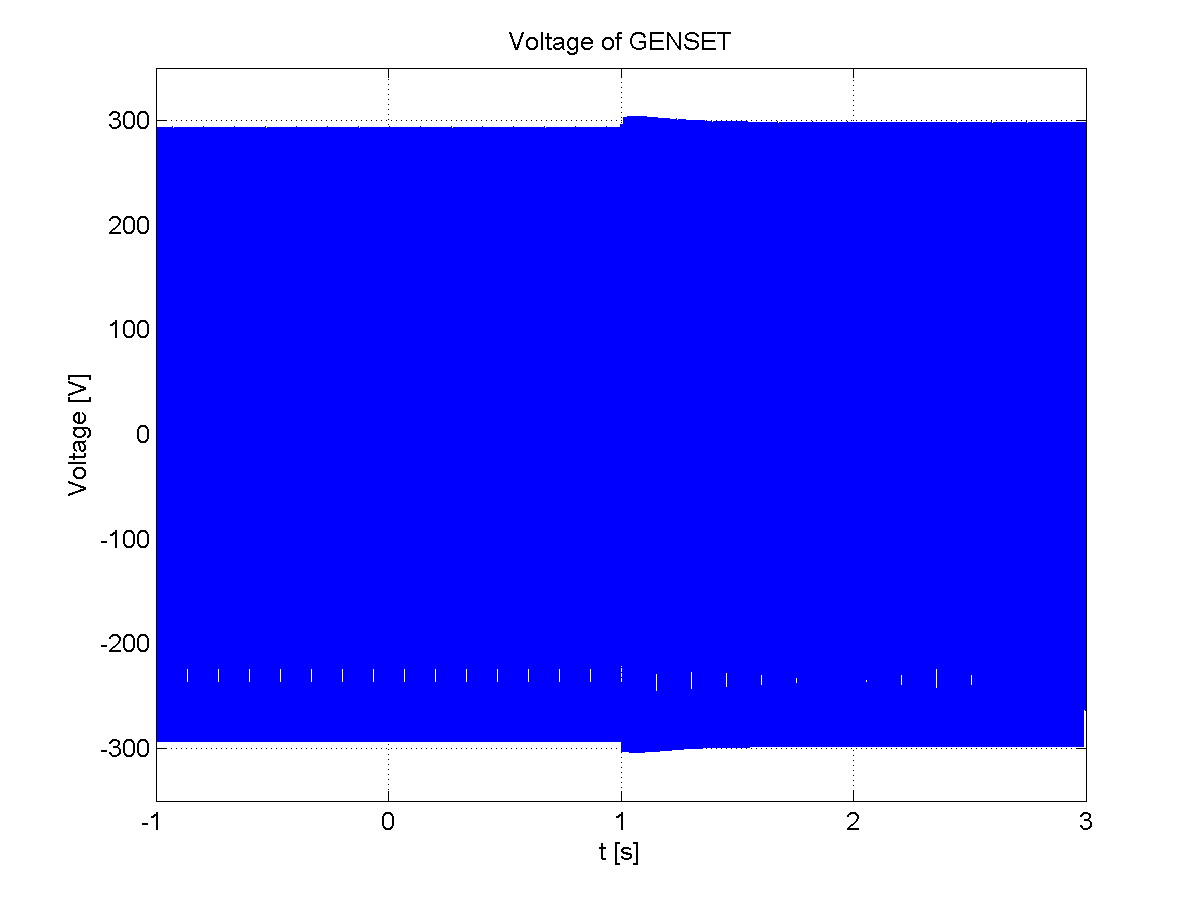
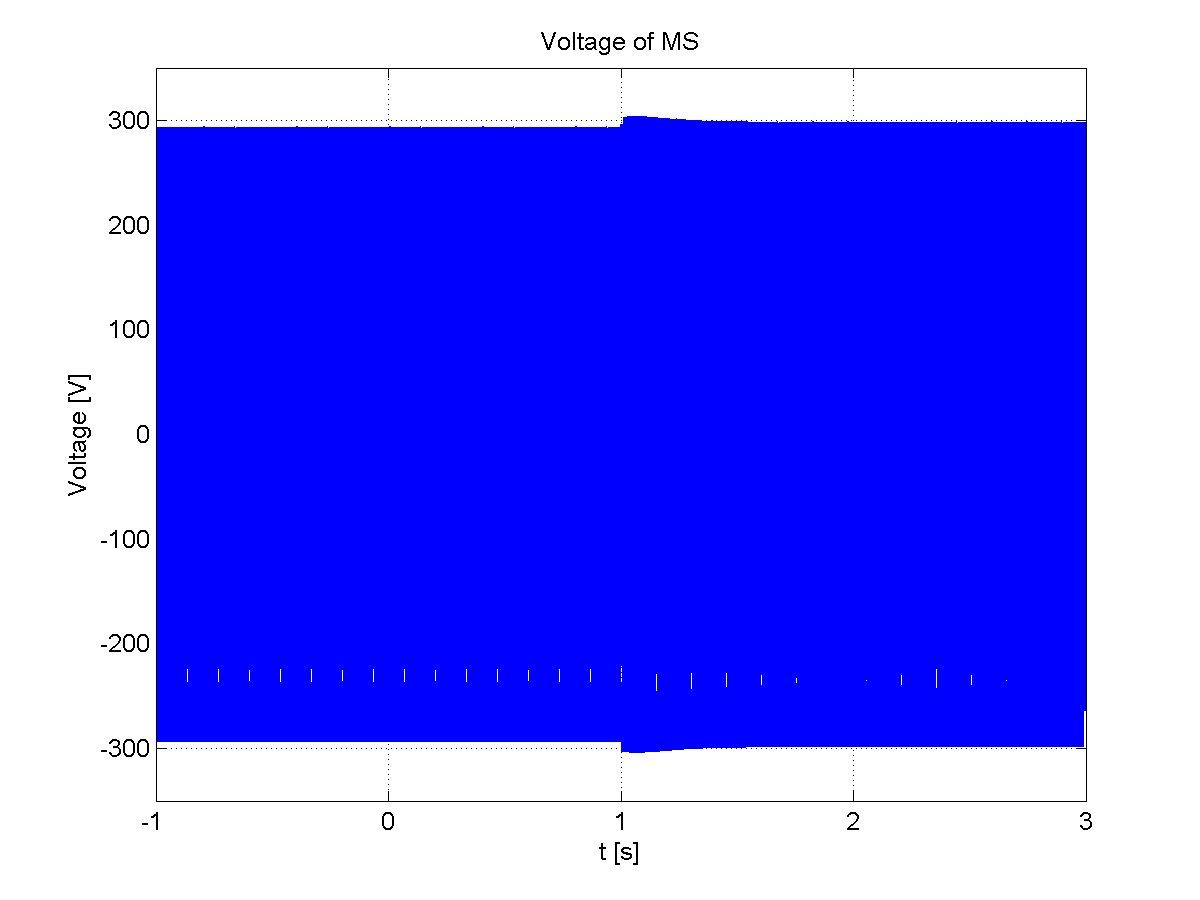
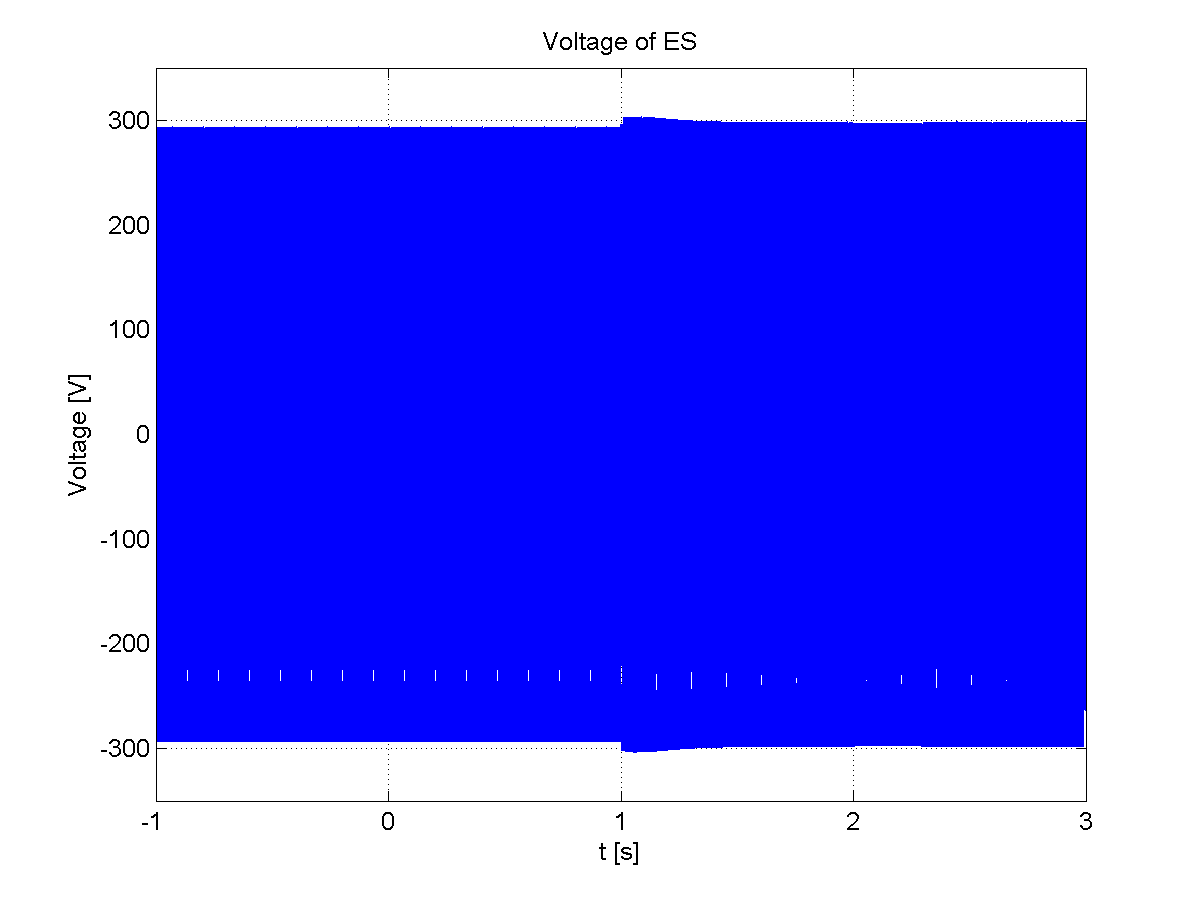
Case 1) initial 28 kW load with 18 kW imported from grid:  
 islanding at t = 1 second  
 additional 4kW lo ad at t = 2.0 seconds

Figure : Comparison results for UND simulator, case 1



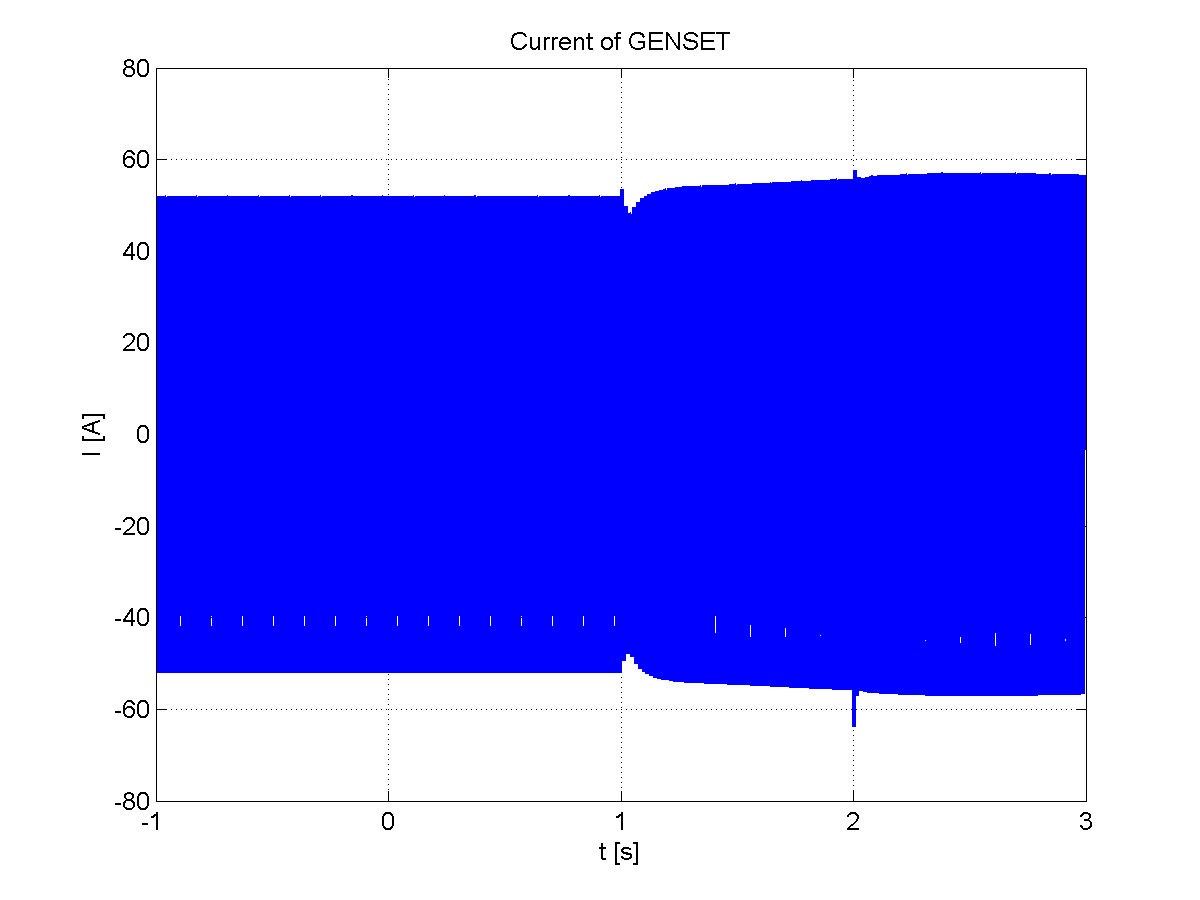
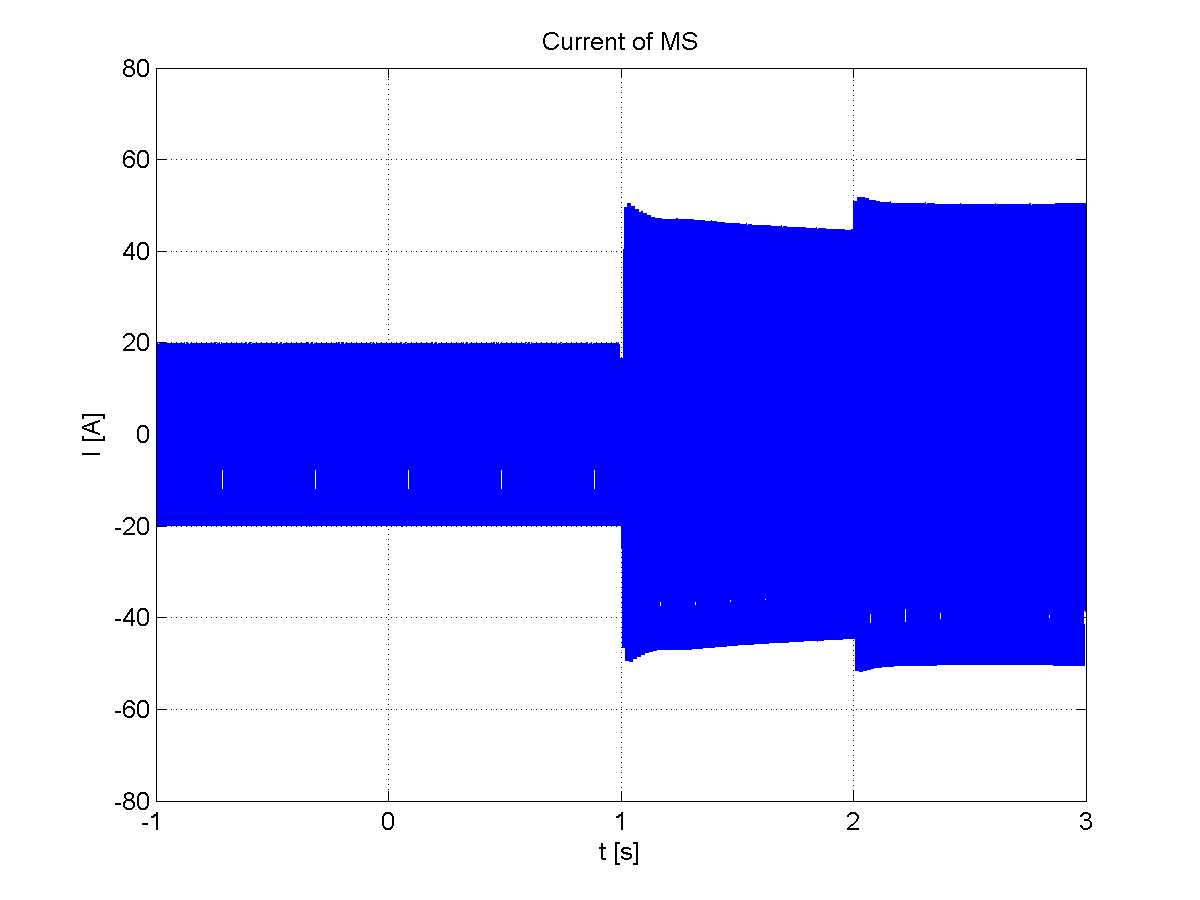
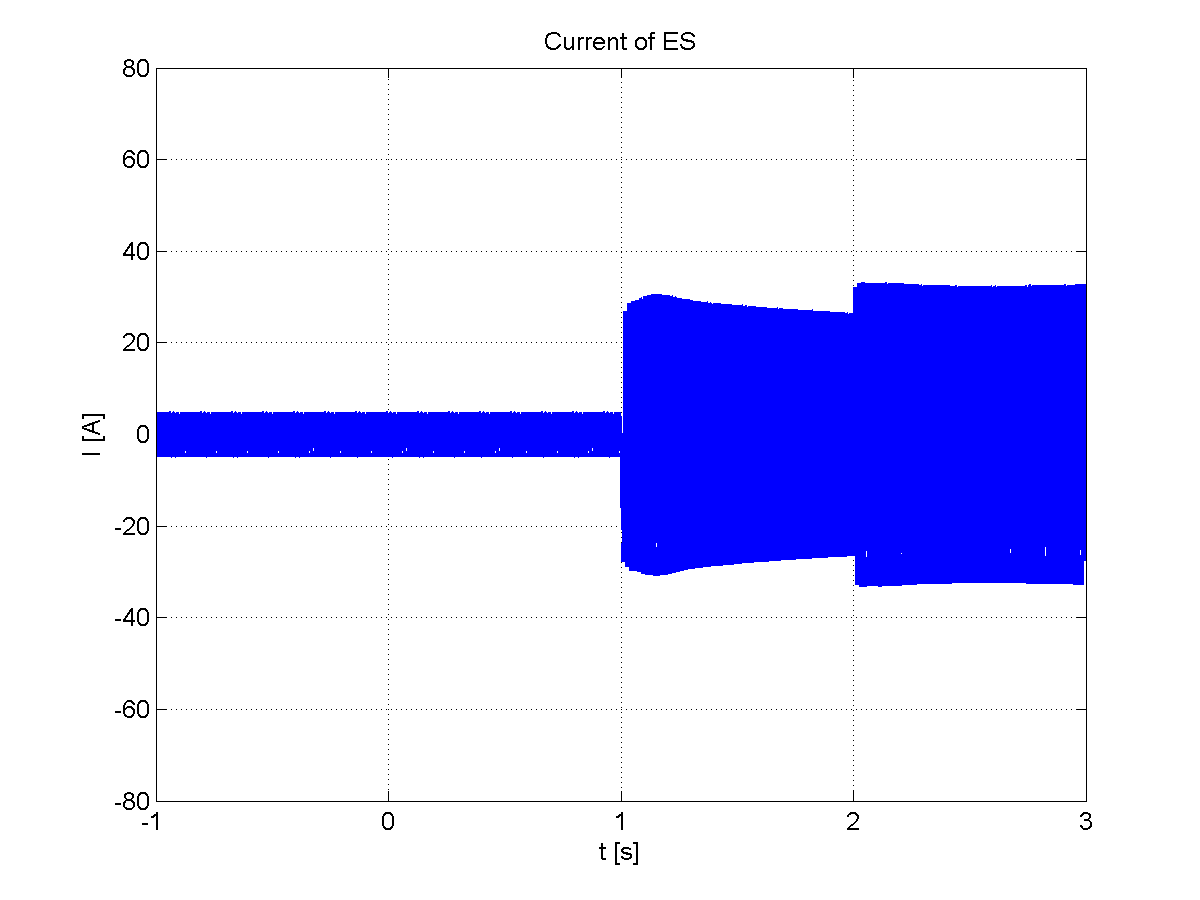
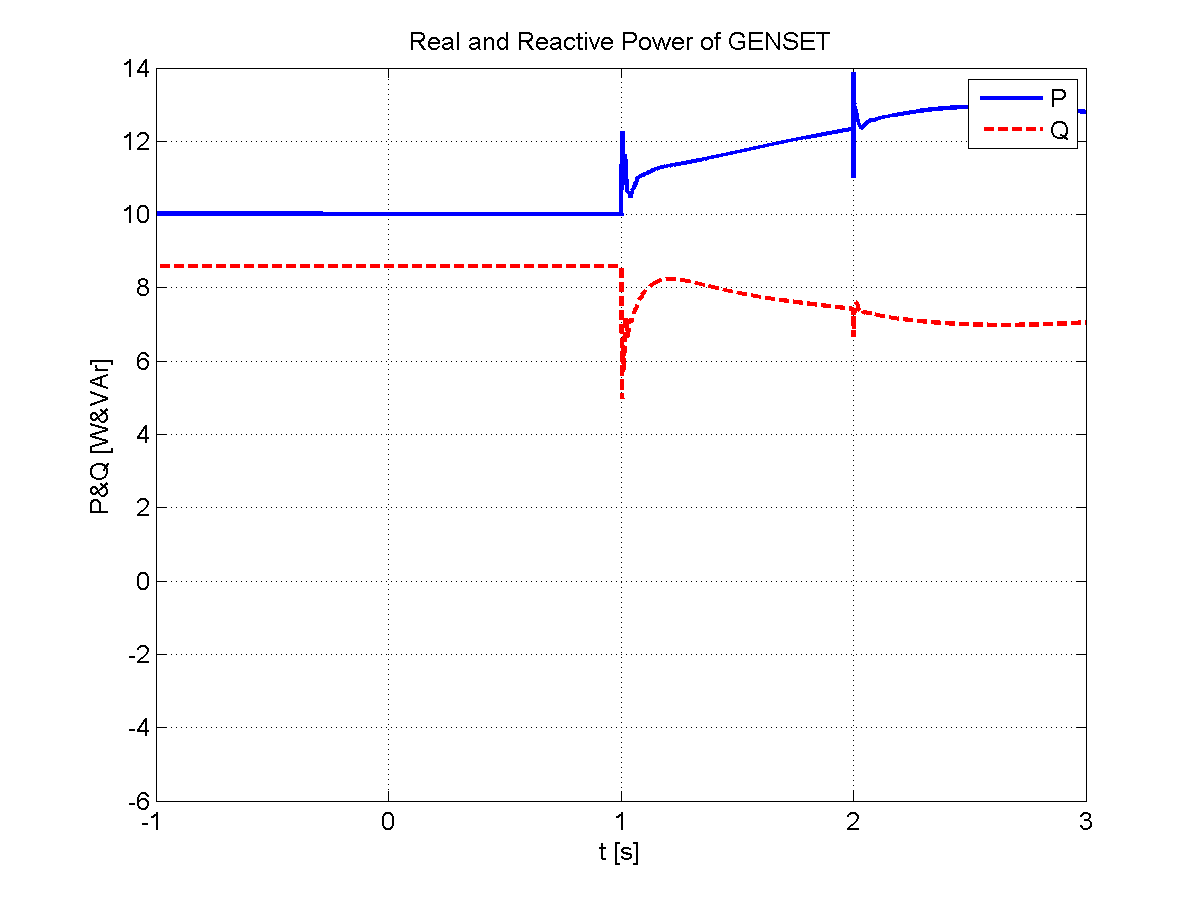
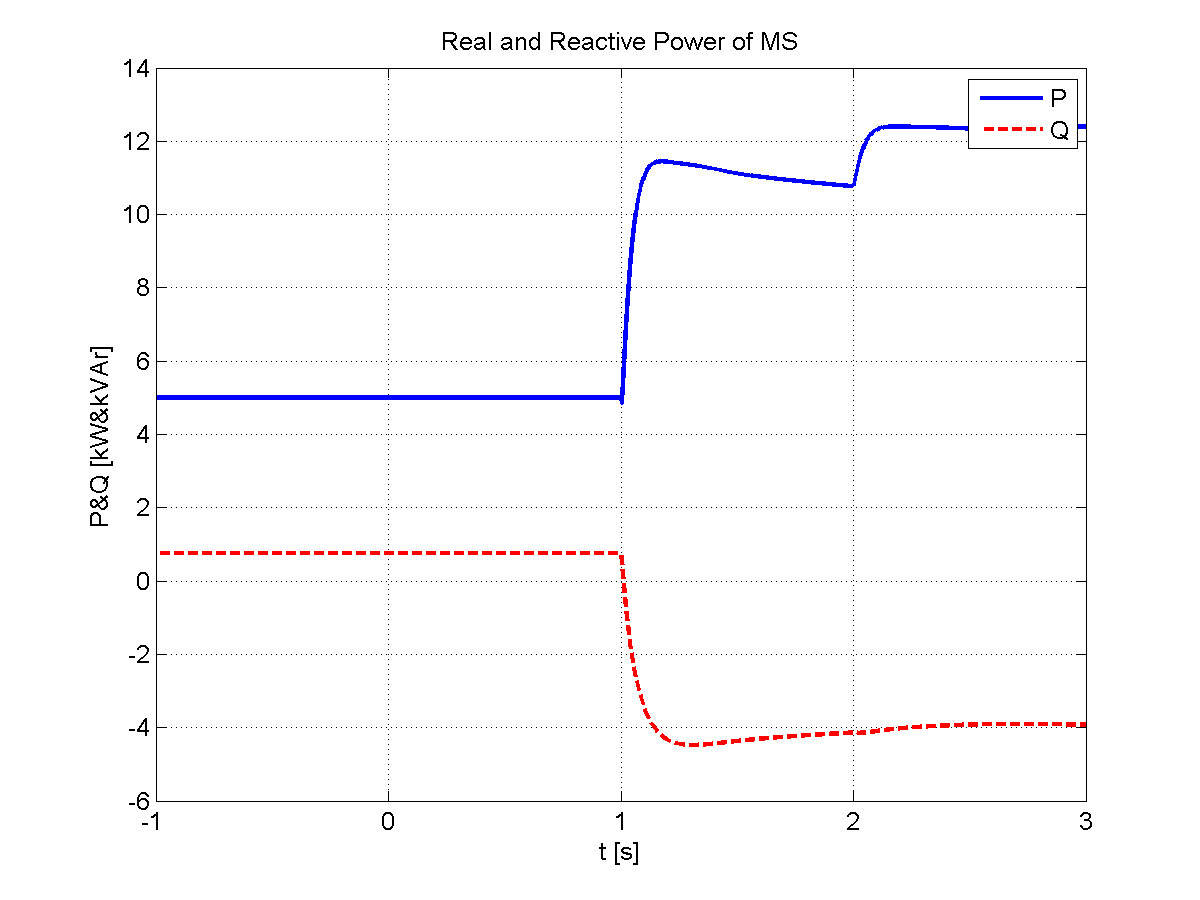
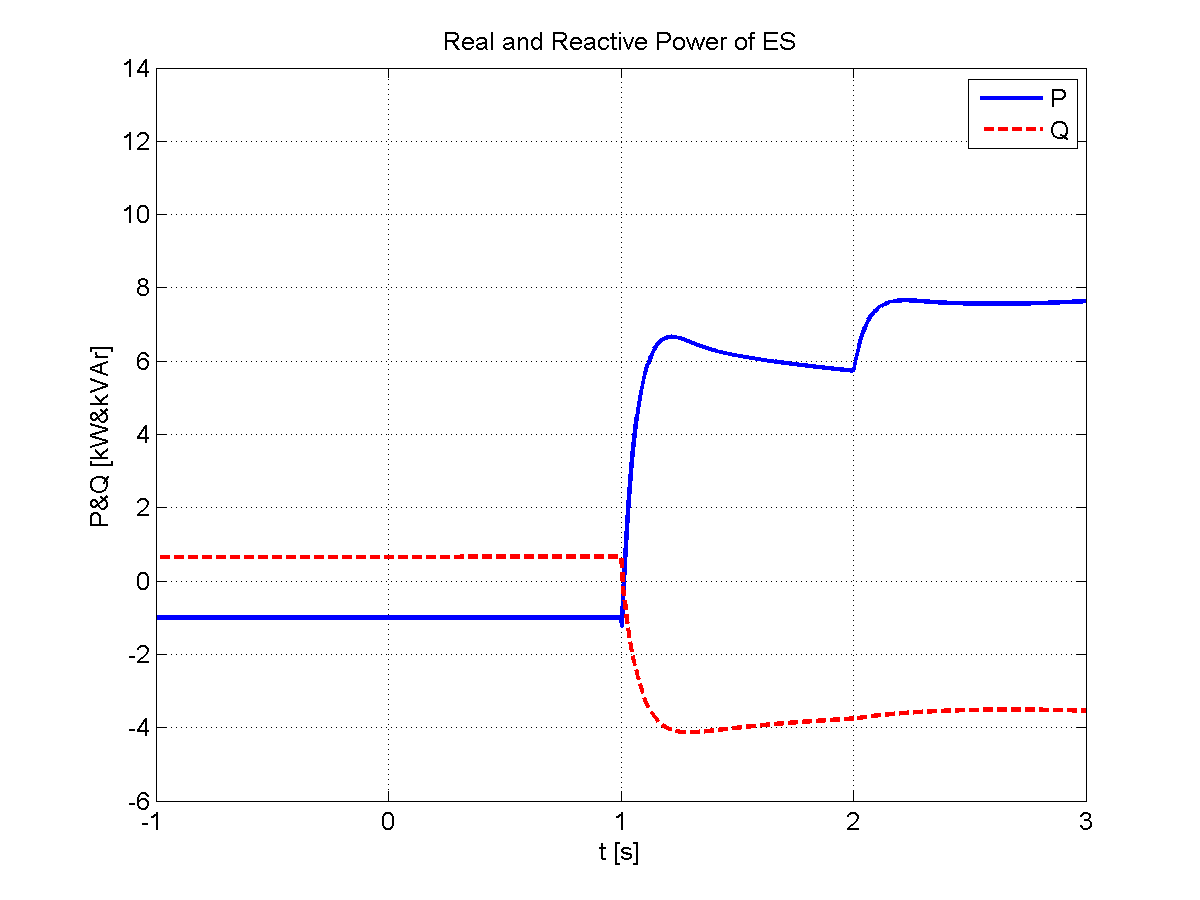
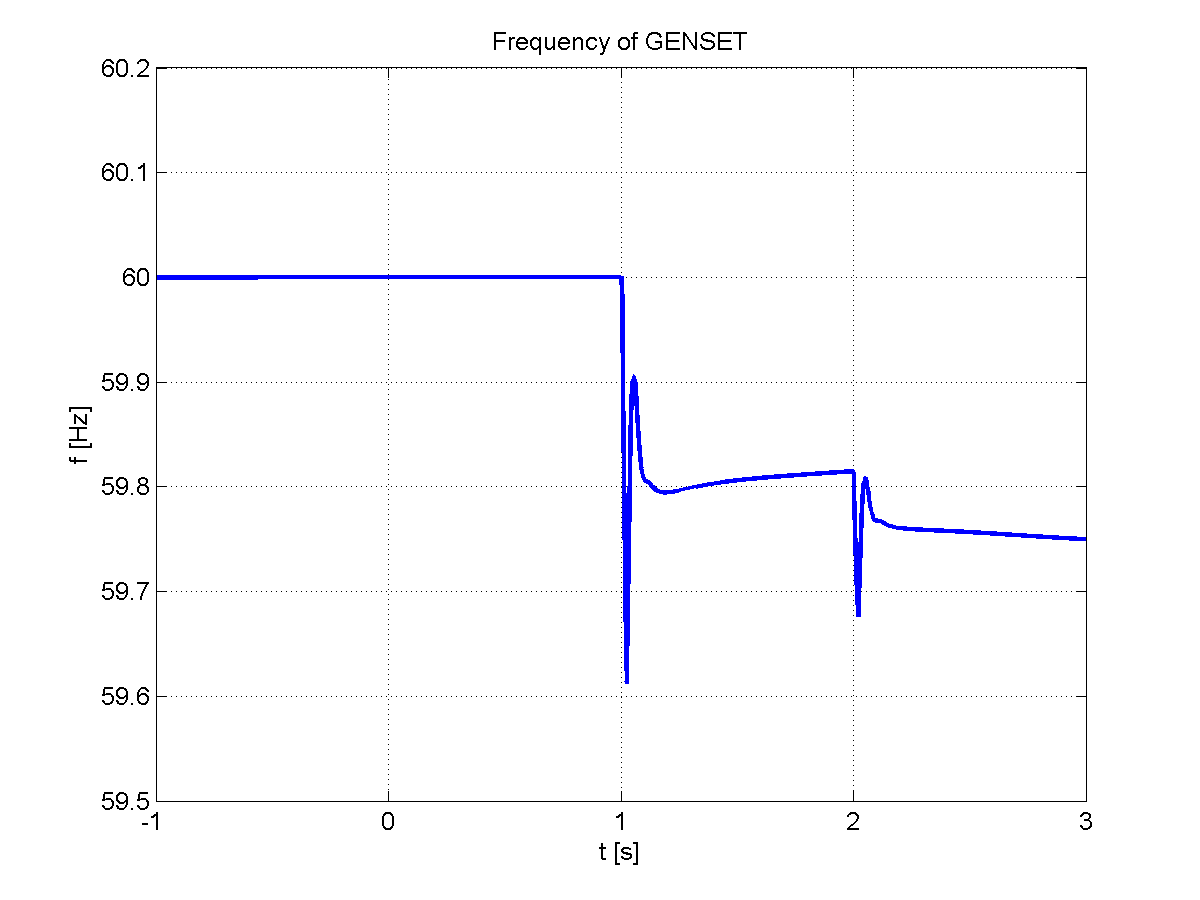
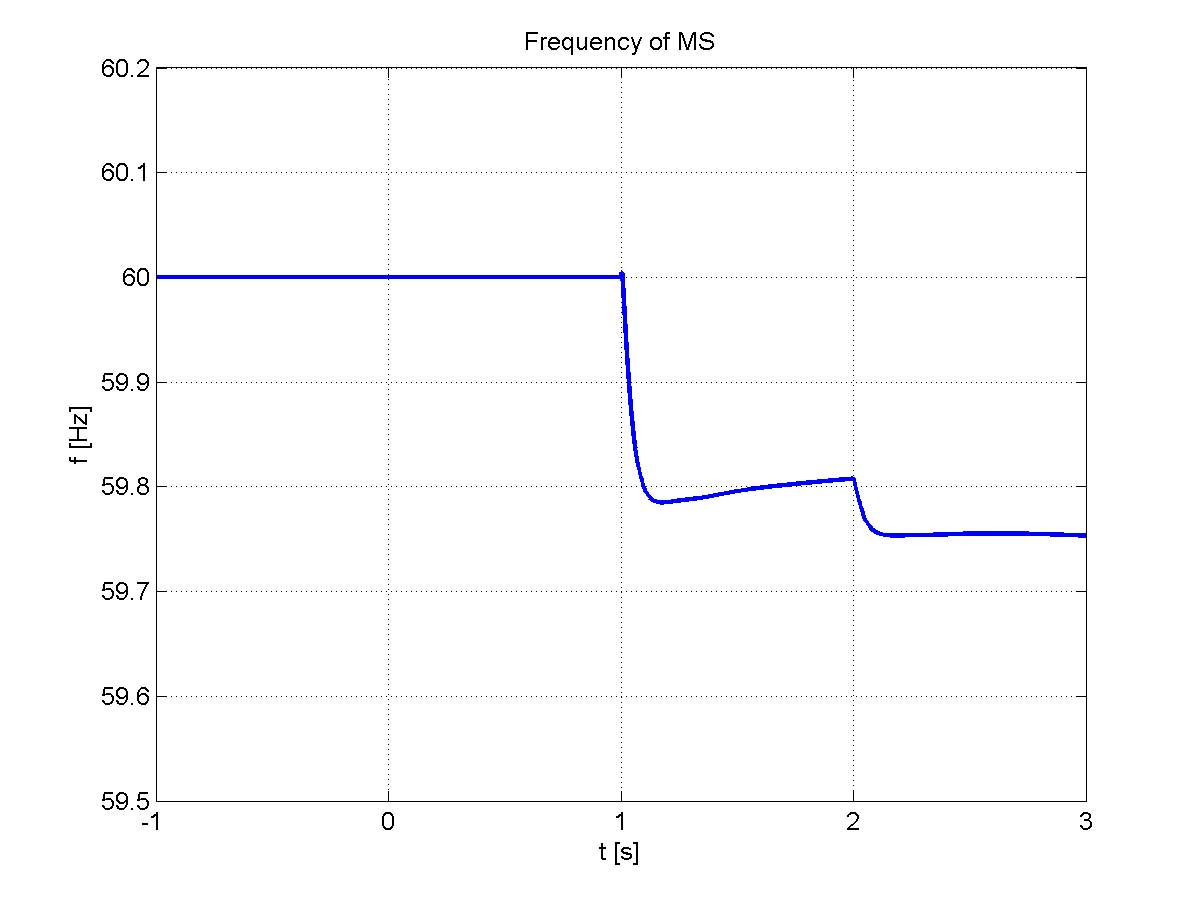
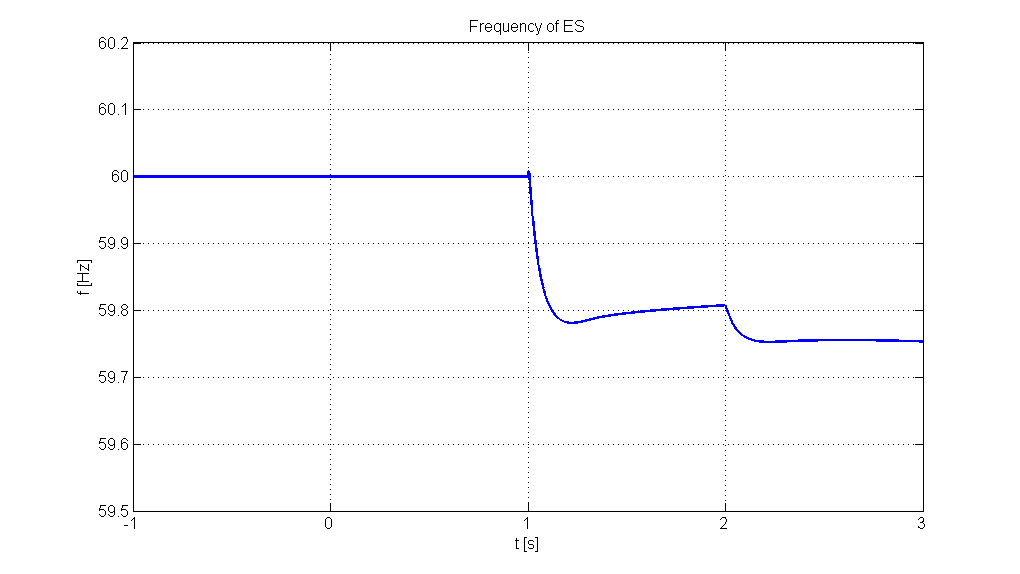


Figure 15: original results from UWM simulator, case 1





Case 2) islanded with 32 kW loading  
 open cable (PC3) at t=2 seconds

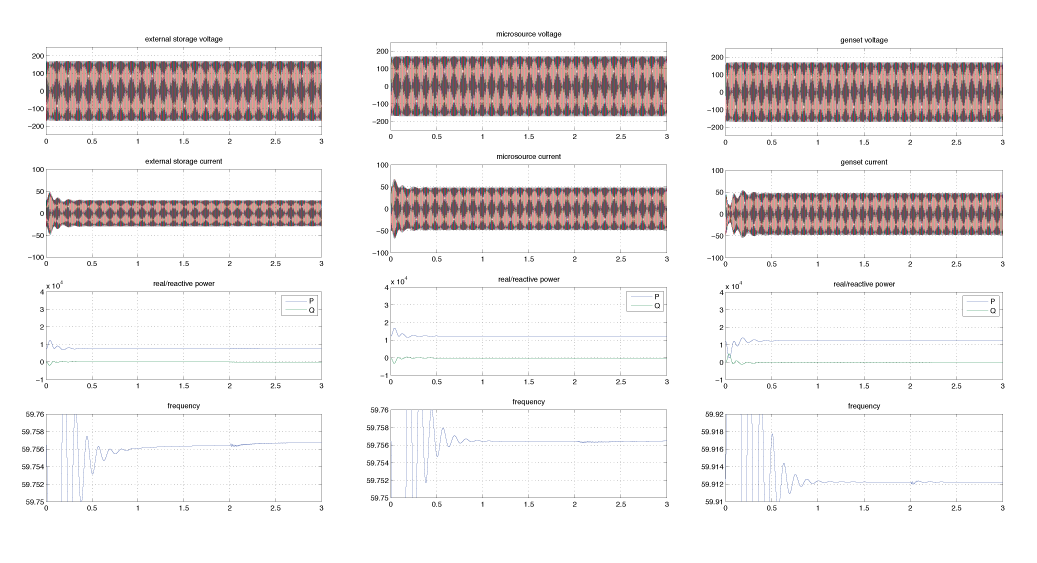
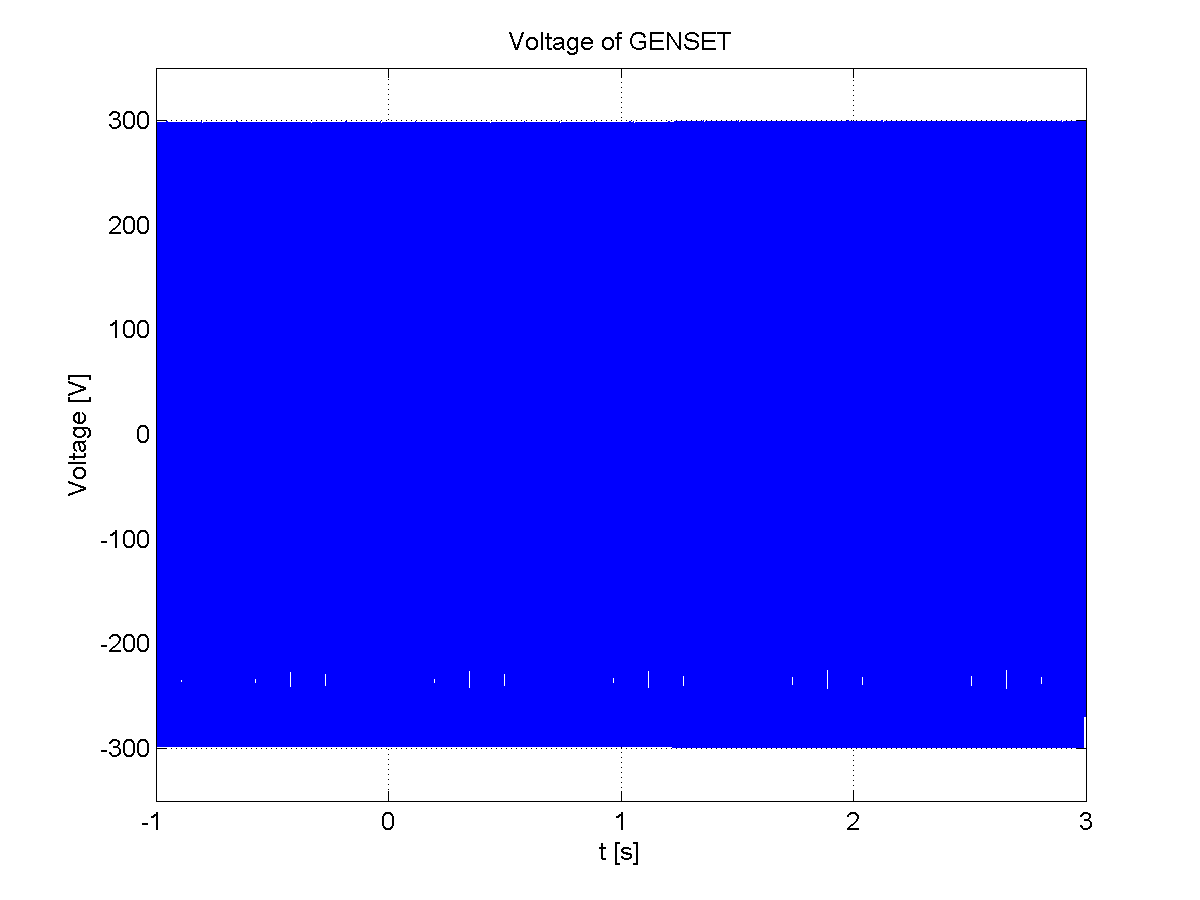
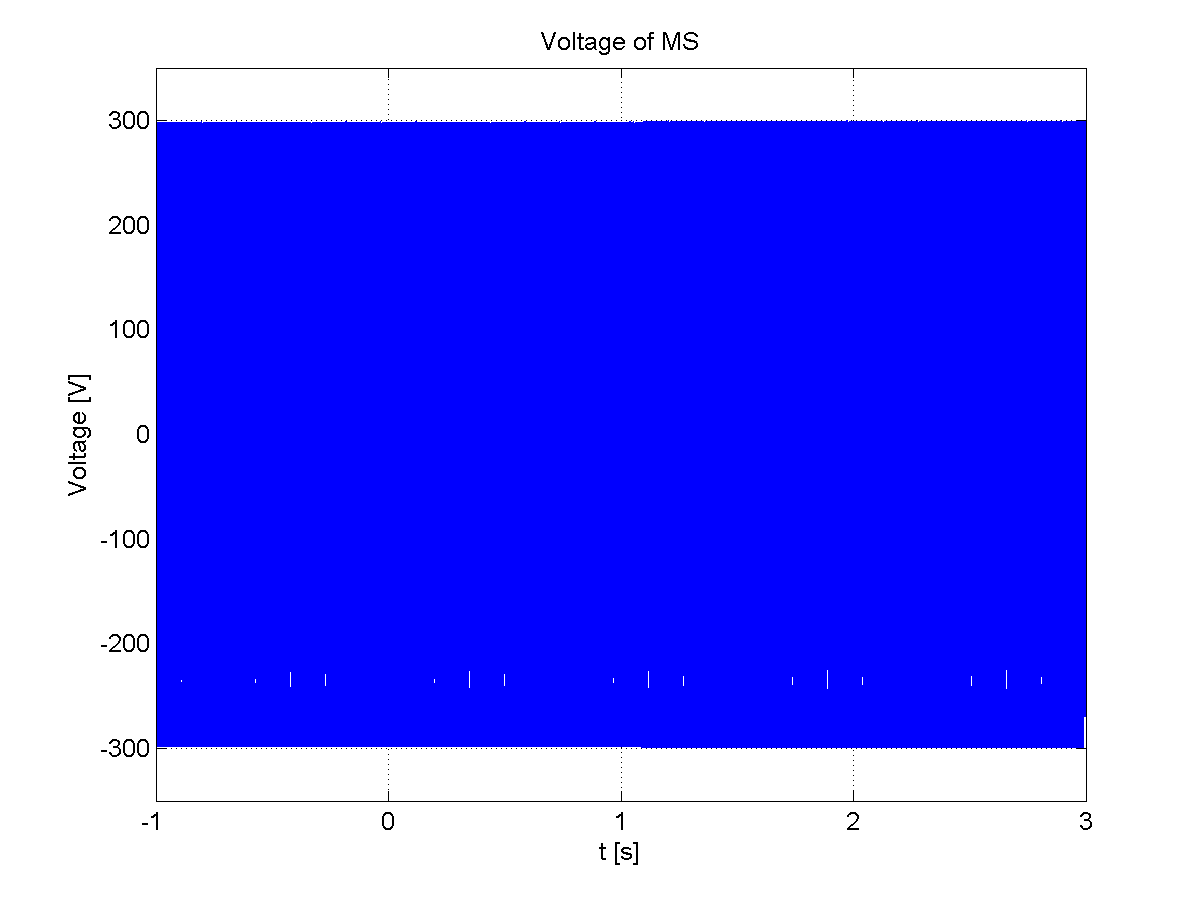
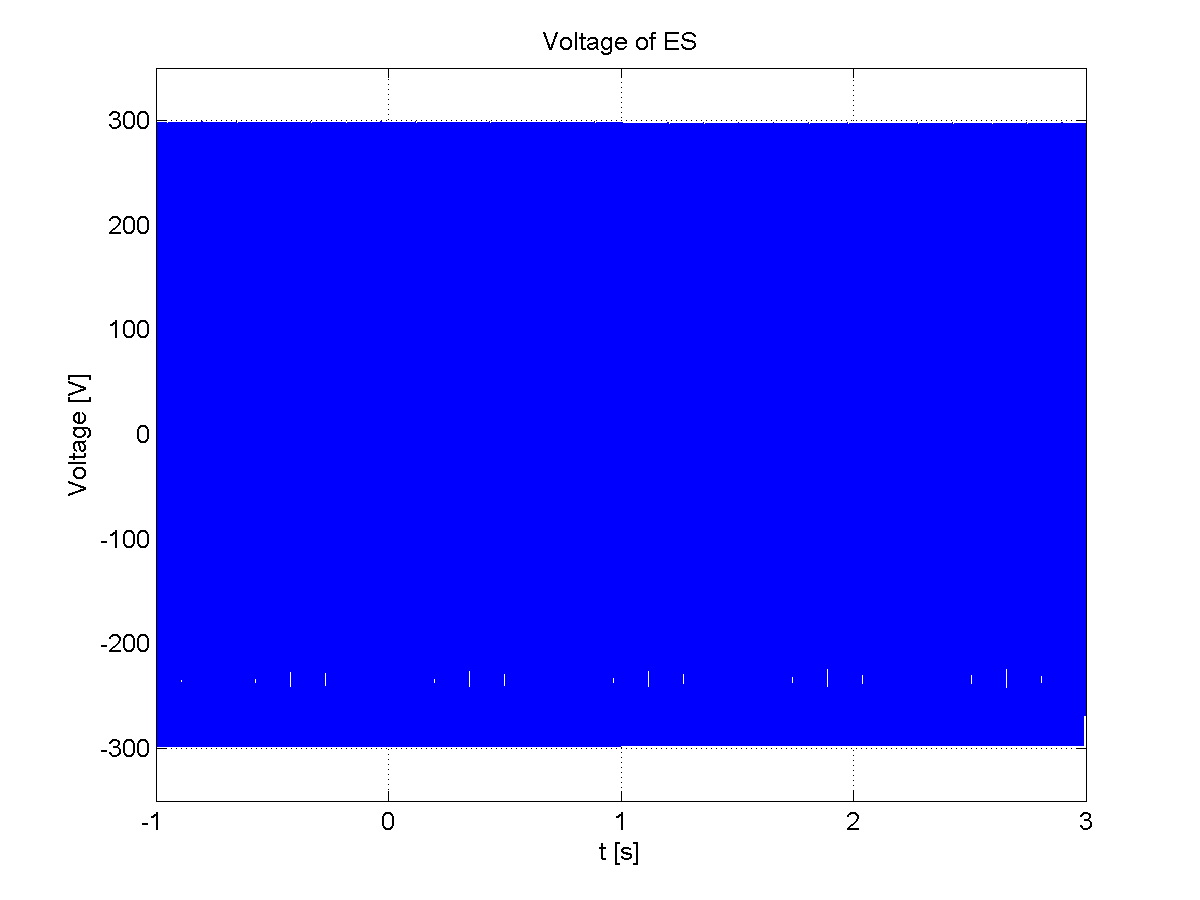


Figure 16: Comparison results for UND simulator, case 2



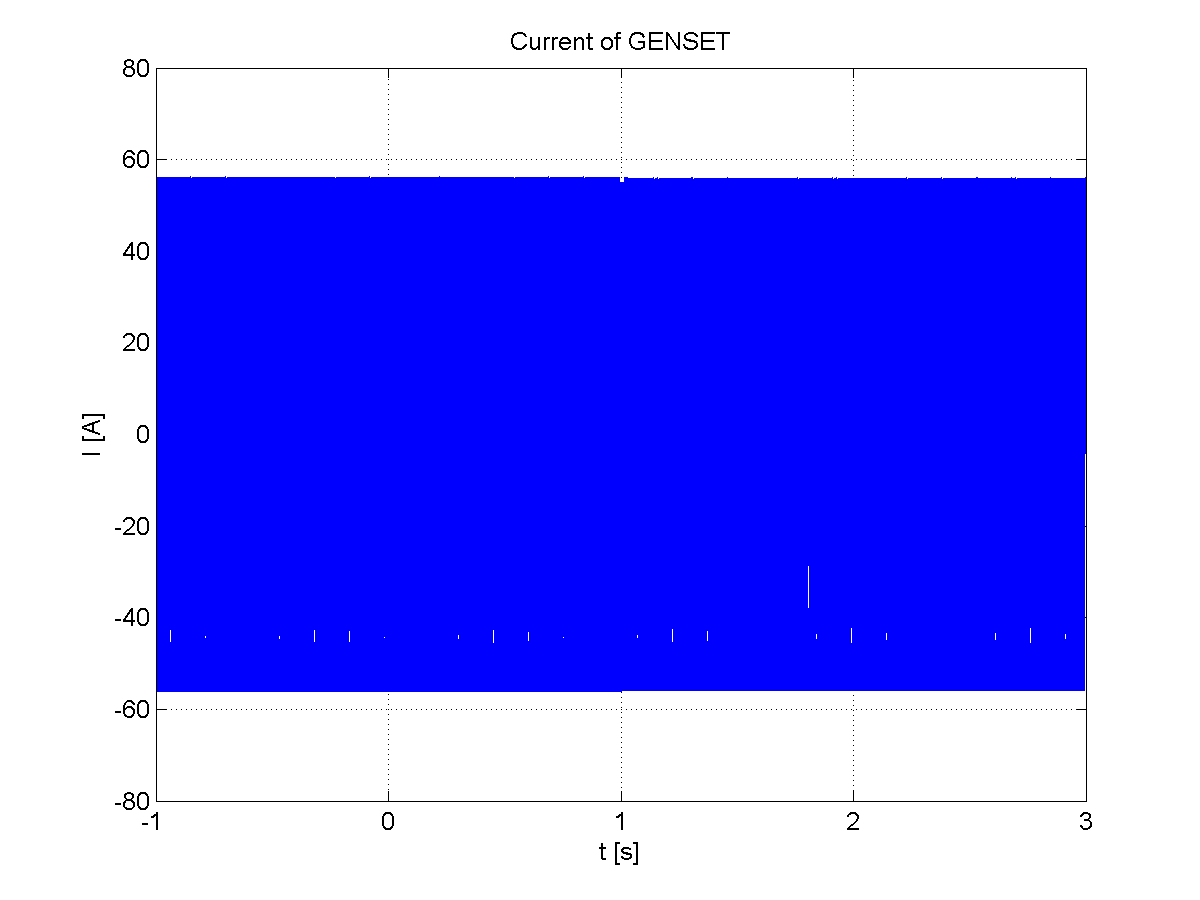
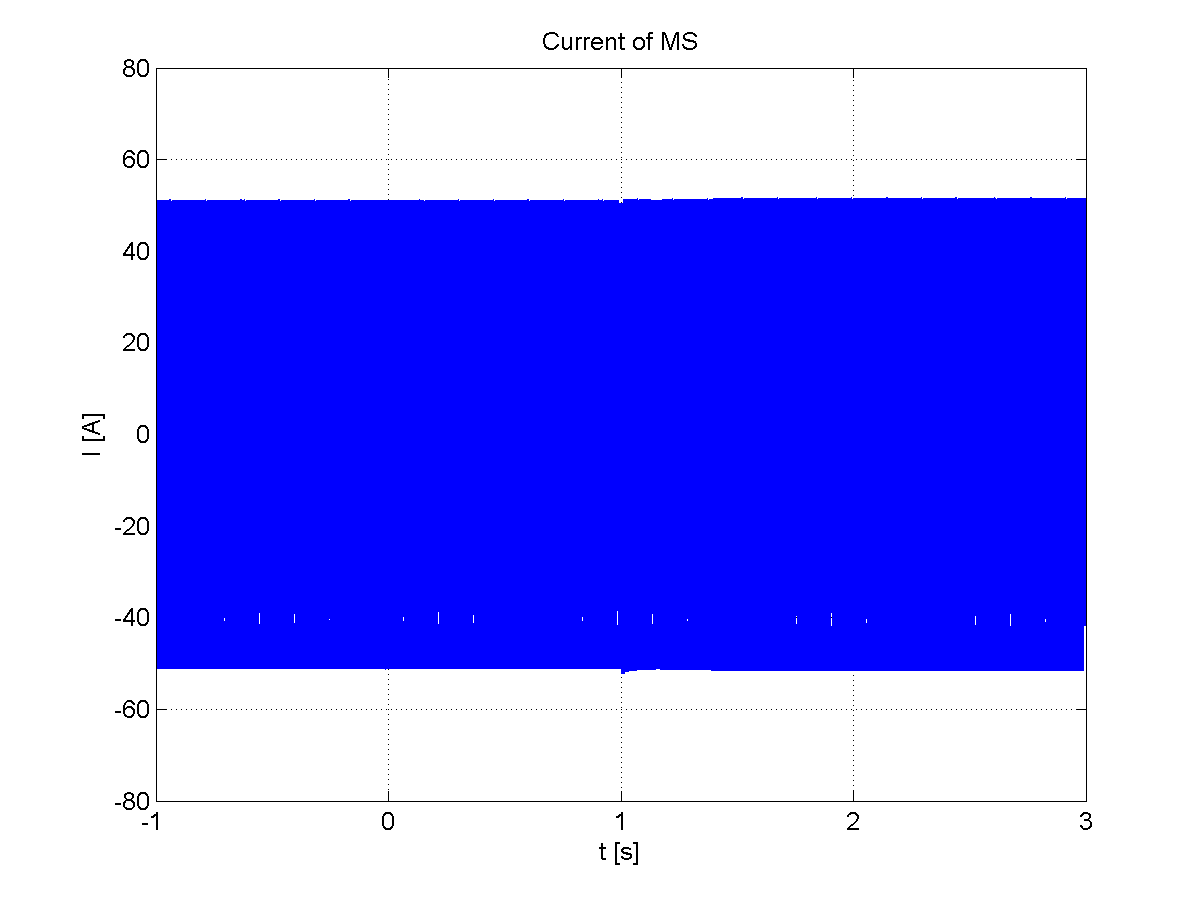
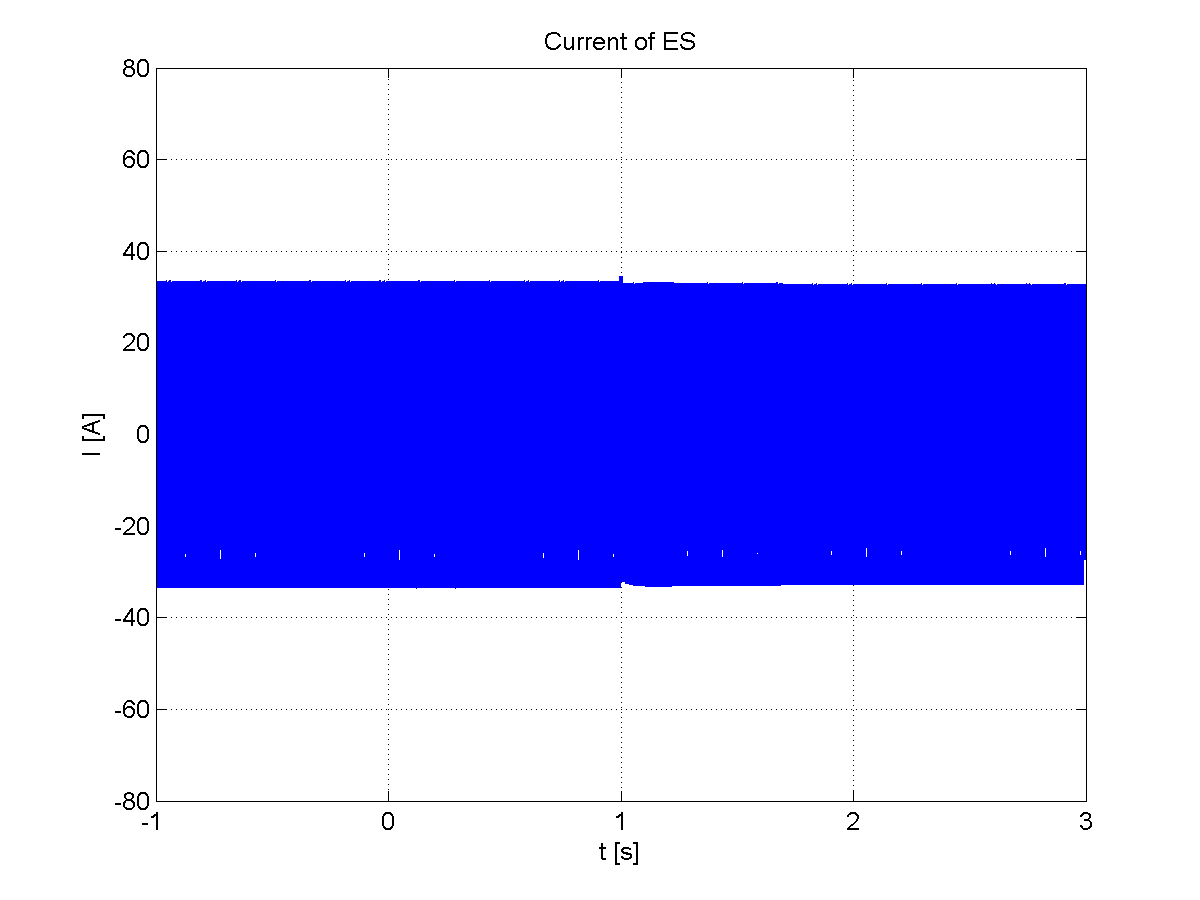
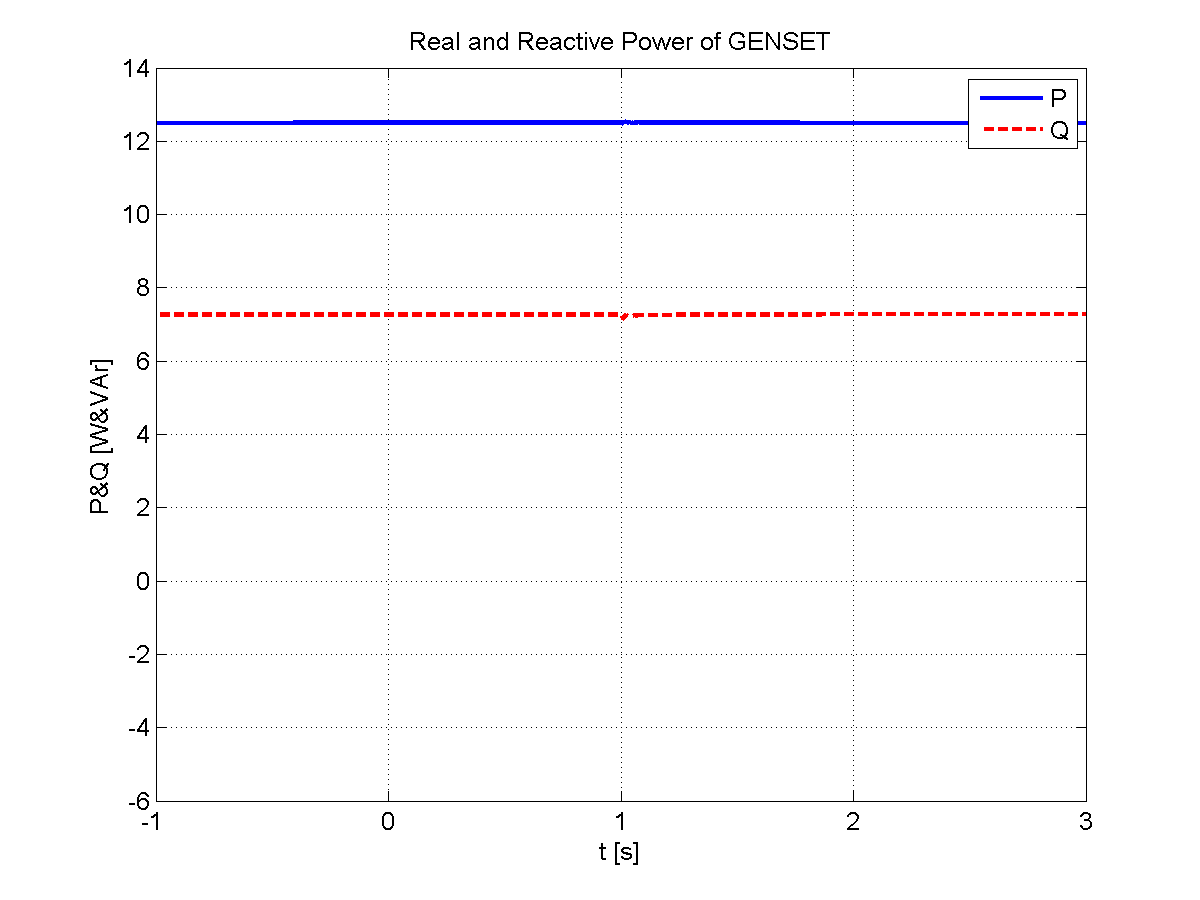
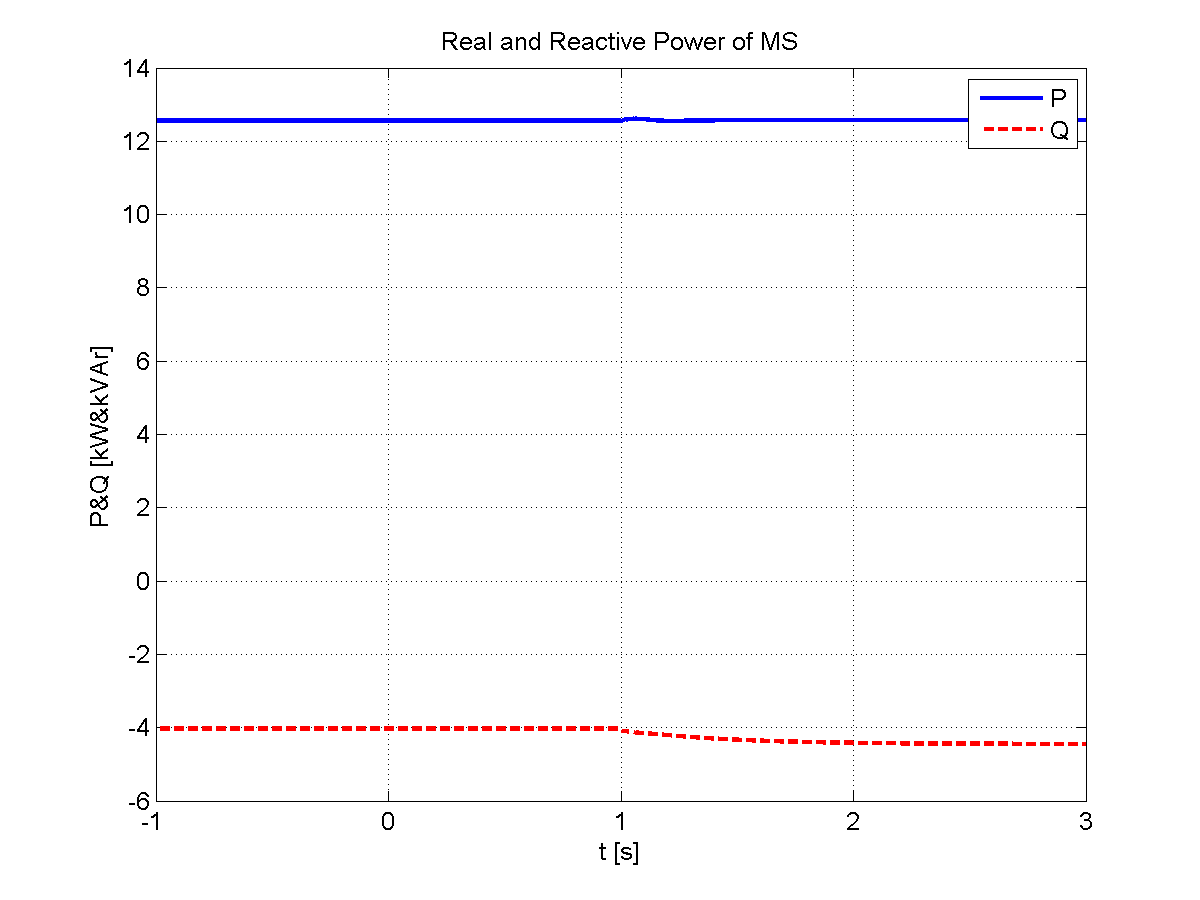
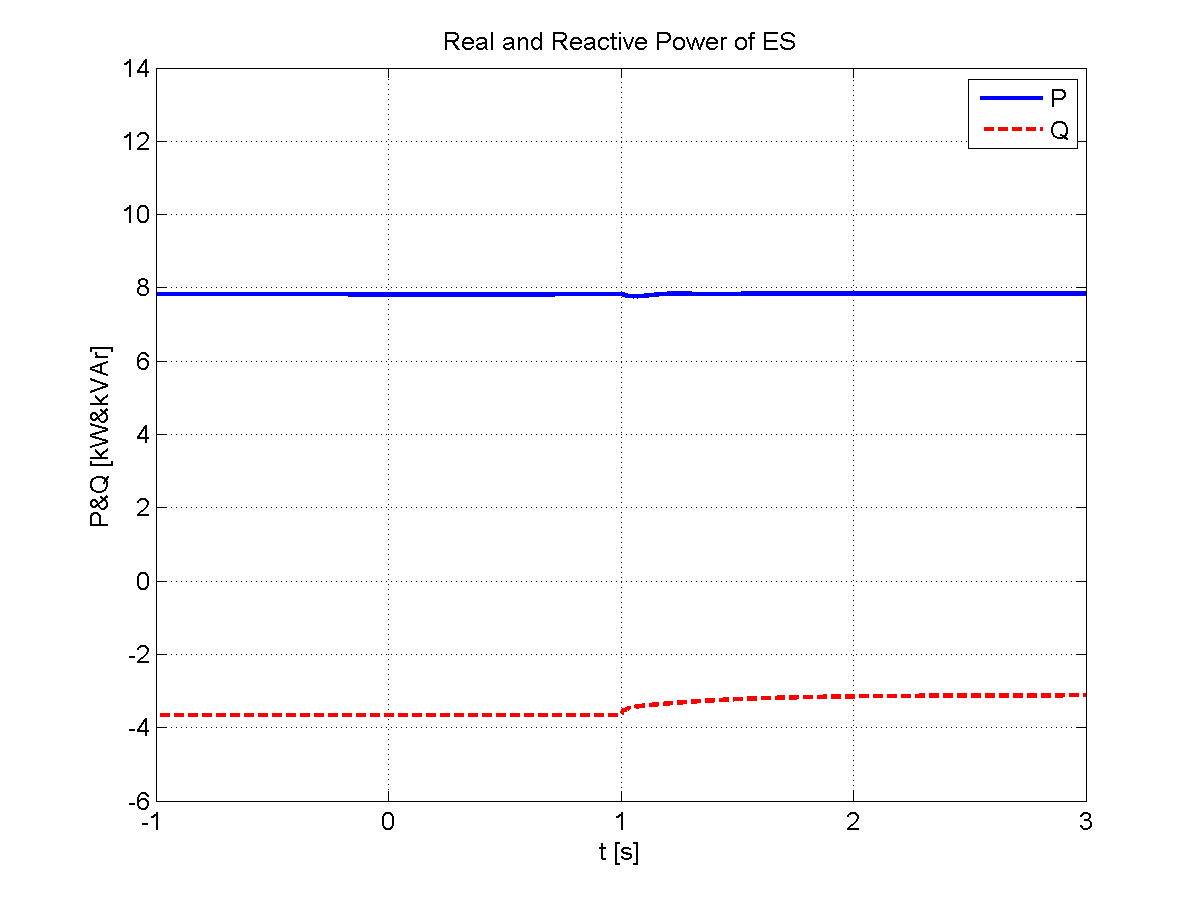
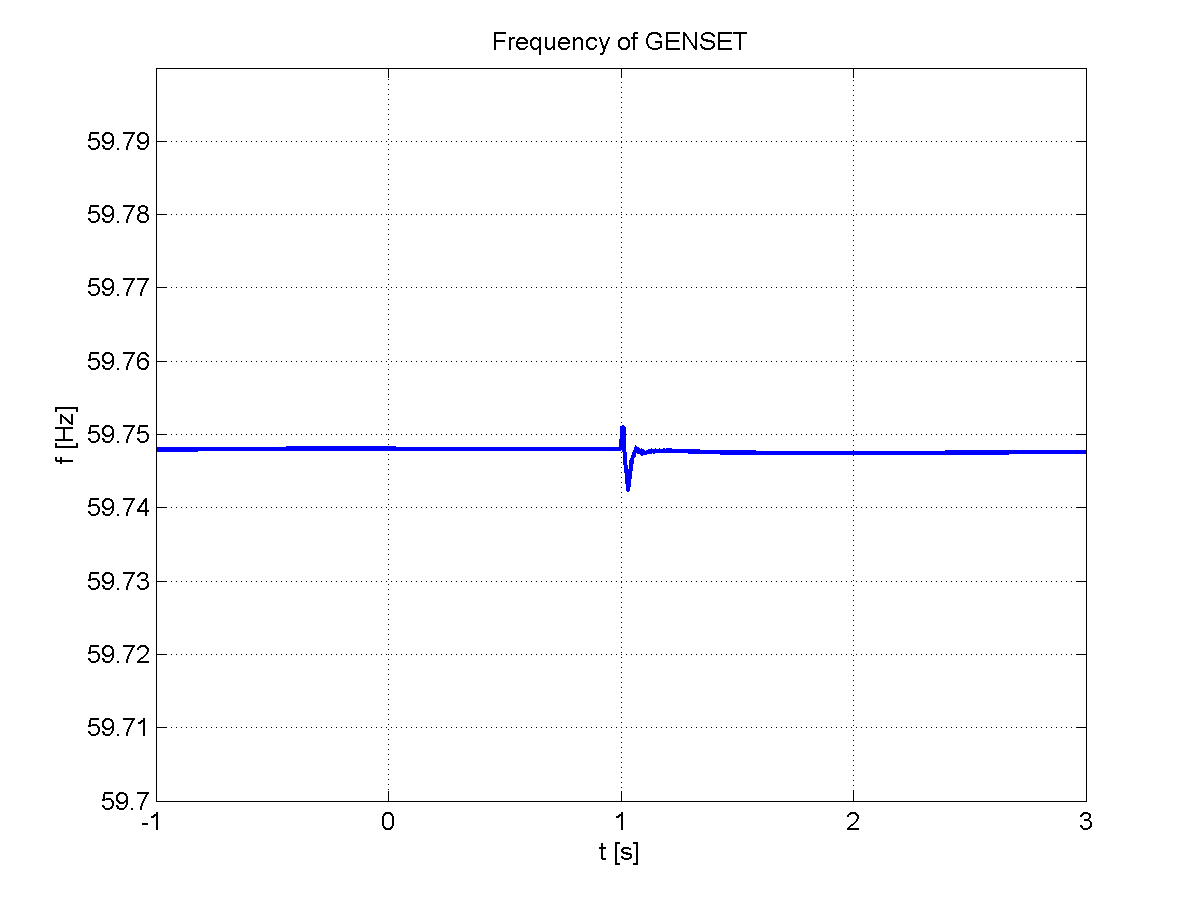
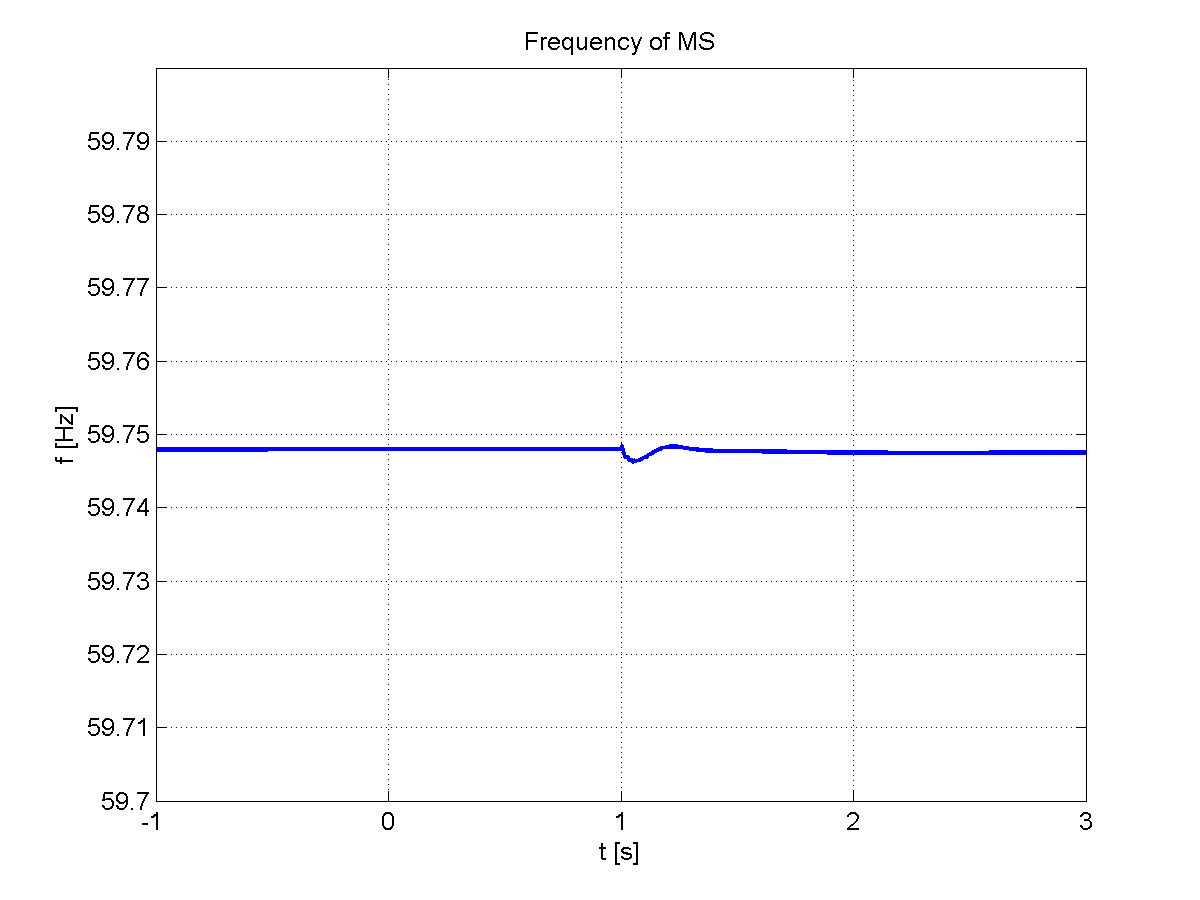
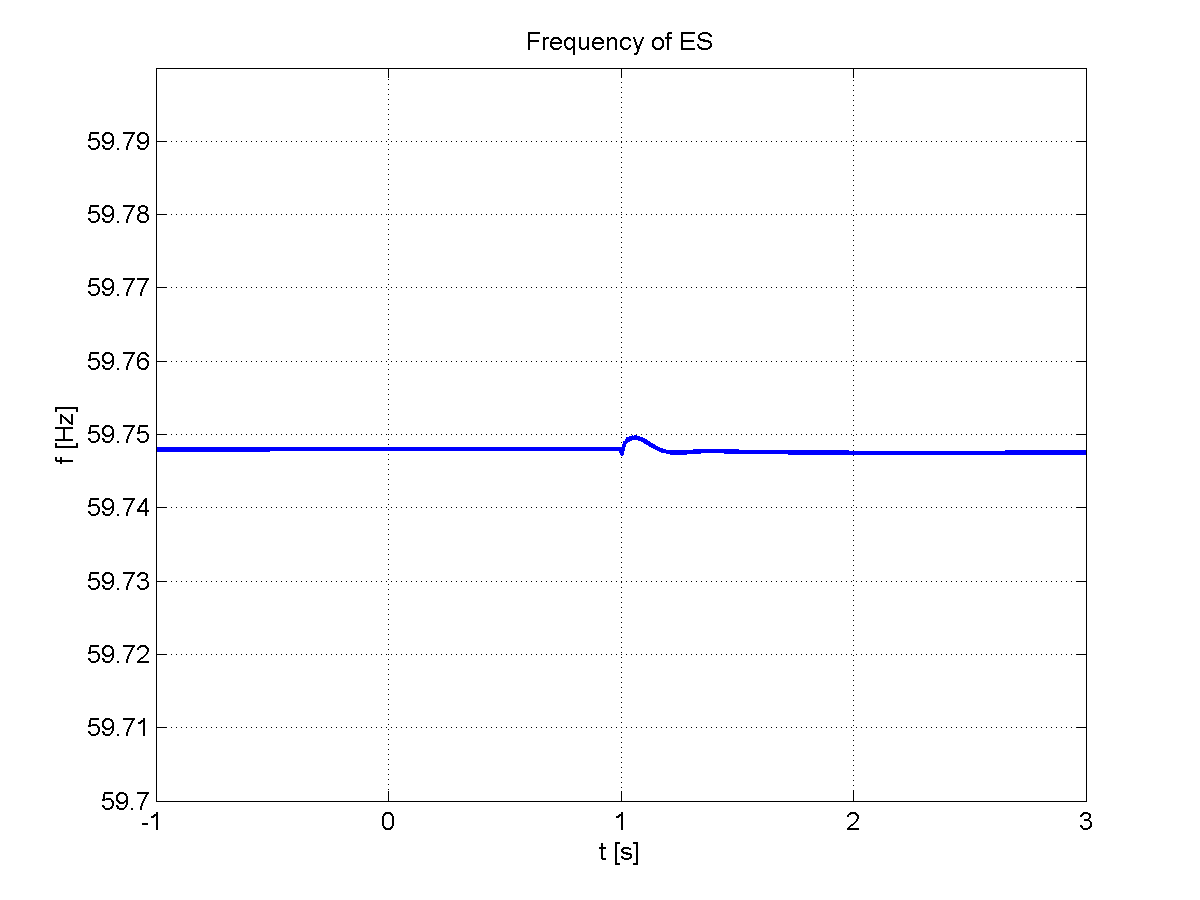


Figure 17: original results from UWM simulator, case 2





Case 3) Islanding at t=4 second with on-site 32 kW load

Genset lost at t=5 seconds

Low frequency trip at about t=6.5 seconds sheds 16 kW on bus 1 and 3

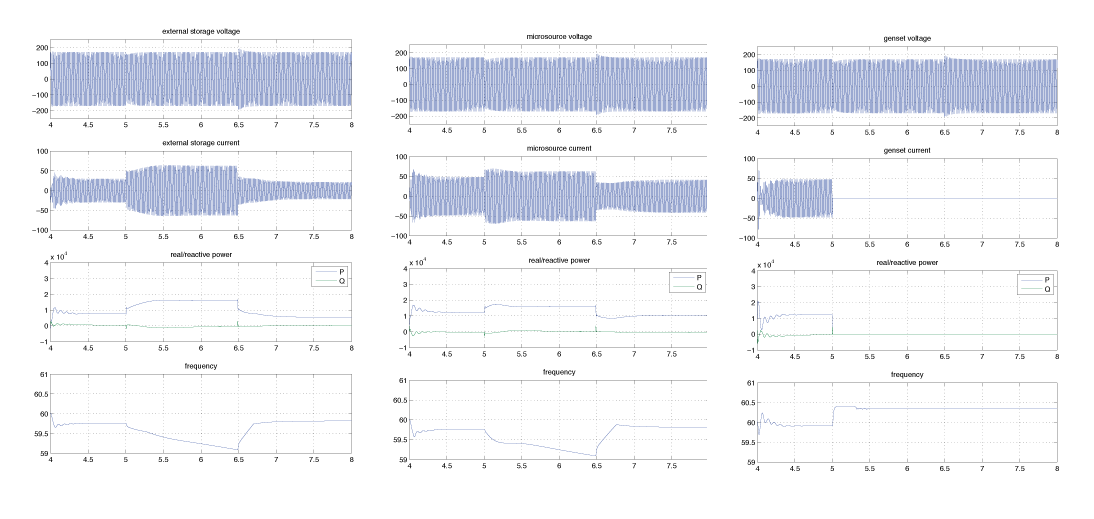
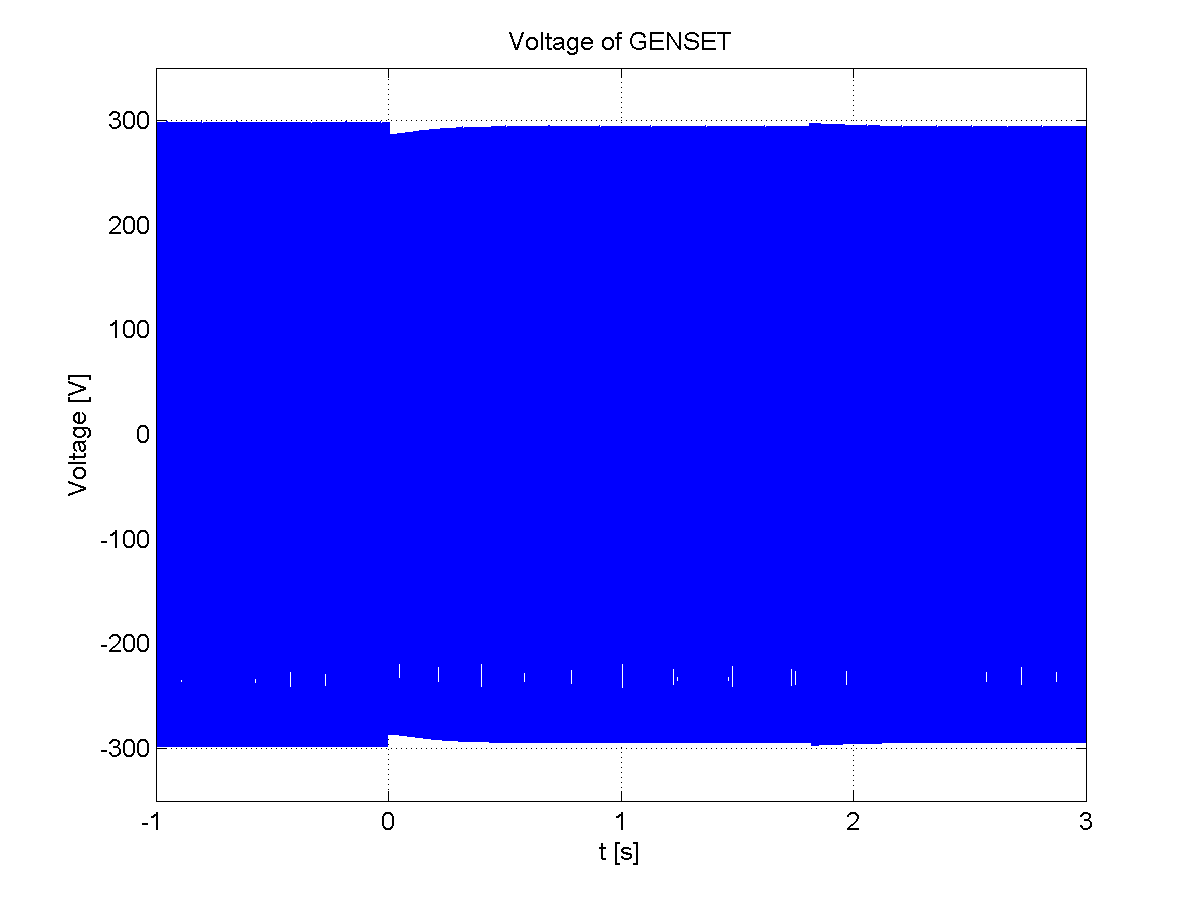
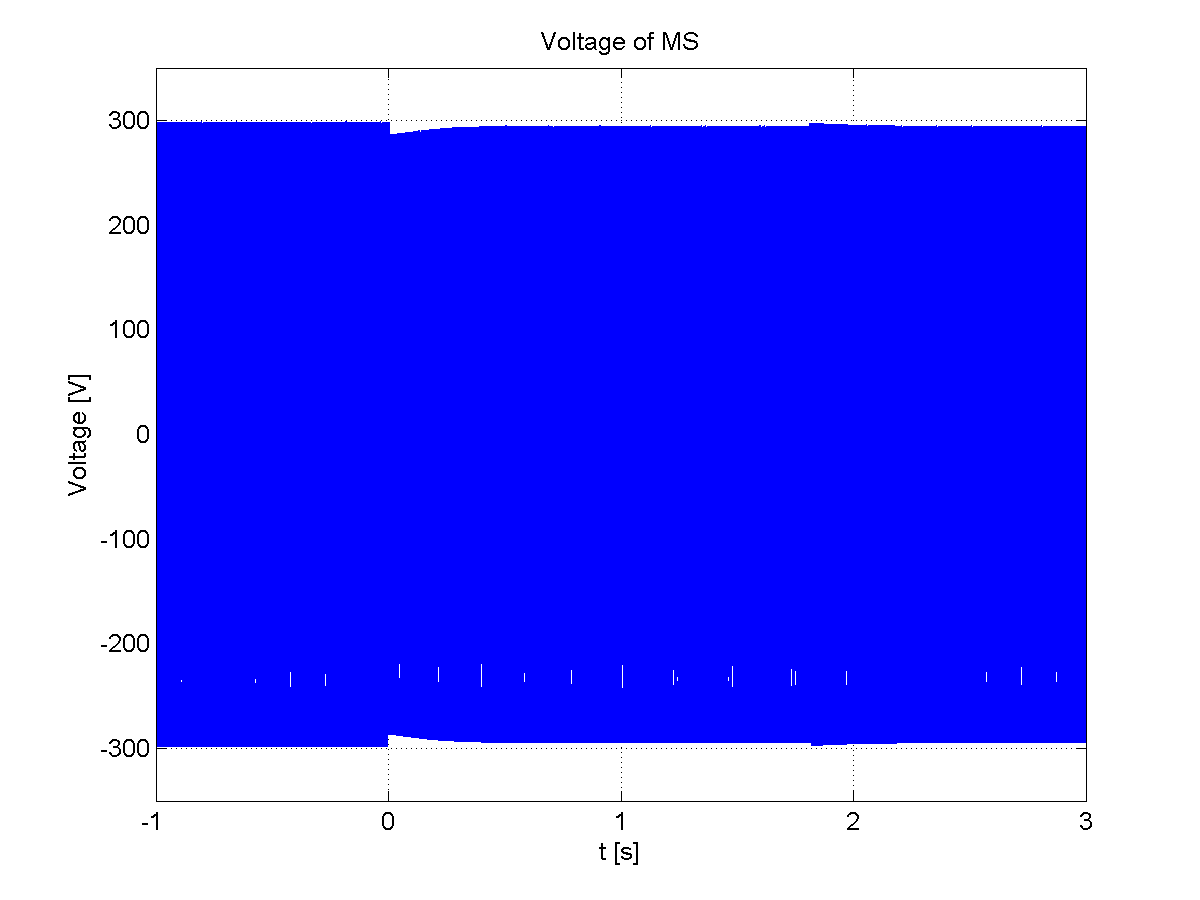
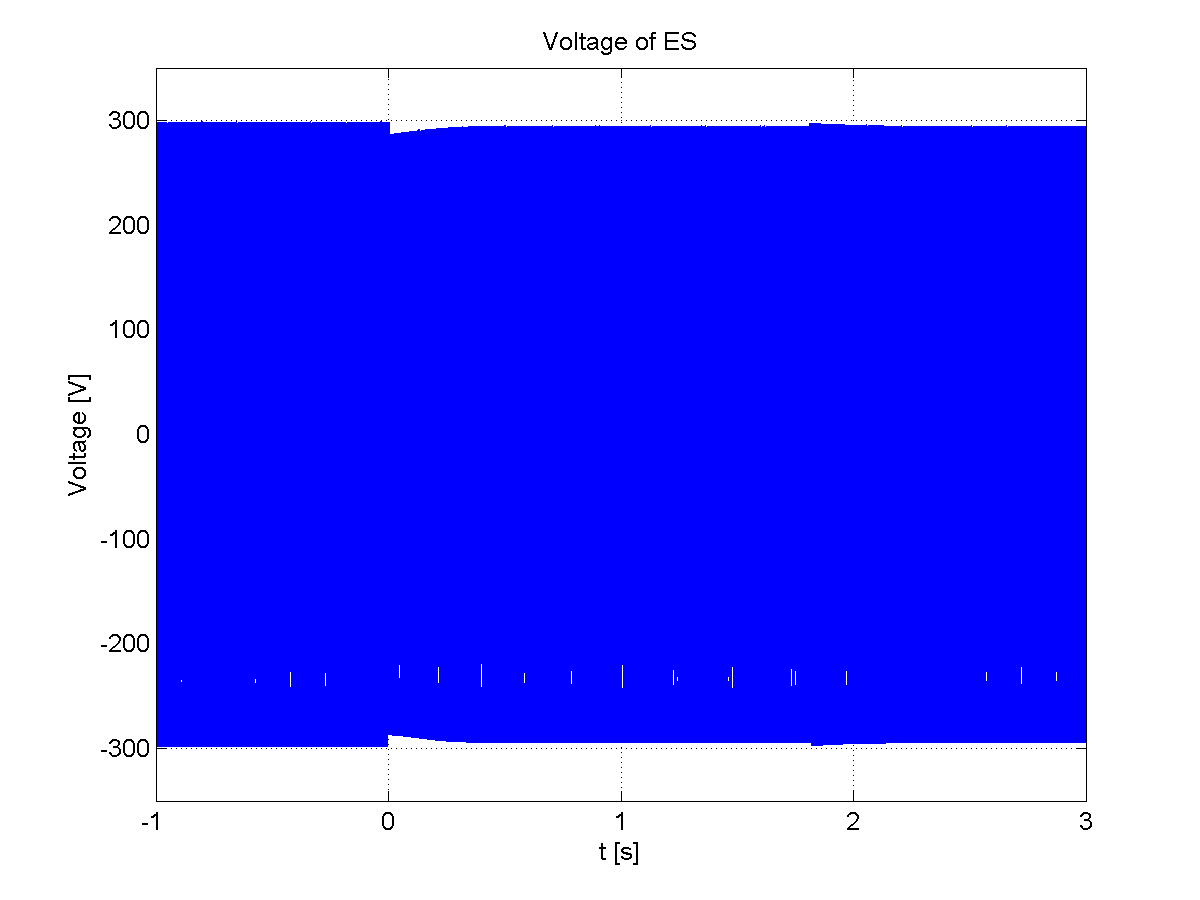


Figure 18: Comparison results for UND simulator, case 3



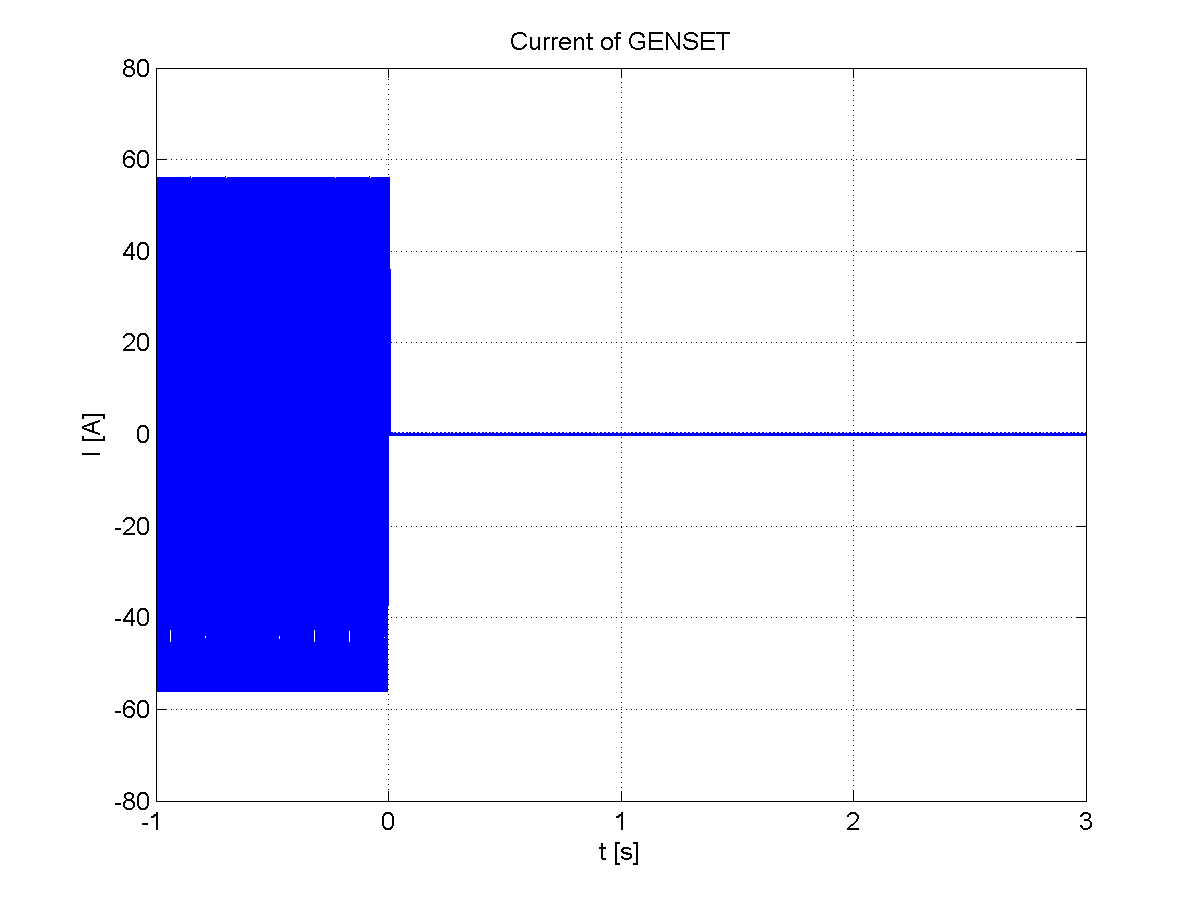
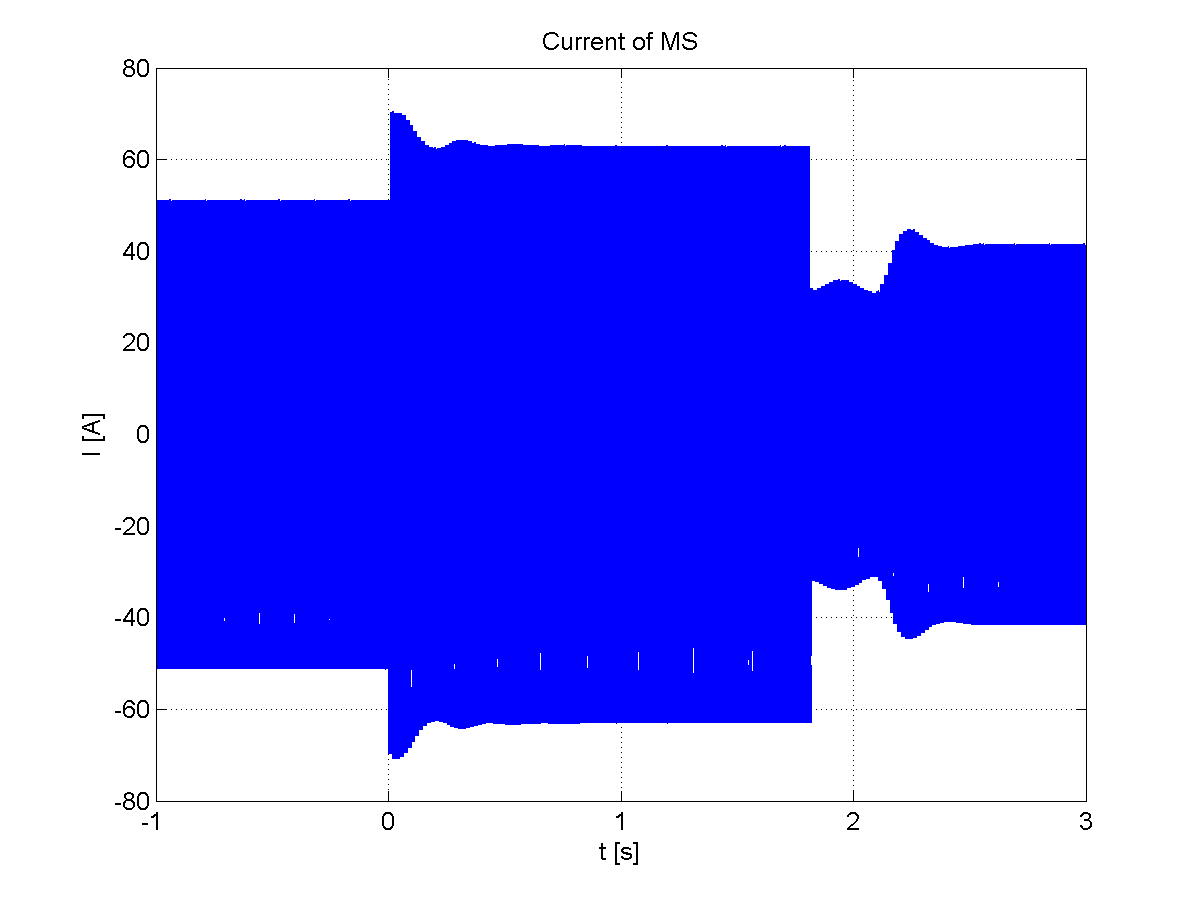
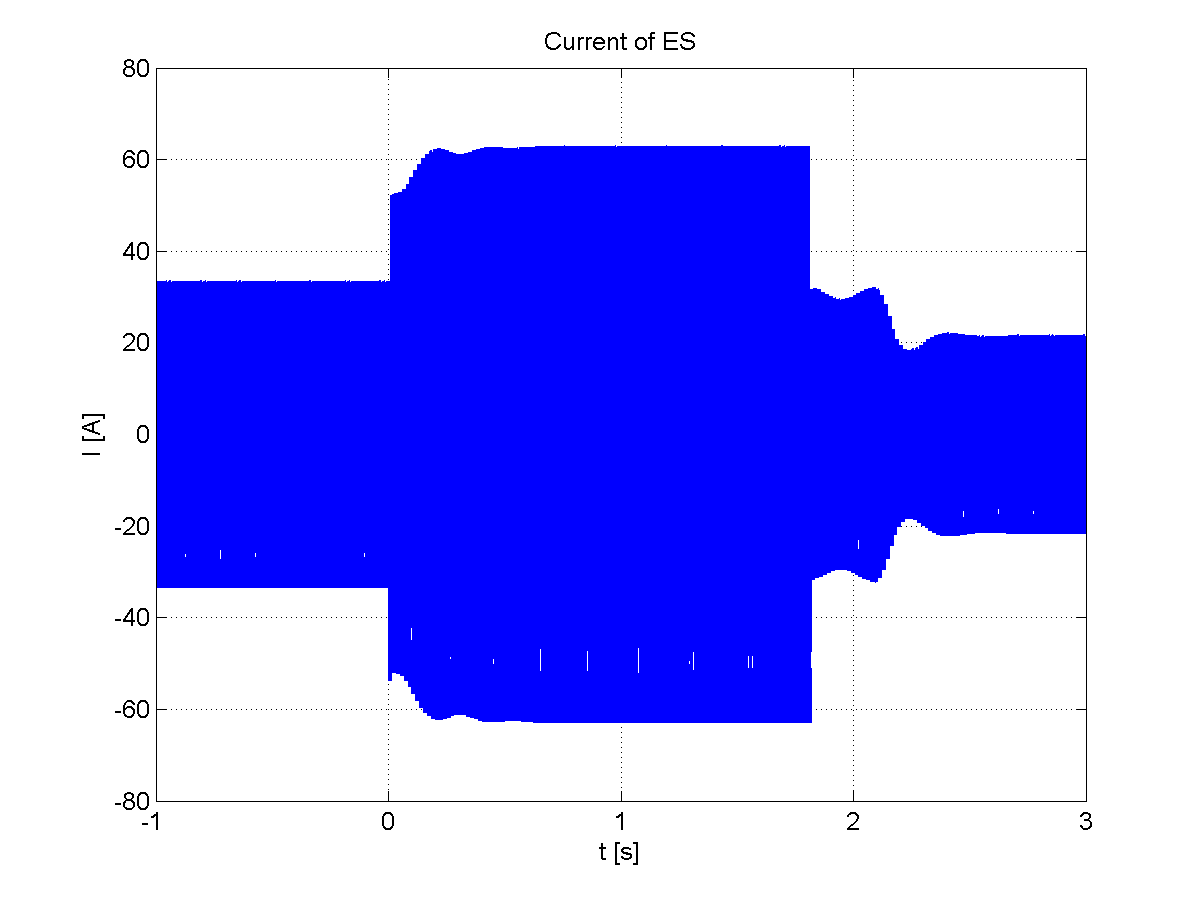
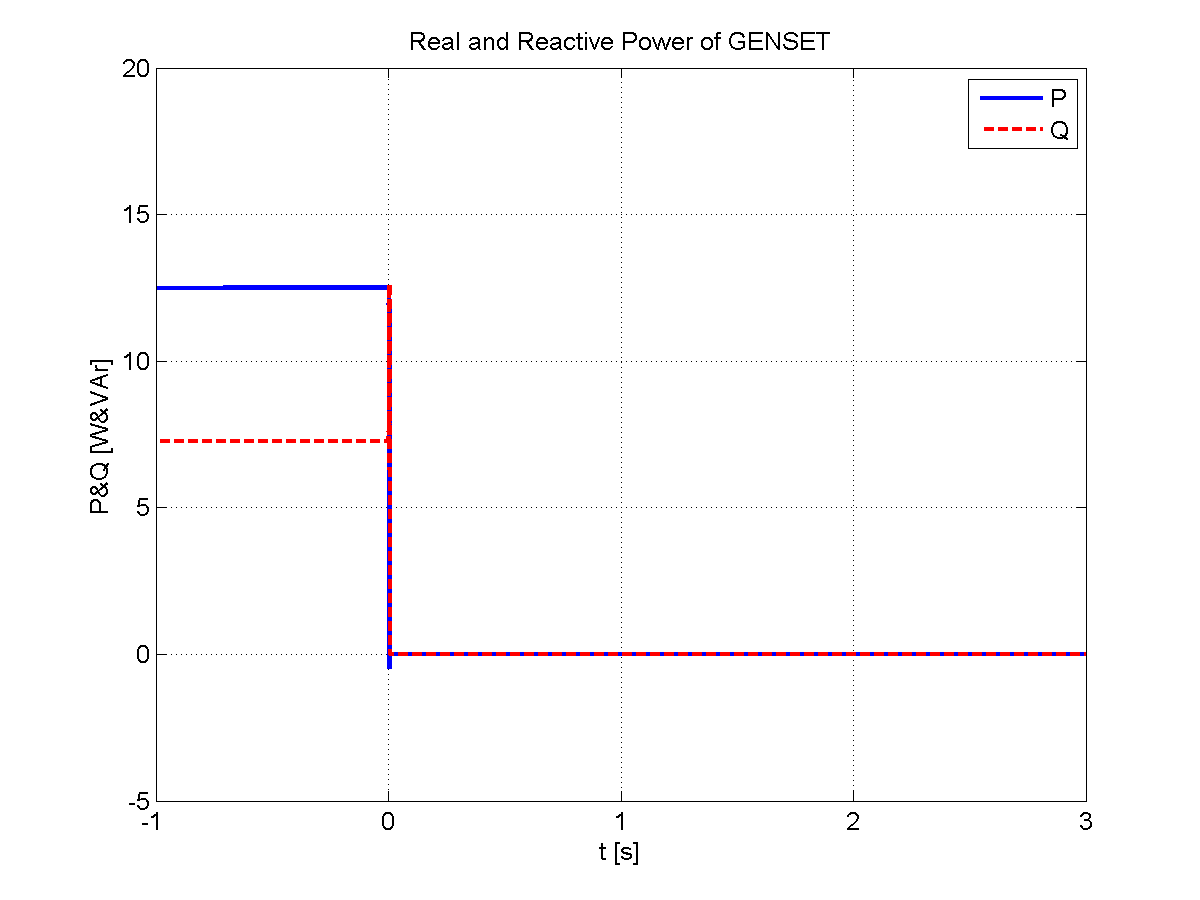
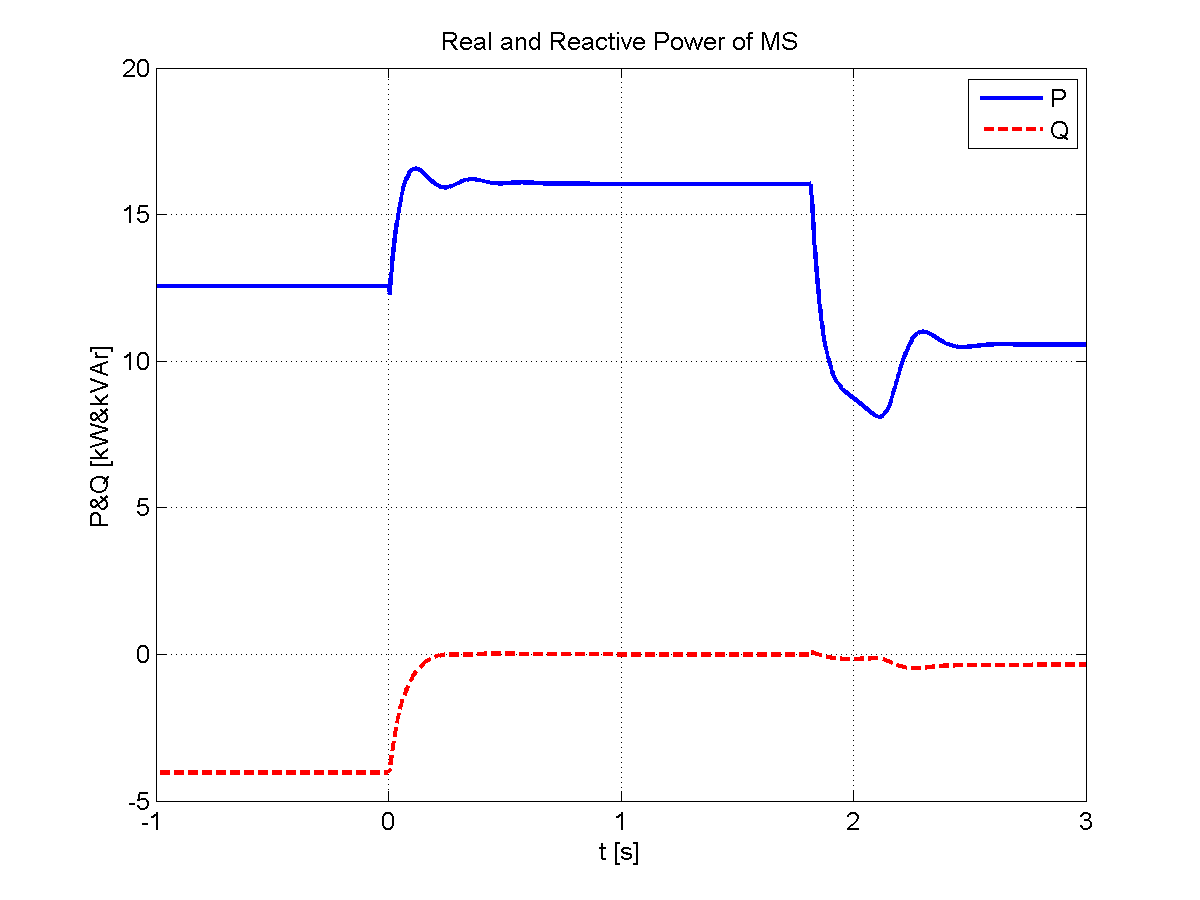
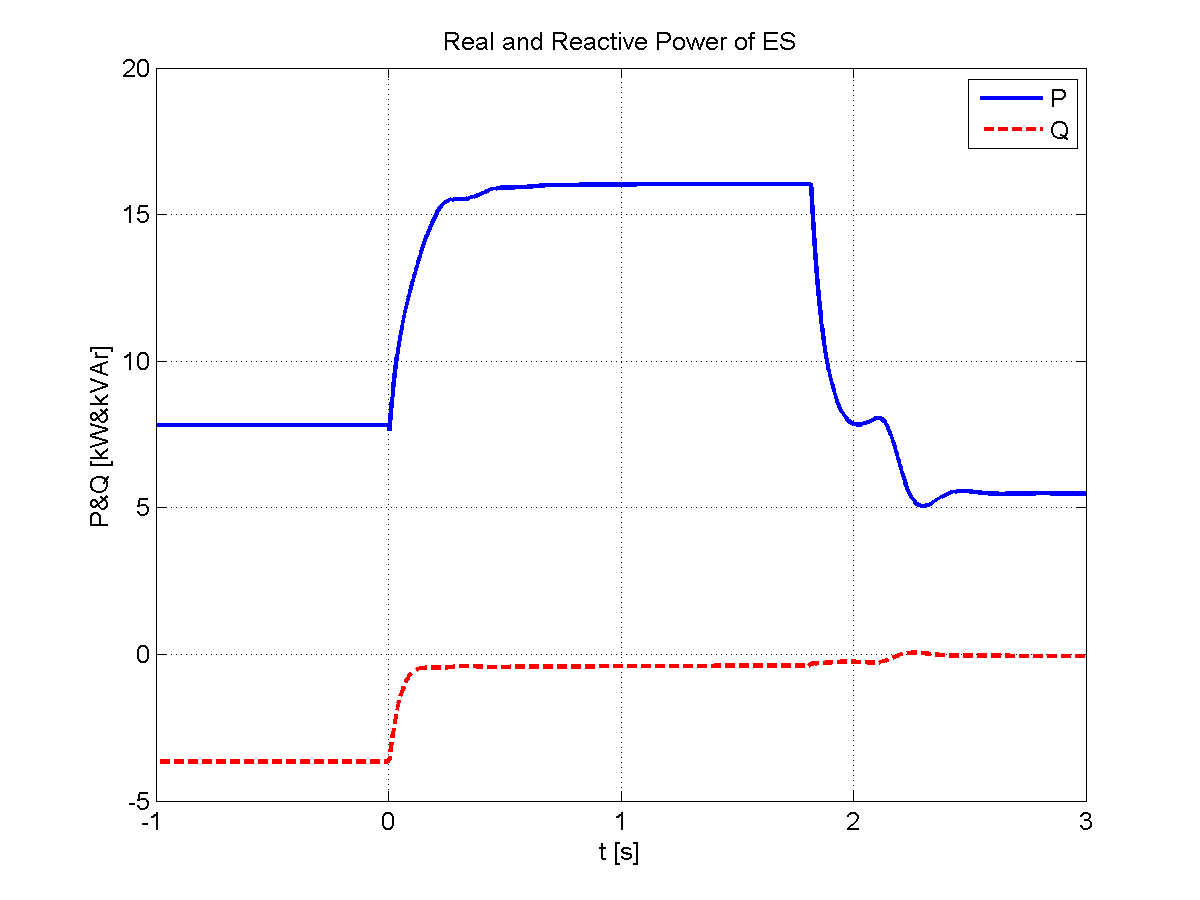
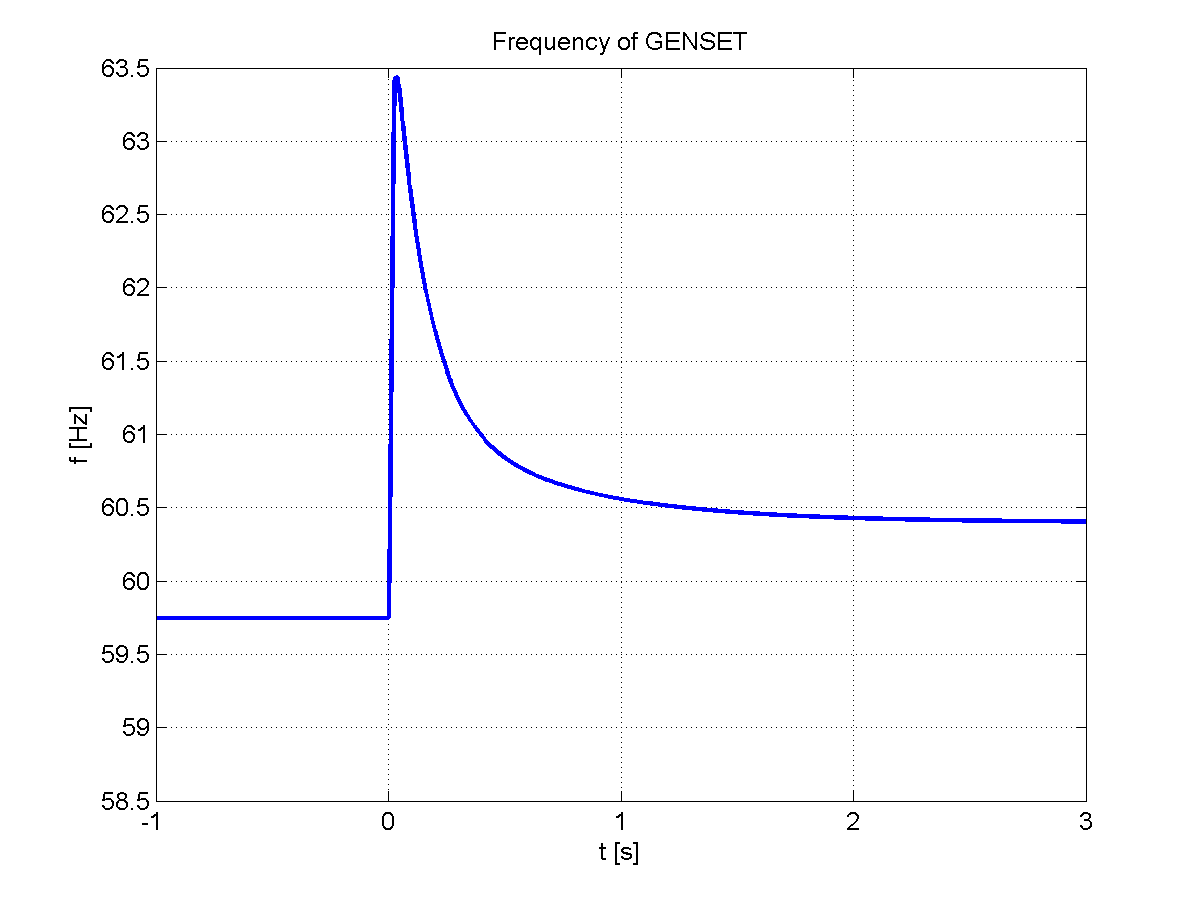
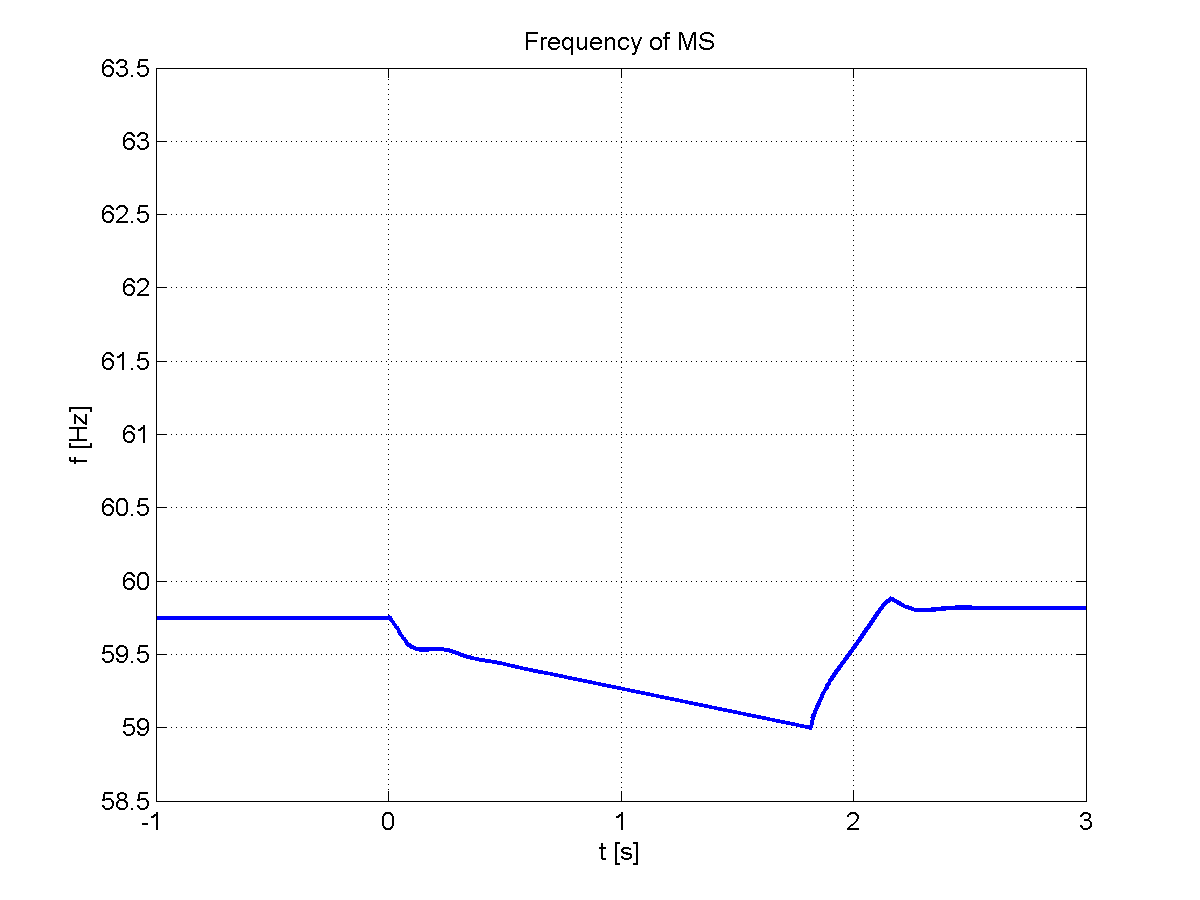
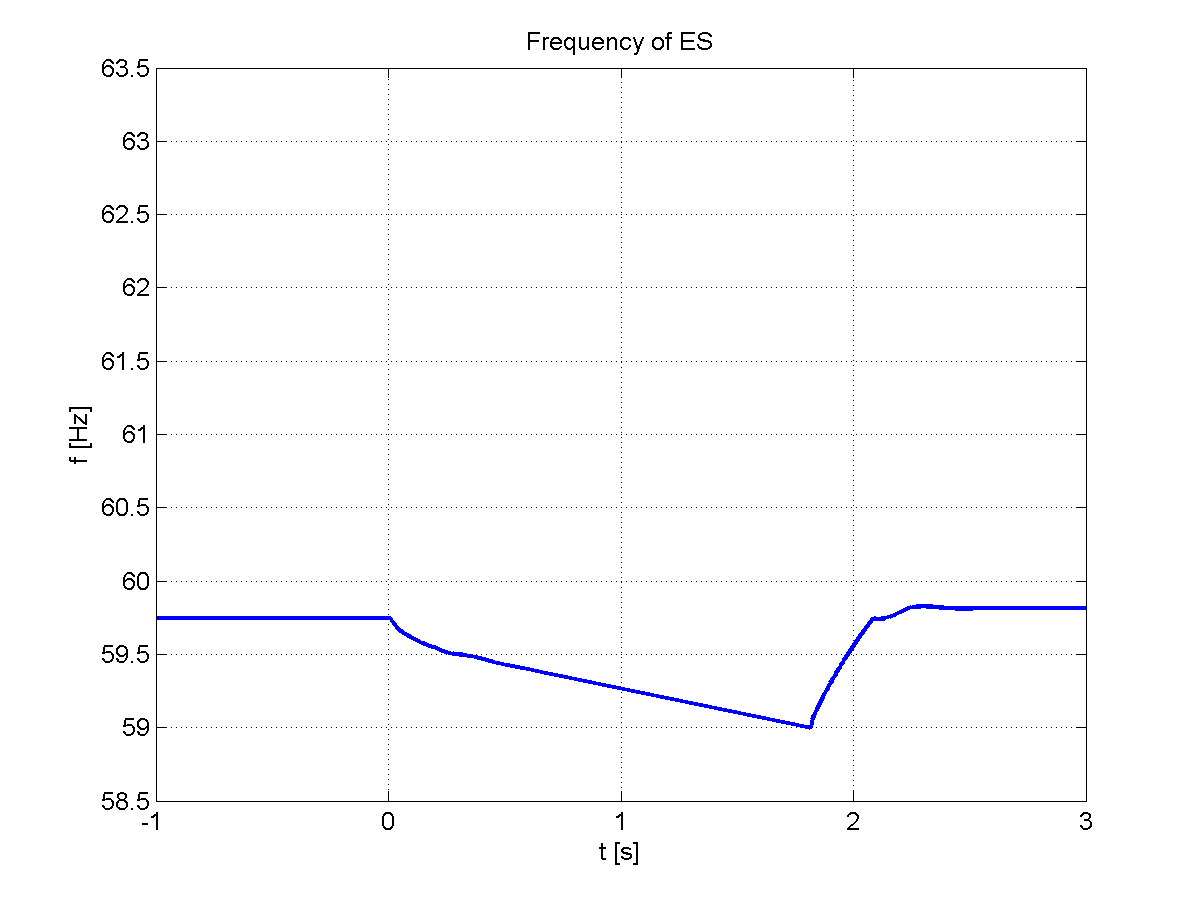


Figure 19: original results from UWM simulator, case 3





These simulated results for the ND simulation closely follow results obtained by UWM.

Differences between the simulations may be seen in the startup transients and the genset transients. These differences were seen as being minor and were attributed to differences in circuit models for cables and some of the genset saturation levels.

# Chapter 5: UND Simulator for Odyssian Testbed

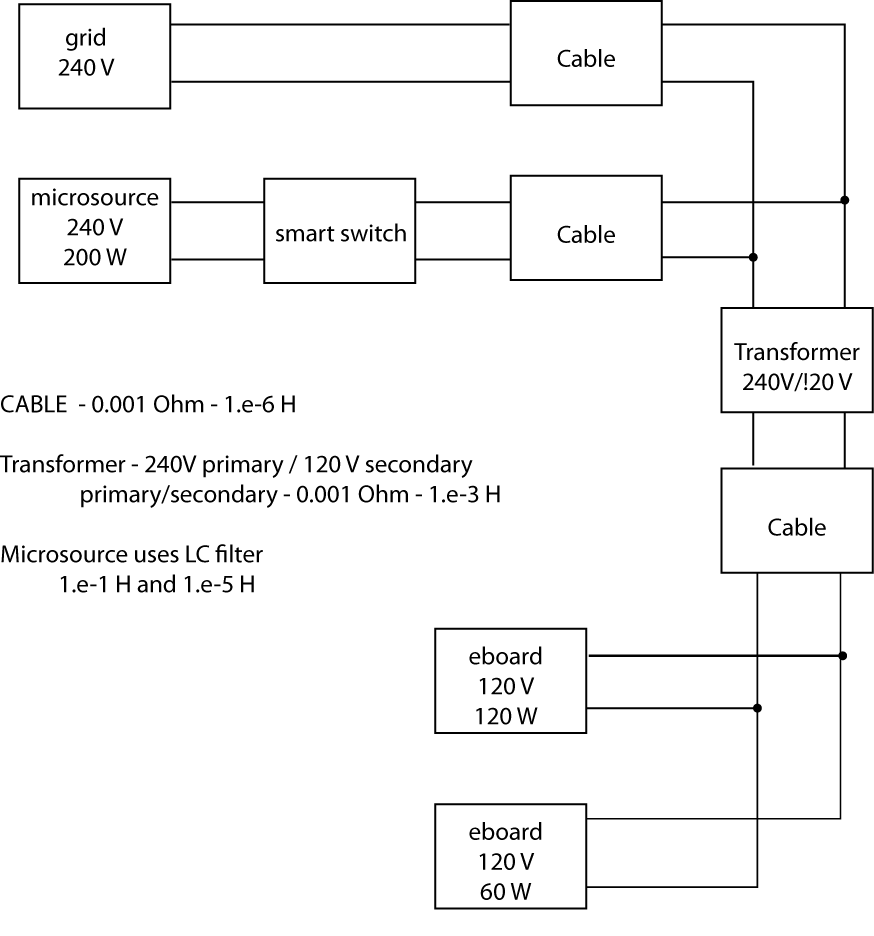


Figure 20: Initial Odyssian benchscale system

Odyssian’s bench scale microgrid is a single-phase system consisting of several 200W/240V microinverters that are connected through a single transformer to a collection of e-board load shedding devices. A simPower model for a simplified version of this testbed was built. The simplifications were needed to ensure that the simulation run time was not overly long. The modified testbed is shown in the block diagram (figure 20). This system consists of the main grid, a single inverter connected through a “smart” switch, and step-down transformer (240V to 120V) and two eboard modules. One e-board module has a critical load of 120 W and the other has a non-critical load of 60 W.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| The single-phase inverter models the system as a voltage source with an LC filter. To preserve system stability the inductor in this filter had to be chosen rather large (1.e-1). Subsequent discussions with UIUC indicated that they were connecting their inverters as current sources, rather than voltage sources, to avoid such large components. A block diagram for the simPower voltage-source inverter is shown below.  Untitled-1  Figure 21: Single phase inverter (simPower) model  The preceding system was simulated with the following events.   |  |  | | --- | --- | | Time  seconds | Event | | 0.0 | Simulation starts with main grid connected, e-boards disconnected, inverter disconnected (Preq= 0.8 pu) | | 0.1 | e-board loads (180 W) connected. | | 2.0 | Inverter (Preq=0.8 pu) connects. | | 3.0 | Inverter setpoint reduced to Preq = 0.2 pu. | | 4.0 | Microgrid islands. | | 4.3 | e-board sheds non-critical (60W) load | | 6.0 | Inverter setpoint increased to Preq = 0.8 pu. | | 7.1 | e-board reconnects non-critical (60W) load. | |

The following plots shows the inverter’s commanded frequency and the e-board’s estimated frequency. At the beginning the commanded frequency and estimated frequency don’t agree because the inverter is not connected to the microgrid. Once the inverter connects (2 seconds), the estimated and commanded frequency agree with each other. At 3 seconds into the simulation, the inverter’s Preq is reduced to 0.2 pu. This results in a transient in the commanded frequency. Because the main grid is still connected to the grid, there is no drop in the estimated frequency. At 4 seconds into the simulation the main grid disconnects. We see a drop in the frequency that causes the e-board to shed 60W of non-critical load at 4.3 seconds. Upon shedding the load, the frequency stabilizes at about 59.8 Hz. When the inverter Preq is increased to 0.8 pu at 6 seconds, the frequency rises to 60.1 Hz and the e-board reconnects the non-critical load.

This system works as expected.

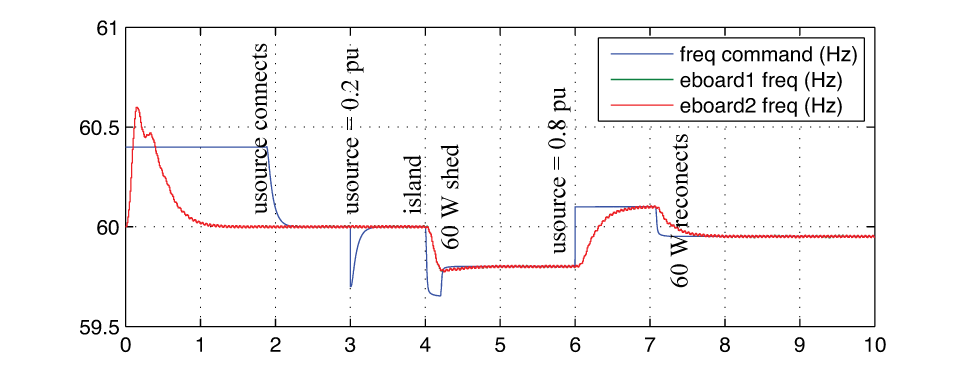


Figure 22: Response of initial Odyssian bench scale simulation

The simulation of Odyssian’s single-phase testbed was extended to include 3 microsources (200 W) with 4 e-boards (120 W). The simulation models were developed from last month’s microsource models, which modeled the inverter as a controlled voltage source. A block diagram for the simulated single-phase testbed is shown in figure 23. The total amount of microgrid generation is 600 W with a total load of 480 W. This dispatch logic is implemented using “computational agents” attached to the point of common coupling (PCC) and each microsources. This network of agents is used to adjust the microsource requires real power (Preq).

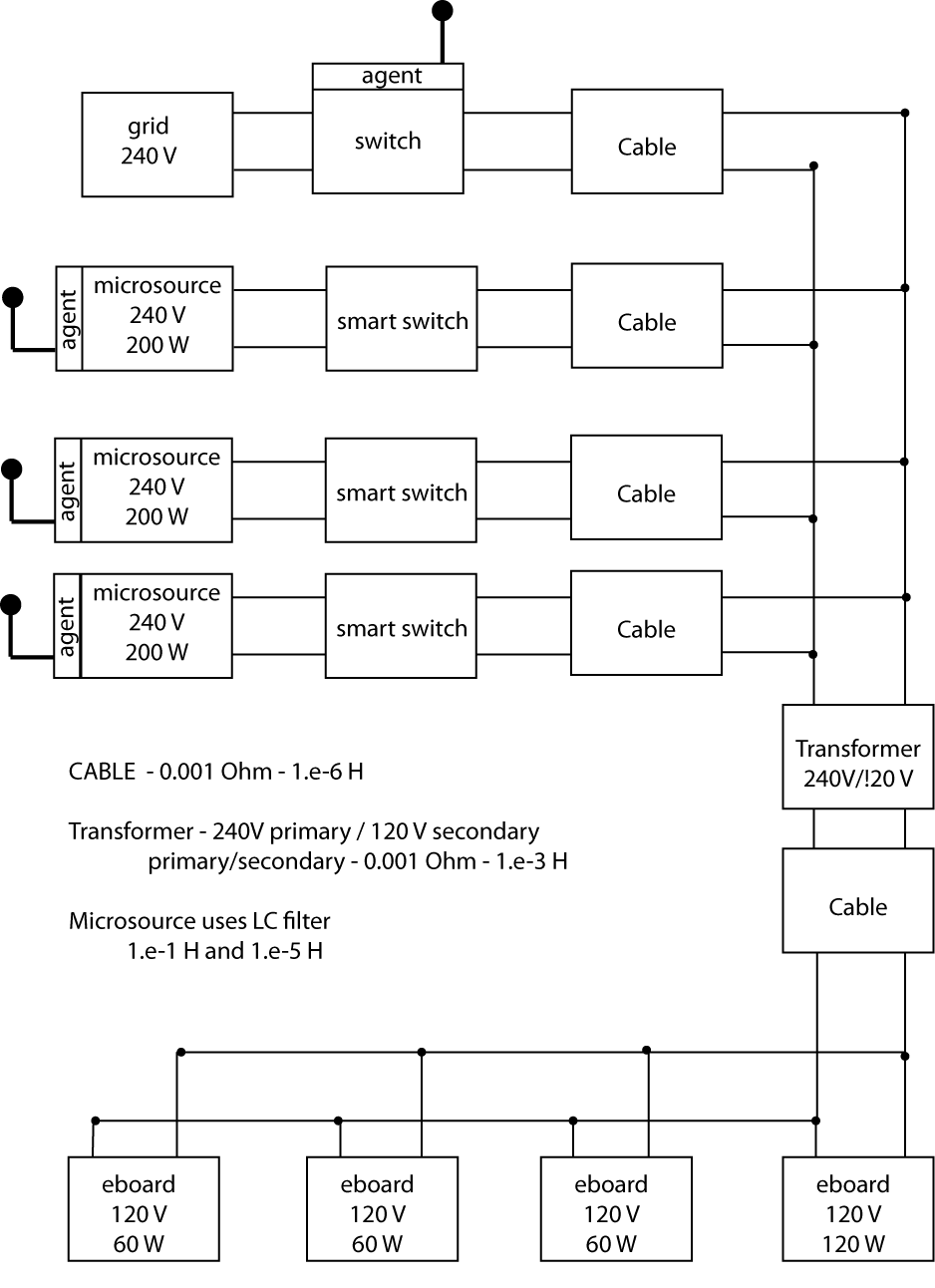
The dispatch algorithms for the Odyssian testbed are simpler than the earlier dispatch agents for mesh microgrids. In the first place, we don’t use event-triggered signaling in these dispatch agents. This will simplify the demonstration’s implementation. Secondly, there is no need for a highly distributed algorithm since all sources and loads in this testbed are directly interconnected. The dispatch logic, therefore, is much simpler. A preliminary form of the logic was developed this month and implemented on the simulation.

Figure 23: Odyssian system simulation with three sources.

The agent at the point of common coupling (PCC) monitors the real power flowing through the switch. This agent determines whether or not the microgrid has islanded from the main grid. That information is then transmitted to each of the microsource computational agents.

The agents attached to the microsources adjust the requested power to the UWM controller in a manner that reduces the cost of power generation subject to meeting the power balance relation. We use the commanded frequency from the UWM controller as a measure of the power balance. This leads to the following update algorithm for the requested power,

preq-update

where  is a constant (step size) and *k* is a gain. The line frequency, , and the requested power are in pu. The update consists of two terms. The first term is a function of frequency and increases or decreases Preq to keep the frequency at 1 pu (60 Hz). The second term is a slack term that decreases the requested power if it exceeds 1 pu. The above adjustment is only performed by a microsource if there is no power flowing through the PCC (i.e. the microgrid is islanded). If there is power flowing through the PCC, then all dispatch agents simply set Preq to 1 pu.

The following simulation illustrates the operation of the dispatch logic. The simulation scenario has the following time line

|  |  |
| --- | --- |
| Time  seconds | Event |
| 0.0 | Simulation starts with main grid connected, e-boards disconnected, and  Inverters disconnected |
| 0.1 | All e-board loads (480 W) connect. |
| 1.5 | All inverters (600W) connects and exports power to the main grid |
| 2.0 | Islanding event with all inverters (600W) serving the e-boards (480W) |
| 3.0 | Inverter number 3 disconnects (total available power drops to 400W) |
| 3.7 | Non-critical e-board sheds 120W load |
| 4.5 | Inverter number 3 reconnects to the grid |
| 5.1 | e-board reconnects non-critical load to the microgrid |

Figure 24 shows the time histories for the main grid (at PCC), inverter 3 (at its smart switch), and the terminal leading to all loads.

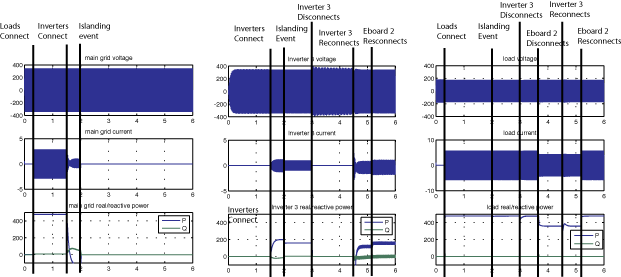


Figure 24: Simulation results for Odyssian bench scale system with 3 sources

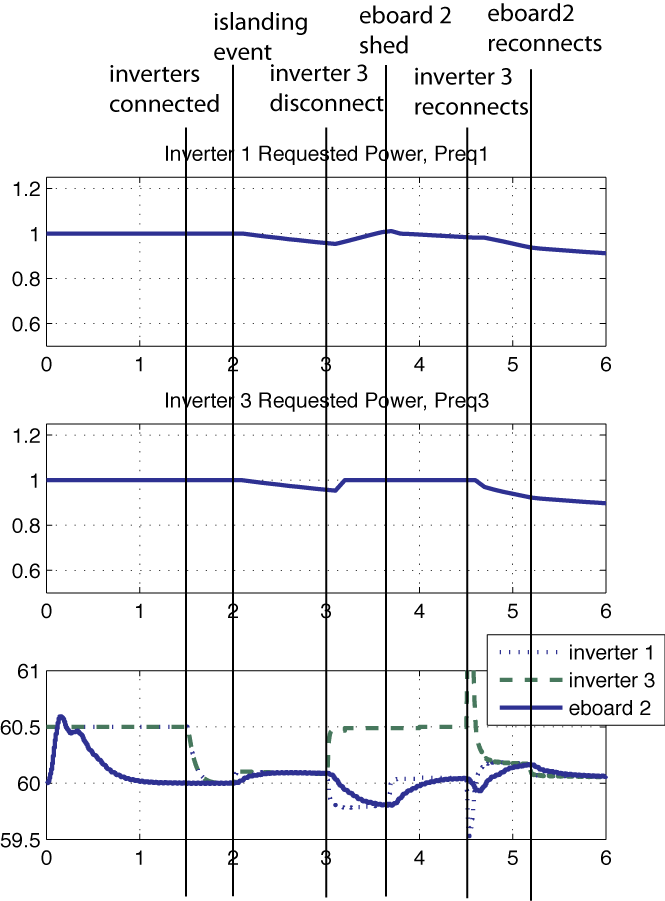
From this figure one can see that the system is able to maintain the voltage levels to the loads during islanding events, disconnection/reconnection of loads, and disconnection/reconnection of inverter.

Figure 25 frequency response and requested power generated by dispatch agents.

The frequency response and the requested powers generated by the dispatch agent are shown in the figure 25. Prior to the islanding event, the requested power is set to 1 pu. Upon islanding, there is excess capacity (since frequency is greater than 1 pu) and the requested power setpoints begin to reduce. When inverter 3 drops off, we see an increase in inverter 3’s frequency (since it has no load) and the requested power is set to 1 pu (in anticipation of reconnecting later). In the meanwhile, the dispatch logic begins increasing the requested power level in order to raise the line frequency. When inverter 3 reconnects, there is a short transient in the frequency estimate as it converges to the commanded line frequency and the requested power level begins to decrease since there is now excess capacity.

# Chapter 6: Load Shedding Algorithms

This chapter documents UND’s algorithm and simulation efforts regarding frequency-based load shedding in the UWM microgrid. An important feature of the UWM controller is that the line frequency droops when there is power drawn by system loads exceeds the requested power set point. This frequency droop can be seen as a symptom of system stress and one way of quickly addressing this issue is to drop resistive loads. Odyssian’s e-board load units carried out the load shedding function. This section describes the algorithms recommended for Odyssian’s e-board units.

**6.1 Line Frequency Estimator Design and Evaluation**

Load-shedding algorithm drop load when the frequency droops below a specified threshold. The load shedding algorithms used by Odyssian’s e-board, would therefore need to estimate the voltage line’s frequency with sufficient accuracy to ensure fast disconnection when the frequency droops. The original frequency estimator proposed by Odyssian was based on counting zero-crossings over a specified time interval. This approach generated estimates that had a precision of 1.0 Hz every 0.3 seconds.

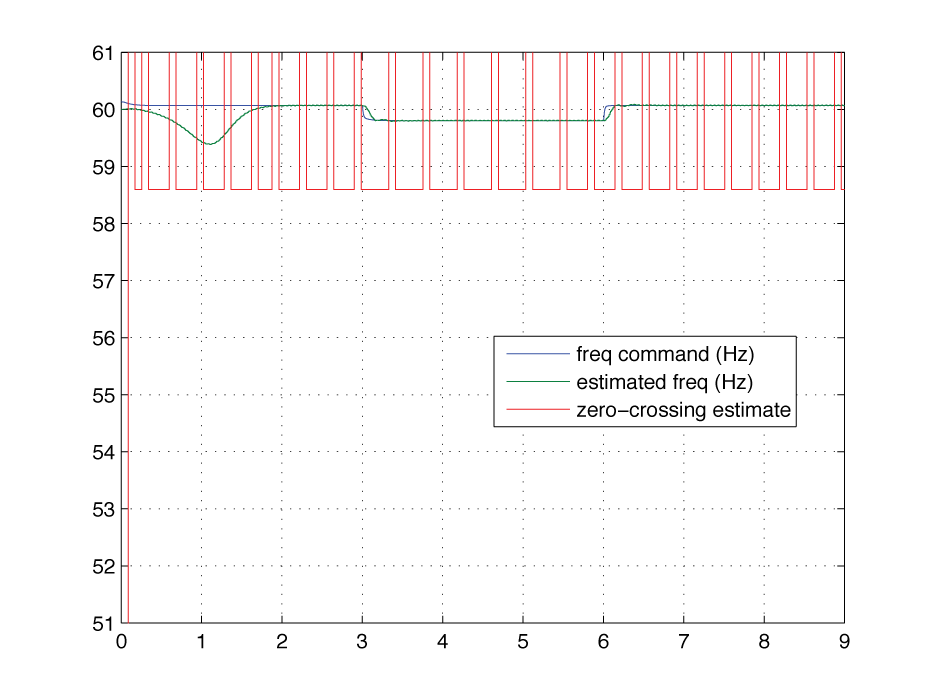
An example of the estimated frequency is shown in figure 26. This simulation was generated for the microgrid simulation model used in testing the CERTS controller on a microsource. To generate frequency estimates with greater precision would require a greater interval between estimates. This level of estimation performance is inadequate for load shedding.

Figure 26: Estimation performance of Odyssian’s original zero-crossing frequency estimator

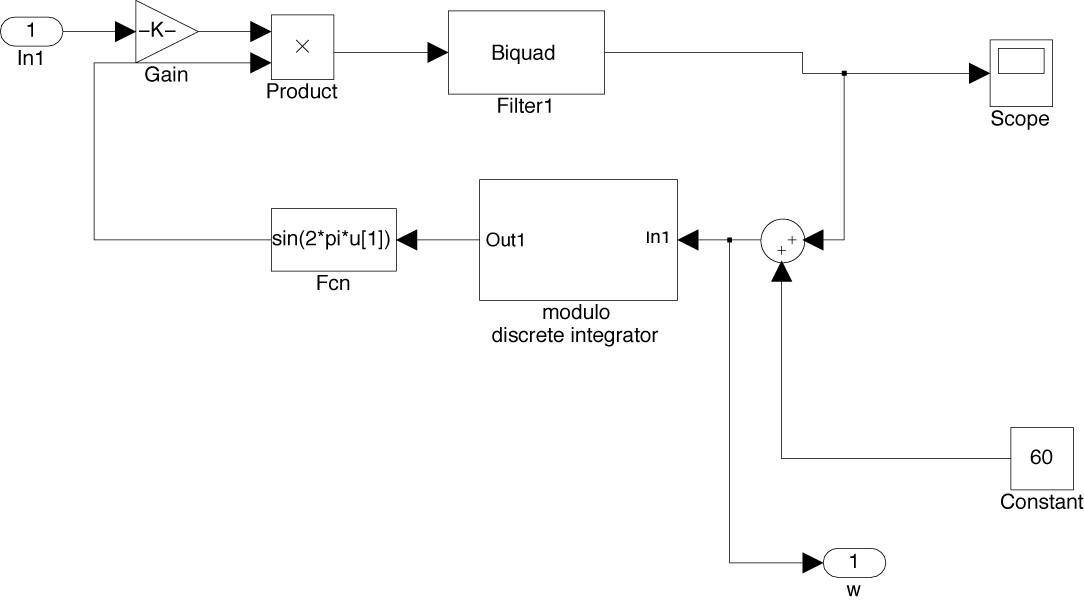


Figure 27: Simulink model of phase-locked loop estimator

An alternative frequency estimator was designed using a phase-locked loop (PLL). In this estimator, the input waveform is mixed with a 60 Hz sinusoid and then run through a low pass filter. The output of this filter is used to change the frequency of a voltage-controlled oscillator (VCO) generating the mixing sinusoid. The output of the filter represents the change in frequency from the 60 Hz reference sinusoid. This method provides an extremely fast and accurate estimate of the input signal’s frequency. A simulink block diagram for the PLL frequency estimator is shown in figure 27.

The simulink model was constructed as a digital model assuming a sampling time of 10 msec. The key components in the filter are the biquad filter and the modulo digital integrator. The “modulo” nature of the integrator prevents integrator overflows that can reduce accuracy. The Biquad filter is an elliptical low pass digital filter with a 100 Hz sampling frequency, a 5 Hz cutoff frequency, and a 20 Hz stop band frequency with 40 dB attenuation. The filter was implemented as a two-stage biquad structure in direct form II. A block diagram for this filter is shown below in figure 28.

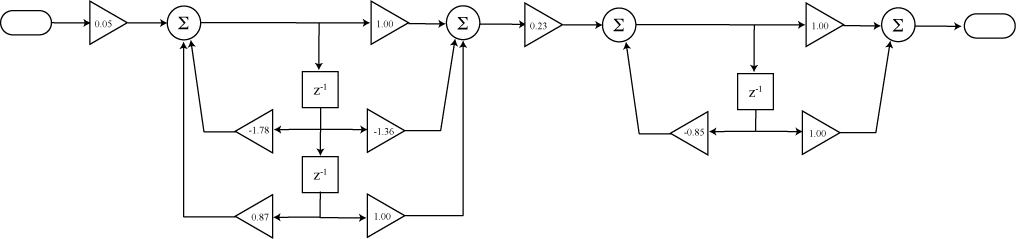


Figure 28: Signal Flow Graph of Direct Form II Structure of Low Pass Filter

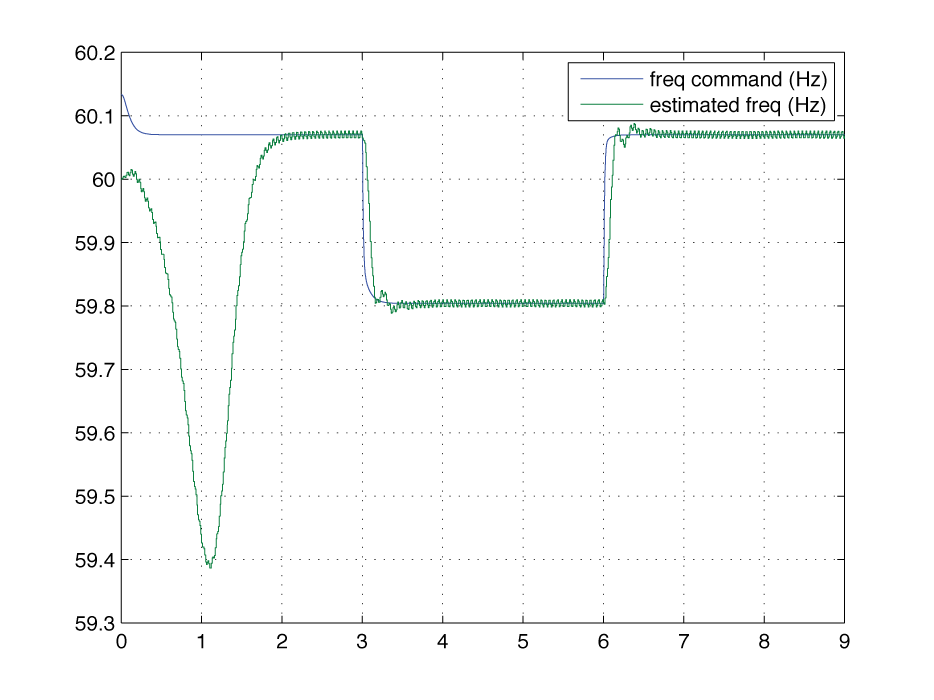
 The performance of this estimator is shown in figure 29. There is an initial transient as the estimated frequency converges to the true value. At 3 seconds into the simulation an additional load is reconnected to the grid thereby causing a frequency droop. This droop is seen in the commanded frequency generated by the generator’s CERTS microsource controller. As can be seen the frequency estimator tracks this commanded frequency with a higher degree of precision and more quickly than the zero-counting frequency estimator.

Figure 29: Commanded and frequency estimate obtained from PLL estimator on UWM Simulation

**6.2 Simplified Frequency-based Load-shedding**

The frequency estimator of section 6.1 was integrated into a prototype load-shedding module that was tested on the UWM simulator. The load shedding component was designed to disconnect a load if the line frequency drops below 59.7 Hz or the RMS line current exceeds 100 A. Once the load has been disconnected, the load remains disconnected until a reset pulse is sent to the load-shedding component.

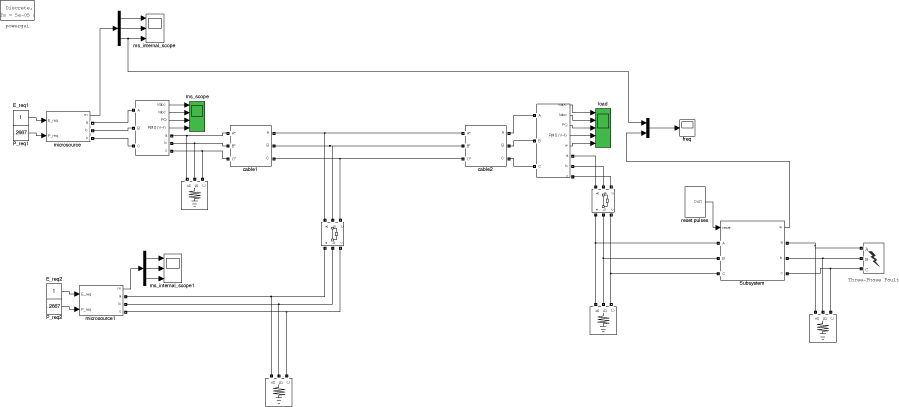


Figure 30: Simulink model used in testing simplified (non-adaptive) load shedding component

Figure 30 shows a simulink model for the microgrid used to test the load-shedding component. This grid has two CERTS controlled 15 kW microsource generators connected to a total of 20 kW loads. 8 kW’s of these loads can be shed on the basis of either a current or frequency limit. At 3 second a ground fault occurs, which clears it self in 4/60 seconds. At 5 seconds the second microsource generator is disconnected. Reset pulses are sent to the load shedding module 1 and 4 seconds. These reset pulses clear the disconnection due to the fault, thereby allowing the system to shed to extra load when the second microsource is lost at 5 seconds.

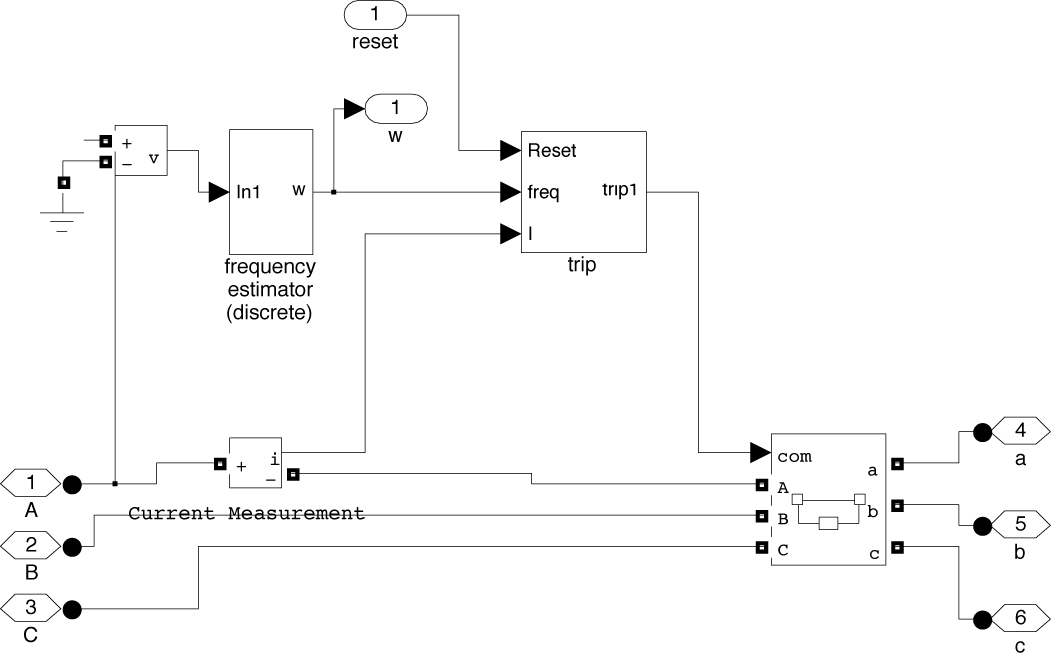


Figure 31: Load shedding module

The simulink block in figure 31 shows the load-shedding module. The inputs to this module are the 3 phase lines. The voltage on phase A is fed into the frequency estimator (described above), whose output is fed into the trigger component. The current on phase A is also measured and fed into the triggering component. The other input to the load-shedding module is the reset pulse. The triggering component uses the reset signal, the line frequency, and the RMS line current to open or close the breaker.

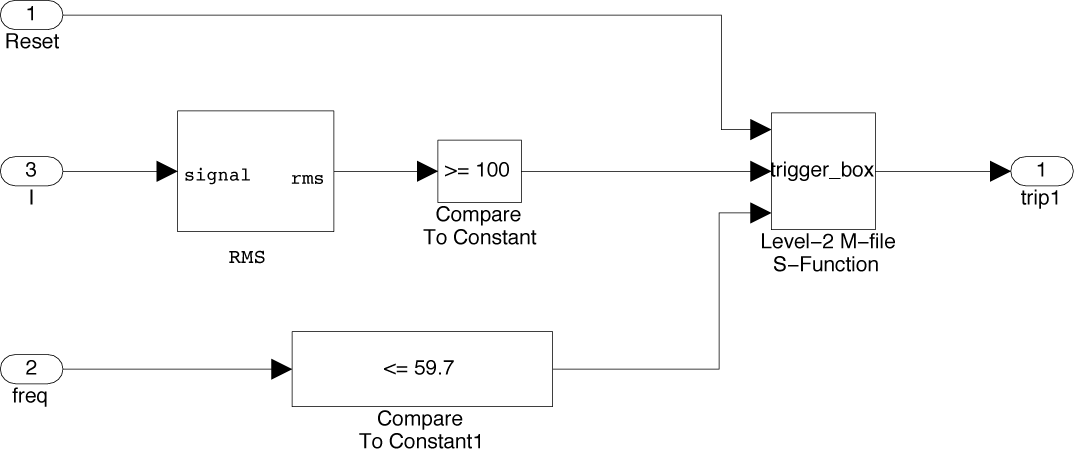
The triggering component’s simulink model is shown in figure 32. This component generates a frequency event, if the frequency falls below 59.7 Hz and a current event if the RMS current exceeds 100 A. The triggering logic is coded into a level 2 M-file S-function (trigger\_box). The logic opens the breaker if the frequency or current events are triggered. Once opened, the breaker stays open until a reset pulse is received. Receipt of the reset pulse closes the breaker and resets the event logic, so the breaker can be opened again by either a frequency or current event.

Figure 32: Triggering component of load shedding module

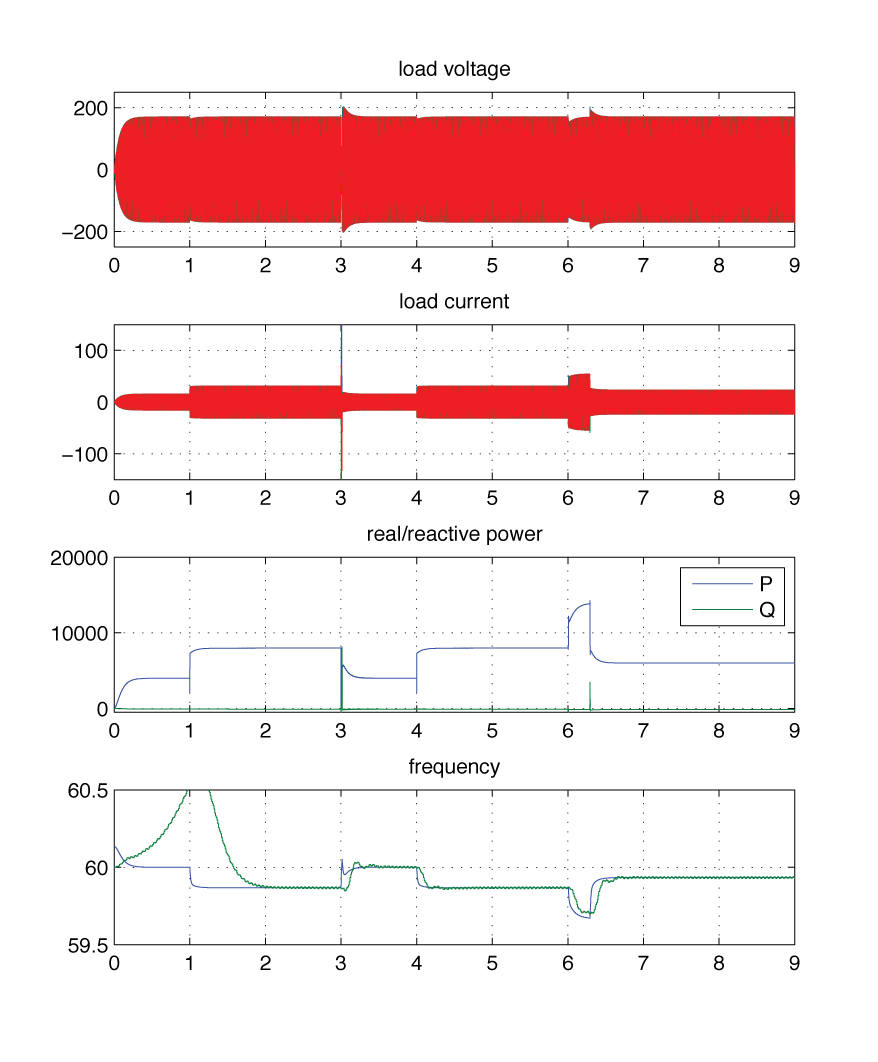
In the simulation used to test this component, we have 3 events. The controlled breaker is initially open. There is an event at 1 second when a reset signal is sent to the breaker, causing it to close. At 3 seconds, a fault occurs on the load connected to this breaker. This results in a current event that causes the breaker to open and thereby drop 8kW of load. At 4 seconds another reset pulse is sent to the load-shedding module, thereby reconnecting the load (since the fault has cleared). At 6 seconds, the second microsource is disconnected, causing the line frequency to drop below 59.7 Hz. The load shedding module successfully detects the frequency event and drops the 8 kW load again at 6.4 seconds into the simulation. The simulation plots are shown in figure 33. These plots clearly show the occurrence of the current and frequency events. The step changes in current and power clearly show that the extra load is dropped as expected.

Figure 33: Simulation Results for simplified load shedding algorithm

**6.3 Automatic Load-shedding with adaptive reconnection**

This section discussed an extension of the load shedding logic presented in section 6.2 This new load shedding algorithm uses an adaptive method for recomputing the frequency at which a shed load would be reconnected. This method involves having the load broadcast its status to everyone else when it disconnects from the microgrid. All loads then keep track of how much power has been shed by these loads and uses that to compute a reconnection frequency through the formula

****

where *load\_shed* is the total amount of load that has been shed, *fin*, is the line frequency (Hz) about 0.25 seconds after the load was shed and *fconnect* is the frequency at which the shed load will reconnect to the microgrid.

This logic was implemented in a simPower component modeling Odyssian’s e-board. The following pseudo-code represents the update logic used by this component.

Function Update(block)

%block.Dwork is internal state of program (stored variables)

%block.InputPort is the input information to the program

%block.OutputPort is the output information generated by the program.

%fetch stored program states

% output\_state: what was last output by program

% connected (1) or unconnected (0)

% latch\_state: a shed load remains unconnected

% (i.e., latched) until it is unlateched.

% once unlatched the load is free to reconnect

% latched (1) / unlatched (0)

% timer\_state: value of one-shot timer

% positive integer values

% freq\_connect: frequency at which reconnection occurs

% this is a positive real-valued number

output\_state = block.Dwork(1).Data(1);

latch\_state = block.Dwork(1).Data(2);

timer\_state = block.Dwork(1).Data(3);

freq\_connect = block.Dwork(1).Data(4);

%fetch inputs to program

% reset\_input: command signal that reinitializes component to its home state

% freq\_input: line frequency at e-board (Hz – postive real)

% current\_input: RMS current flowing into e-board (A)

% voltage\_input: RMS voltage across e-board terminals (V)

% load\_priority: characterizes priority level of eboard

% (1) = critical , (2) = non-critical

% shed\_load: the total amount of load (pu) that has been shed so far

% this is computed using the output\_states of all boards

reset\_input = block.InputPort(1).Data;

freq\_input = block.InputPort(2).Data;

current\_input = block.InputPort(3).Data;

voltage\_input = block.InputPort(4).Data;

load\_priority = block.InputPort(5).Data;

shed\_load = block.InputPort(6).Data;

%load shedding update logic

% if the timer\_state is positive, then the one-shot timer is running.

% So decrement the timer (if timer\_state > 0)

% If timer\_state == 0, then timer has expired (or is not running)

% in which case we set the unlatch the board (latch\_state=0)

% and compute a new connection frequency (freq\_connect)

if (timer\_state > 0)

timer\_state = timer\_state-1;

if (timer\_state == 0)

latch\_state = 0;

freq\_connect = min(freq\_input + shed\_load/2 , 60.5);

end;

end;

%If a reset pulse is received, then restore the home state of the board

if (reset\_input==1);

timer\_state=0;

latch\_state=0;

output\_state=1;

freq\_connect = 60.5;

end;

%start the timer if the output is latched

%and frequency is greater than 60.5

%reconnect if the output is not latched and

%the frequencyis greater than the connect frequency.

% if the board is not latched and the line frequency is greater than

% the previously computed connection frequency, reconnect the board

if (latch\_state == 0)&&(freq\_input >= freq\_connect)

output\_state = 1;

end;

%this is the load shedding logic

% switch on the type of load

% if load\_priority = 1 (critical), then shed at 58.5 Hz

% if load\_priority = 2 (non-critical), then shed at 59.5 Hz

%

% when the load is shed, start the timer and set reconnect frequency

% to 60.5 Hz. This connection frequency gets recomputed after 0.25 seconds

% (timer expiration) using the total amount of shed load and the line

% frequency when the timer expires. Upon shedding the load, the disconnected

% status of the board is “latched” (i.e., latch\_state=1) and the output state

% is set to 0 (unconnected).

Switch load\_priority

case {1}

if(latch\_state==0)&&(freq\_input<=58.5)&&(output\_state==1)

latch\_state=1;

output\_state=0;

timer\_state = 25;

freq\_connect = 60.5;

end;

case {2}

if(latch\_state==0)&&(freq\_input<=59.5)&&(output\_state==1)

latch\_state=1;

output\_state=0;

timer\_state = 25;

freq\_connect = 60.5;

end;

end;

%save the past states for the next iteration

block.Dwork(1).Data(1)=output\_state;

block.Dwork(1).Data(2)=latch\_state;

block.Dwork(1).Data(3)=timer\_state;

block.Dwork(1).Data(4)=freq\_connect;

%endfunction

function Output(block)

output\_state = block.Dwork(1).Data(1);

block.OutputPort(1).Data = output\_state;

%endfunction

This algorithm was tested on the simPower simulation for the UWM testbed. The results of this simulation also included results for the dispatcher. These simulation results are documented in chapter 8. The next chapter discusses the implementation of the dispatch algorithms that were used in the chapter 8 simulation results.

# Chapter 7: Distributed Dispatch Algorithm

Chapter 2 presented the basic principles behind distributed power dispatch algorithms. Chapter 3 presented an event-triggered version of these algorithms that was tested at the very beginning of this project. The dispatch algorithms discussed in this chapter represent simplifications of the original algorithms in chapters 2 and 3. The algorithms in this chapter were simplified due to the limited capabilities of the actual UWM testbed and Odyssian testbeds they were intended to support. The UWM testbed that Odyssian finally ended up testing their algorithms on consisted of only two microsources. Odyssian’s bench scale testbed was even simpler as it had a single bus to which all loads and generators were connected. In addition to this, neither testbed was able to measure load power or line power in real-time. The algorithms from chapter 2/3 assumed a full mesh topology with load/line power measurements available. Since this was not the case for the testbeds that finally appeared at the end of the project, the original dispatch algorithms had to be simplified. This chapter discusses the resulting simplified dispatch algorithms and their simulation testing.

The remainder of this chapter is organized as follows. Section 7.1 presents a centralized version of the simplified dispatch algorithm that was designed for the UWM testbed. While this algorithm was centralized, it was coded in a way that was easily distributed between the generators. Section 7.2 describes the simPower model that was used to test this distributed implementation. Section 7.3 describes the dispatch agent’s interface specification that was delivered to UWM.

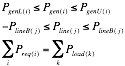
**7.1 Centralized Implementation of Dispatch Algorithm in UWM Testbed**

The dispatcher recomputes the desired *Preq* input to the UWM controllers, to minimize the cost of operating the microgrid subject to generator power limits and power line flow constraints. This update approach makes use of the augmented Lagrangian method that was tested earlier (chapter 3) for the event-triggered dispatcher. The algorithm is developed as follows.

Define a cost function, *C*(*Pgen*), which represents the aggregate cost of running all generators. For this work, we assume *C* is a quadratic function of the generator power *Pgen*.



We’d like to minimize this cost subject to the following constraints



where *Pgen(i)* is the power flowing from the generator *I*, *Pline(j)* is the power flowing through feeder line *j*, *Preq(i)* is the power level set point for generator *I,* and *Pload(k)* is the power flowing into load *k*. The first constraint requires the generator power to lie between a lower limit, *PgenL(i)*, and upper limit *PgenU(i)*. The second constraint requires the absolute value of the line power to be less than *PlineB(j)*. The last constraint is a power balance condition requiring the total generators’ power setpoint equal the total load power, *Pload*.

We solve this optimization problem using an augmented Lagrangian method. This involves minimizing an augmented cost function of the form,



The functions, *χ*( ), *ψ*( ), and *ξ*( ) in the above are terms penalizing the violation of the constraints given in the original optimization problem. The first two functions, in particular, take the form,



The last discounting function,  will be defined in a somewhat different manner. In general, it will not be feasible to assume that generators have easy access to measurements of load power. Since the generators all use the CERTS droop controllers, one way of directly measuring the mismatch between *Preq* and *Pload* is to use the frequency droop from 60 Hz. So, rather the actual ξ function to be used will take the following form,



The dispatch algorithm searches for a local minimum of the augmented Lagrangian, *L*, using a simple gradient descent procedure. The challenge is to restructure the computation of this gradient so the generators can do it in a distributed manner. In looking at the above expression for *L*, it should be apparent that the terms involving the cost function *C* and the constraint function, χ, are easily distributed between the generators. The power balance constraint in ξ is also distributed since we compute this as a function of the frequency, *fgen(i)*, computed by the generator. The only term in question is the line power constraint, *ψ,* since this is a function of the feeder lines rather than the generator.

To handle this problem, let us define a “transmission line” state,



and let’ define a generator state



We’ve represented these states as functions of time, since they are computed from either line or generator voltages that are measured in real-time.

Following the analysis in chapter 3, we can now compute an update for the *i*th generator’s *Preq(i)* that takes the form,

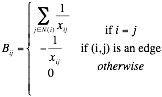


where



where *A* is the weighted incidence matrix of the microgrid’s graph defined as

*A=DI.* In this equation *I* is the incidence matrix of the graph (map from nodes to links) and *D* is a diagonal matrix whose diagonal components are the reactances of the transmission lines. The matrix *B* is a weighted Laplacian matrix for the graph whose *ij*th component is



The preceding discussion characterizes how the dispatcher logic updates a generator’s power set point, *Preq*. We now turn to discuss when the dispatcher should be adjusting this set point. If one attempts to adjust the set point immediately after an islanding event, load shedding event, or loss of generator event, then we’ll see significant interference between the dispatch logic and the UWM CERTS controllers. What this means is that operating the dispatch logic during such events can reduce power quality by increasing the amount of time it takes for the CERTS controller to stabilize.

One way around this problem is to simply turn off the dispatcher during such transients. This is a realistic thing to do, since the dispatcher’s role is only to optimize microgrid operation over the long-term. The dispatch logic, therefore, monitors the generator power and frequency command to detect if an event has occurred. In particular, if the line frequency drops below 59.5 Hz or if the generator power changes by more than 0.2 pu over a 0.01 second interval, the dispatcher assumes an event has occurred and it switches itself off for a specified interval of time of 0.5 seconds. This time interval gives the CERTS controller to stabilize in a way that subsequent operation of the dispatcher will not adversely impact the controller’s performance.

Figure 34 (below) shows a block diagram for the UWM simulator with this centralized dispatcher. The simulink blocks for the dispatcher are shown inside the box marked off in figure 34. The left hand side of the box has goto components that are connected to scopes measuring the various line powers needed by the algorithm. The right hand side of the box has goto components that connect the requested power computed by the algorithm to each of the generators. The large component in the middle is the centralized dispatcher.

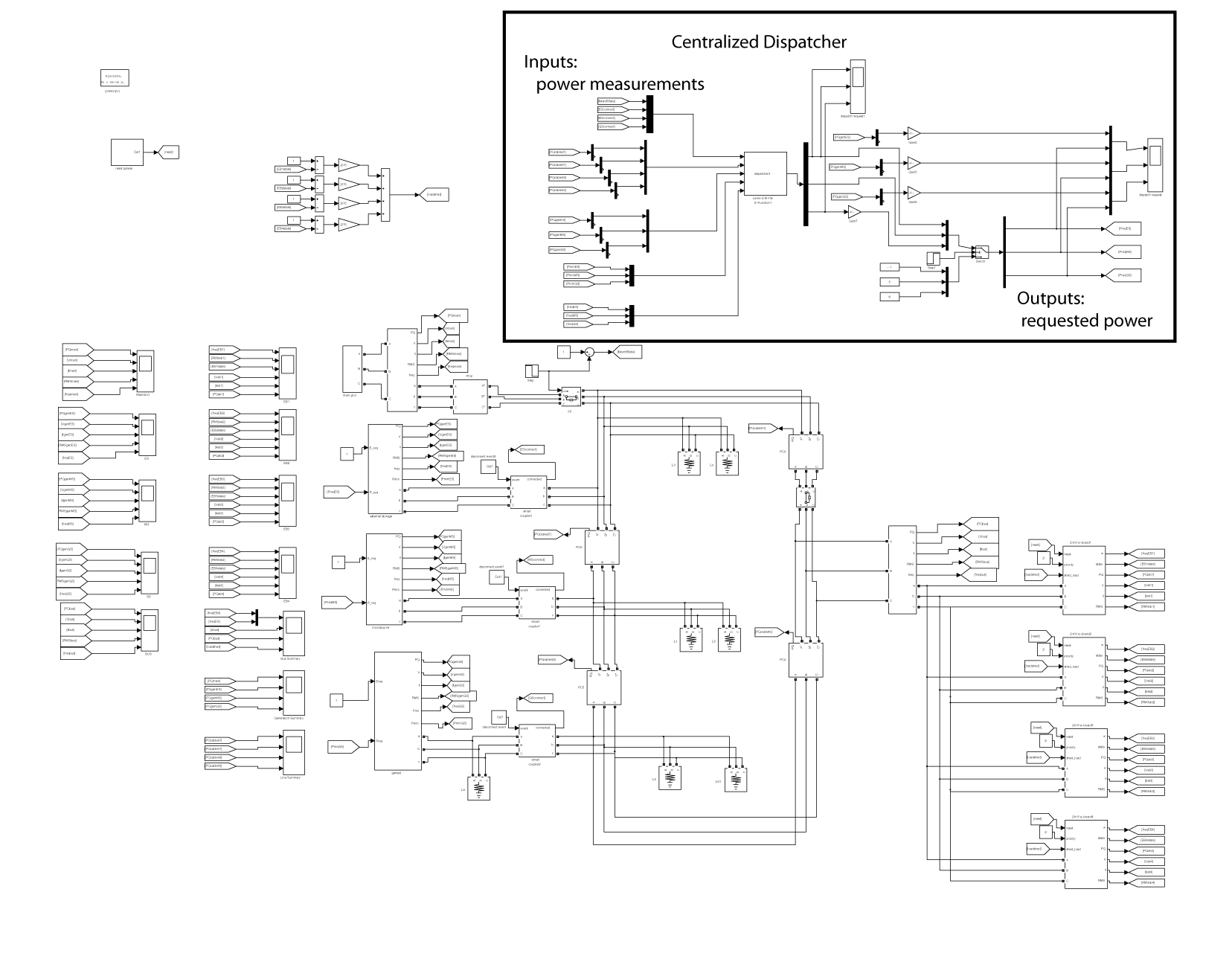


Figure 34: simPower Model for UWM simulator with centralized dispatch logic

The centralized dispatcher logic was implemented as a simPower S-function within the single component shown in figure 34. The code for this is shown below.

Function Update(block)

%block.Dwork is internal state of program (stored variables)

%block.InputPort is the input information to the program

%block.OutputPort is the output information generated by the program.

%parameters

% Cost(i) = coefficients for cost of running generator I (ND)

% PgenUlimit(i) = upper limit on generator I power (pu)

% PgenLlimit(i) = lower limit on generator I power (pu)

% PlineLimit(j) = limit on power flowing in line j (pu)

% w = coefficient used to fix solution’s tolerance

% gam = update’

%

% Sbase = base power of 15 kW (used to convert measured power to pu)

% w0 = desired line frequency in rad/sec

% A = Weighted Incidence matrix for microgrid’s graph

% B = Weighted Laplacian matrix for microgrid’s graph

Cost = [1 0 0;0 .5 0;0 0 1];

PgenUlimit = [1;1;.8333];

PgenLlimit = [-.1; 0 ; 0];

PlineLimit = [.5;.5;.5;.5];

w = .05;

gam = 20;

Sbase = 15000;

w0=2\*pi\*60;

Lline21 = .003300/w0;

Lline31 = .000640/w0;

Lline42 = .006600/w0;

Lline43 = .004070/w0;

A21 = 1.0/(Lline21\*w0\*Sbase);

A31 = 1.0/(Lline31\*w0\*Sbase);

A42 = 1.0/(Lline42\*w0\*Sbase);

A43 = 1.0/(Lline43\*w0\*Sbase);

A = [A21 –A21 0 0;

A31 0 -A31 0;

1. A42 0 –A42;

0 0 A43 -A43];

B = [ A21+A31 -A21 -A31 0;

-A21 A21+A42 0 -A43;

-A31 0 A31+A43 -A42

0 -A42 -A43 A42+A43];

%interal program variables

% Preq(i) = last computed power setpoint for generator (i) (pu) (Dwork(1))

% PgenPast(i) = last measured generator power (pu) (Dwork(2))

% timer(i) = timer for generator (i) (postive integer) (Dwork(3))

Preq = block.Dwork(1).Data;

PgenPast = block.Dwork(2).Data;

timer = block.Dwork(3).Data;

%external inputs to program

% connectState(i) = connection status of generator I InputPort(1)

% connected (1) / unconnected (0)

% Pline(j) = power through line j (pu) InputPort(2)

% Pgen(i) = power from generator (i) (pu) InputPort(3)

% freq(i) = commanded frequency of generator (i) (hz) InputPort(5)

connectState(1) = block.InputPort(1).Data(2);

connectState(2) = block.InputPort(1).Data(3);

connectState(3) = block.InputPort(1).Data(4);

Pline = block.InputPort(2).Data./Sbase;

Pgen = block.InputPort(3).Data./Sbase;

freq = block.InputPort(5).Data;

%temporary variables (link and node states)

% mu(j) = link state for link j

% phi(i) = generator i’s state

% z(i) = combined state for generator I used in updating Preq

%link state

mu = max(0,(1/w)\* (Pline-PlineLimit))+ min(0,(1/w)\*(Pline+PlineLimit));

%node state

phi = Cost\*Pgen + (1/pi)\*(freq-60)+ max(0,(1/w)\*(Pgen-PgenUlimit))+min(0,(1/w)\*(Pgen-PgenLlimit)) ;

%z state computation

z(1,1) = A(1,1)\*mu(1)+A(2,1)\*mu(2) + B(2,1)\*phi(2)+B(3,1)\*phi(3)+B(1,1)\*phi(1);

z(2,1) = A(1,2)\*mu(1)+A(3,2)\*mu(3) + B(1,2)\*phi(1)+B(4,2)\*phi(3)+B(2,2)\*phi(2);

z(3,1) = A(2,3)\*mu(2)+A(4,3)\*mu(4) + B(1,3)\*phi(1)+B(4,3)\*phi(3)+B(3,3)\*phi(3);

% change between Preq and Pgen

Pdelta = gam.\*z./pi;

Pdelta = sign(Pdelta).\*min(.5,abs(Pdelta)); %was originally 0.2 pu

%switching logic to determine when to disable the dispatcher

for i=1:1:3;

%service one-shot timer

if (timer(i)>0)

timer(i)=timer(i)-1;

end;

%disable dispatcher if

% commanded frequency (freq) < 59.5 Hz (indicate loads will be shed)

% or

% the outgoingpower has changed by more than 0.2 pu in 0.01 seconds

% If these conditions are satisfied then start one-shot timer for 0.5 sec.

if (freq(i) <= 59.5)||(abs(Pgen(i)-PgenPast(i))>=.2)

timer(i)=50;

end;

%if the timer has expired (or is not running. This occurs when timer=0)

% then you can go ahead and compute Preq (i.e. the dispatcher is active)

% IF the generator is connected,

% then use Pdelta to update Preq

% IF the generator is not connected,

% then set Preq = 0.8 (keeps it running with a large enough frequency

% so the smartswitch reconnects quickly.

%

% NOTE: if the timer is running, then Preq is not updated.

% so the dispatcher is inactive and the past Preq is used.

If (timer(i)==0);

if(connectState(i)==1);

%Preq(i) = Pgen(i) – Pdelta(i) – PmmLimit(i);

Preq(i) = Pgen(i) – Pdelta(i);

Preq(i) = max(PgenLlimit(i),Preq(i));

Preq(i) = min(PgenUlimit(i),Preq(i));

else;

Preq(i)=.8;

end;

end;

end;

%save past values

block.Dwork(1).Data = Preq;

block.Dwork(2).Data = Pgen;

block.Dwork(3).Data = timer;

%endfunction

function Output(block)

%simply output the stored Preq

block.OutputPort(1).Data = block.Dwork(1).Data;

%endfunction

**7.2 Distributed Implementation of Dispatch Algorithm for UWM Testbed**

The code in the centralized dispatcher’s S-function was written so it could be easily distributed between multiple agents. Odyssian’s programmers, however, need a more explicit description of the distributed algorithm that clearly showed how the inputs and outputs of the system were distributed. We therefore modified the implementation of section 7.1 by breaking apart the single S-function block into two smaller blocks as shown in figure 35.

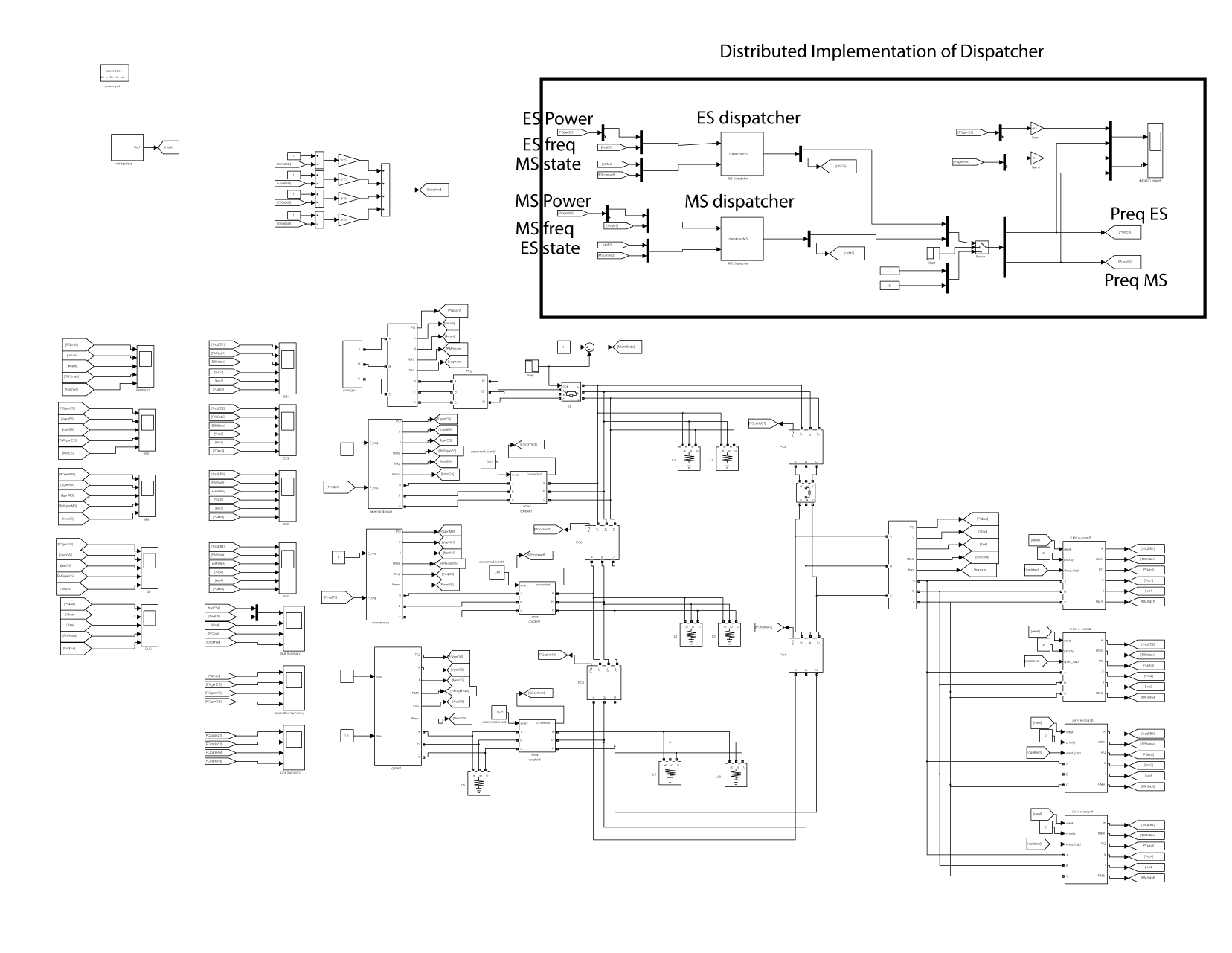


Figure 35: simPower Model for UWM Simulator with distributed dispatch logic

The blocks implementing the distributed dispatcher are marked off by the rectangle in figure 35. Compared to figure 34, we see that the S-function block for the dispatcher now consists of two S-function blocks; one for the ES source and the other for the MS source. The phi states described in section 7.1 are clearly marked as inputs, as are the local power and frequency measurements.

The S-function source code is shown below. This code was greatly restructured from what was originally delivered to Odyssian in that it adheres very closely to the data structures used by Odyssian’s actual dispatch single board computers. In this way, the simPower simulation could be used to help Odyssian debug their implementation of the proposed algorithms.

Function Update(block)

%compute parameters (this can be precomputed and stored).

%parameters

Sbase = 15000; %per unit power

w0 = 2\*pi\*60; %nominal frequency = 60Hz

w = .05; %penalty parameter (.05)

gam = 20; %step size (20)

%set parameters for optimization problem

PgenUlimit = 1; %for ES unit

PgenLlimit = -.1; %for ES unit

Cost = 1; %for ES unit

CostGS = 1;

%cable reactances (specific to UWM grid)

Lline21 = .003300/w0;

Lline31 = .000640/w0;

Lline42 = .006600/w0;

Lline43 = .004070/w0;

%Incidence Matrix

A21 = 1.0/(Lline21\*w0\*Sbase);

A31 = 1.0/(Lline31\*w0\*Sbase);

A42 = 1.0/(Lline42\*w0\*Sbase);

A43 = 1.0/(Lline43\*w0\*Sbase);

%A matrix not needed since we can’t access line power

%A = [A21 –A21 0 0;

% A31 0 -A31 0;

% 0 A42 0 –A42;

% 0 0 A43 -A43];

B = [ A21+A31 -A21 -A31 0;

-A21 A21+A42 0 -A43;

-A31 0 A31+A43 -A42

0 -A42 -A43 A42+A43];

%Fetch data from wireless link

%phi(1) = block.InputPort(1).Data(1);

phi(2) = block.InputPort(2).Data(1); %MS variable

phi(3) = CostGS\*0.8\*12.5/15; %Genset variable

connectState = block.InputPort(2).Data(2); %ES connection state

%Fetch data from serial link

Pgen = block.InputPort(1).Data(1)/Sbase; %ser->Pgen (local)

freq = block.InputPort(1).Data(2); %ser->freq (local)

%Fetch local variables in shared memory and stored locally

Preq = block.Dwork(1).Data(1); %shm->Preq

PgenPast = block.Dwork(1).Data(2); %temp->PgenPast

timer = block.Dwork(1).Data(3); %temp->timer

%link state – not needed if we don’t constraint line power

%mu = max(0,(1/w)\* (Pline-PlineLimit))+ min(0,(1/w)\*(Pline+PlineLimit));

%node state

phi(1) = Cost\*Pgen +(freq-60) + max(0,(1/2)\*(Pgen-PgenUlimit)) + min(0,(1/w)\*(Pgen-PgenLlimit));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% compute update to power req

z = B(2,1)\*phi(2)+B(3,1)\*phi(3)+B(1,1)\*phi(1);

%z(2,1) = A(1,2)\*mu(1)+A(3,2)\*mu(3) + B(1,2)\*phi(1)+B(4,2)\*phi(3)+B(2,2)\*phi(2);

%z(3,1) = A(2,3)\*mu(2)+A(4,3)\*mu(4) + B(1,3)\*phi(1)+B(4,3)\*phi(3)+B(3,3)\*phi(3);

Pdelta = gam.\*z./pi;

Pdelta = sign(Pdelta).\*min(.5,abs(Pdelta)); %was originally 0.2 pu

if (timer > 0)

timer = timer-1;

end;

if(freq <= 59.5)||(abs(Pgen-PgenPast)>=0.2)

timer = 50;

if connectState == 0

Preq = .8;

end;

end;

if (timer==0)

if(connectState==1);

Preq = Pgen-Pdelta;

Preq = max(PgenLlimit,Preq);

Preq = min(PgenUlimit,Preq);

end;

end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%save shm and local variables

block.Dwork(1).Data(1) = Preq; %shm->Preq

block.Dwork(1).Data(2) = Pgen; %temp->PgenPast

block.Dwork(1).Data(3) = timer; %temp->timer

block.Dwork(1).Data(4) = phi(1); %temp->philocal

%endfunction

**7.3 Dispatch Agent Interface Specification**

The microgrid controller being tested by Odyssian Technologies under contract W9132T-10-C-0008 has a hierarchical structure. The bottom layer of the hierarchy consists of the UWM (University of Wisconsin, Madison) controllers [1] and the top layer of the hierarchy consists of the UND (University of Notre Dame) dispatch agents [2]. The UWM controller uses a microprocessor to generate the switching signals for the power inverter connecting a generator to the microgrid’s feeder lines. The UND controller is implemented as a single-board computer with a radio module. This document specifies the hardware and data interface between the UWM controller and the UND dispatch agent.

Figure 36 uses a simPower model to illustrate the interconnections between microsource, UWM controller, and dispatch agent. This block diagram has four blocks. The microsource (generator) takes inputs Vpk and w from the UWM controller. The UWM controller uses the measured terminal voltage and currents (v\_meas and i\_meas) and the requested voltage (E\_req) and power (P\_req) to determine the voltage (Vpk) and frequency (w) command to the microsource. The requested voltage is set to 1 pu. The dispatch agent supplies the requested power (P\_req) input. To compute the requested power, the dispatch agent needs to access the measured real power (P\_meas) and commanded frequency (w). For diagnostic reasons, it will also be convenient for the dispatch agent to access the measured reactive power (Q\_meas), measured RMS voltage (V\_meas), and voltage command (Vpk). The required data that is passed between the dispatch agent and UWM controller is therefore seen in figure 36.

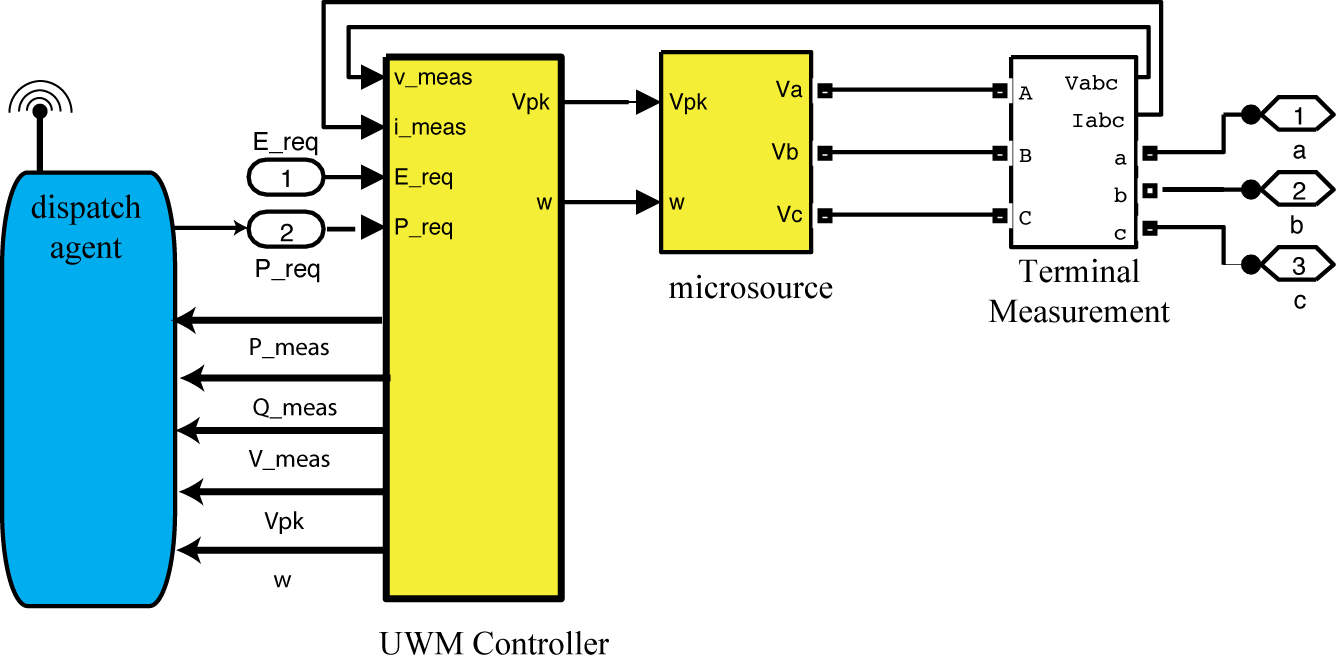


Figure 36: Interface between UWM controller and Odyssian Dispatch Agent

Figure 37 identifies the signals within the UWM controller are required by the dispatch agent. This figure shows the simulink block diagram for the UWM controller. The signals to be passed over the interface are

* P\_meas : This is the measured real power after it has been filtered and scaled to power units. This variable should be transmitted every 10 msec. Variable should be uniformly quantized to sixteen bits as a signed fixed-point number between -10 and 10.
* Q\_meas : This is the measured reactive power after it has been filtered and scaled to power units. This variable should be transmitted every 10 msec. Variable should be uniformly quantized to sixteen bits as a signed fixed-point number between -10 and 10.
* V\_meas : This is the measured RMS voltage after it has been filtered and scaled to power units. This variable should be transmitted every 10 msec. Variable should be uniformly quantized to sixteen bits as a signed fixed-point number between -10 and 10.
* Vpk : This is the voltage command sent to the microsource in units of volts. This variable should be transmitted every 10 msec. Variable should be uniformly quantized to sixteen bits as a signed fixed point number between   
  -1000 and 1000 volts.
* w : This is the frequency command sent to the microsource in units of radians/second. This variable should be transmitted every 10 msec. Variable should be uniformly quantized to sixteen bits as a signed fixed-point number between -1000 and 1000 radians per second.

The interface will therefore transmit 16\*5=80 data bits every 0.01 seconds. This corresponds to a minimum data rate of 8000 bits per second (bps).

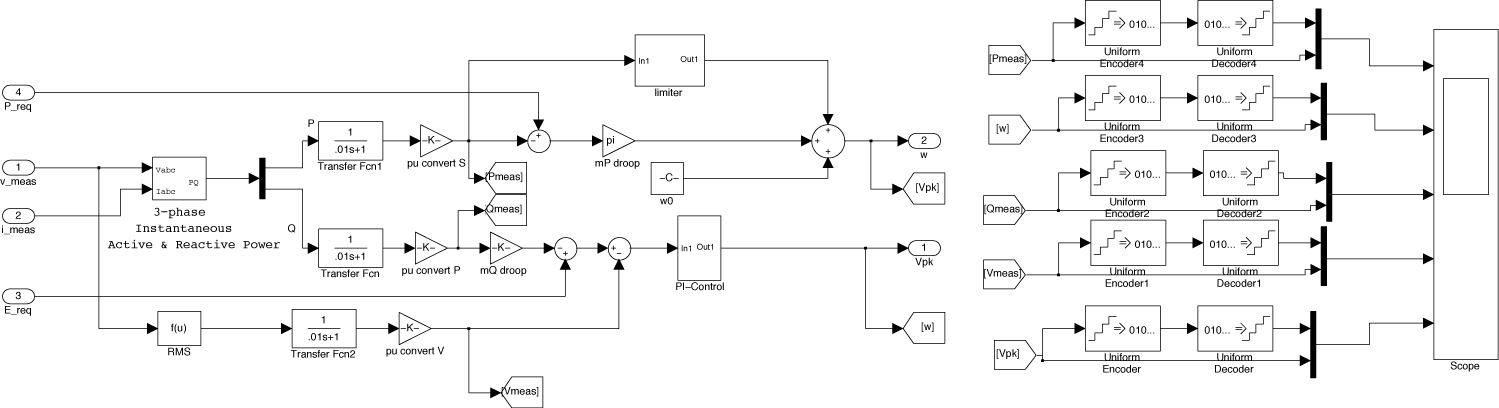


Figure 37: Signals within UWM controller required by the dispatch agent.

The physical interface between the UWM controller and the UND dispatch agent will adhere to a traditional RS-232 serial computer interface standard. It is recommended that a 9-pin connector be used with a baud rate of 115.2 kbps. Raw data bits obtained from the quantization should be transmitted, rather than transmitting as ASCII characters.

# Chapter 8: Testing of Algorithms on UWM Simulation

This chapter describes the simulation testing of the UWM simulator for the combined dispatch and load-shedding algorithms described in chapters 6 and 7. Section 8.1 discusses the simulation testing for the centralized dispatcher (figure 34). Section 8.2 briefly describes some additional work to assess the scalability of the UWM control algorithms.

**8.1 Centralized Dispatcher with Automatic Load Shedding**

The preceding load-shedding and dispatch logic was implemented in the simPower simulation that Notre Dame has built for the UWM microgrid testbed. This section discusses some of the results in that simulation. A detailed picture of the simPower chart is shown below in figure 38. This is, essentially, the same simPower model shown in figure 34.

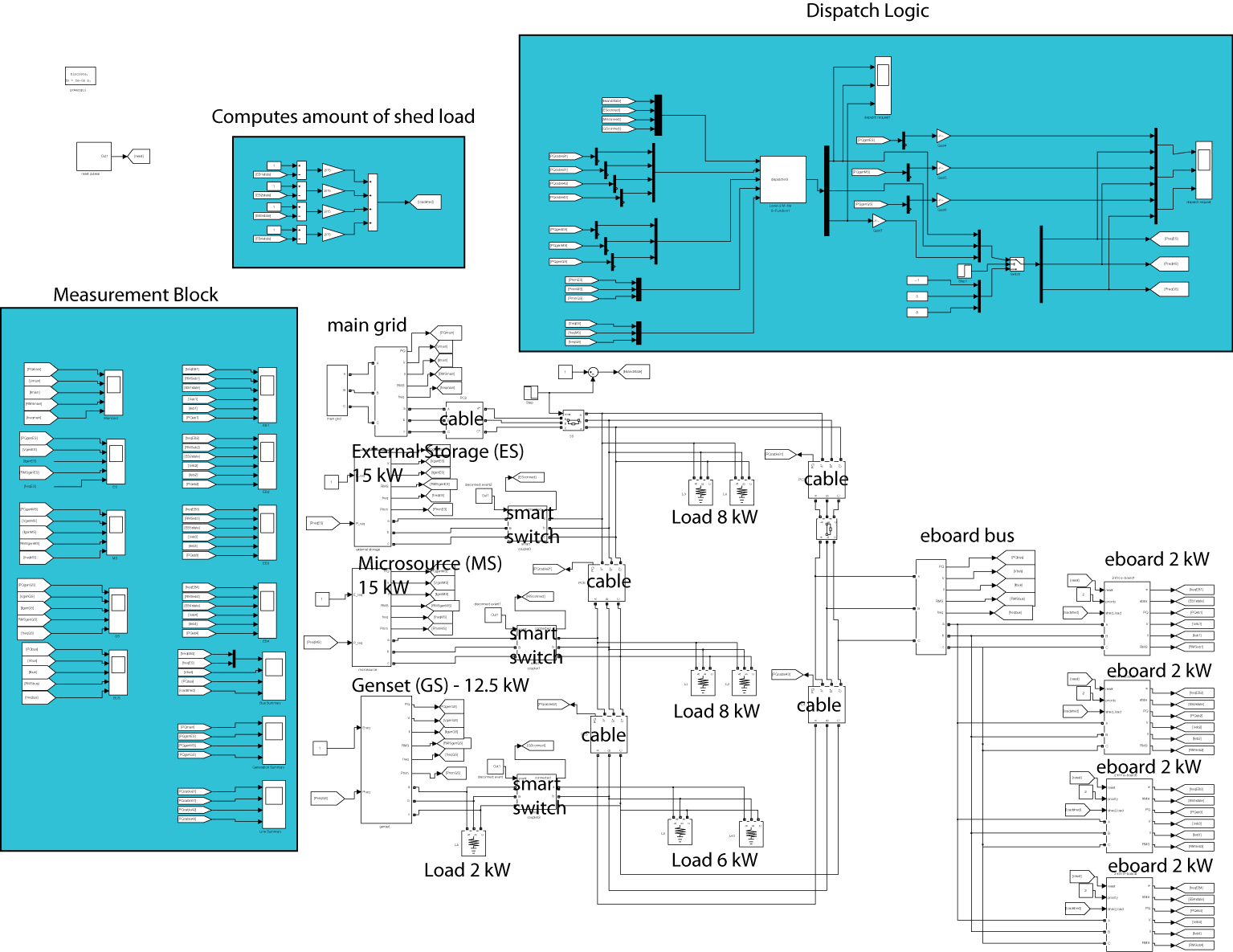


Figure 38: simPower Model for UWM simulator (centralized dispatcher and load shedding modules)

Simulations were conducted on the following scenario

1. Simulation starts connected to main grid (0 sec).
2. Microgrid islanding event (1 sec).
3. Loss of microsource generator (3 sec).
4. Microsource released for reconnection (3.5 sec).

Four cases were considered with these simulation: case

1. no dispatcher with reconnection of shed loads at 60 Hz,
2. dispatcher starts at 0.1 seconds with reconnection of shed loads at 60 Hz.
3. no dispatcher with adaptive reconnection of shed loads
4. dispatcher starts at 0.1 seconds with adaptive load reconnection.

In all cases, we plot an 8 second time history of the

1. ES command frequency and line frequency estimated at e-board 4 (Hz).
2. The instantaneous voltages of all three phases at the e-board bus (V).
3. The real (yellow) and reactive (blue) power flowing into the e-board bus (W).
4. and the amount of shed load power (pu).

For those cases where the dispatcher was enabled, we also plot 8 second time histories of *Pgen* and *Preq* for the ES, MS, and GS sources.

**Case 1:** *no dispatcher with reconnection of shed loads at 60 Hz*

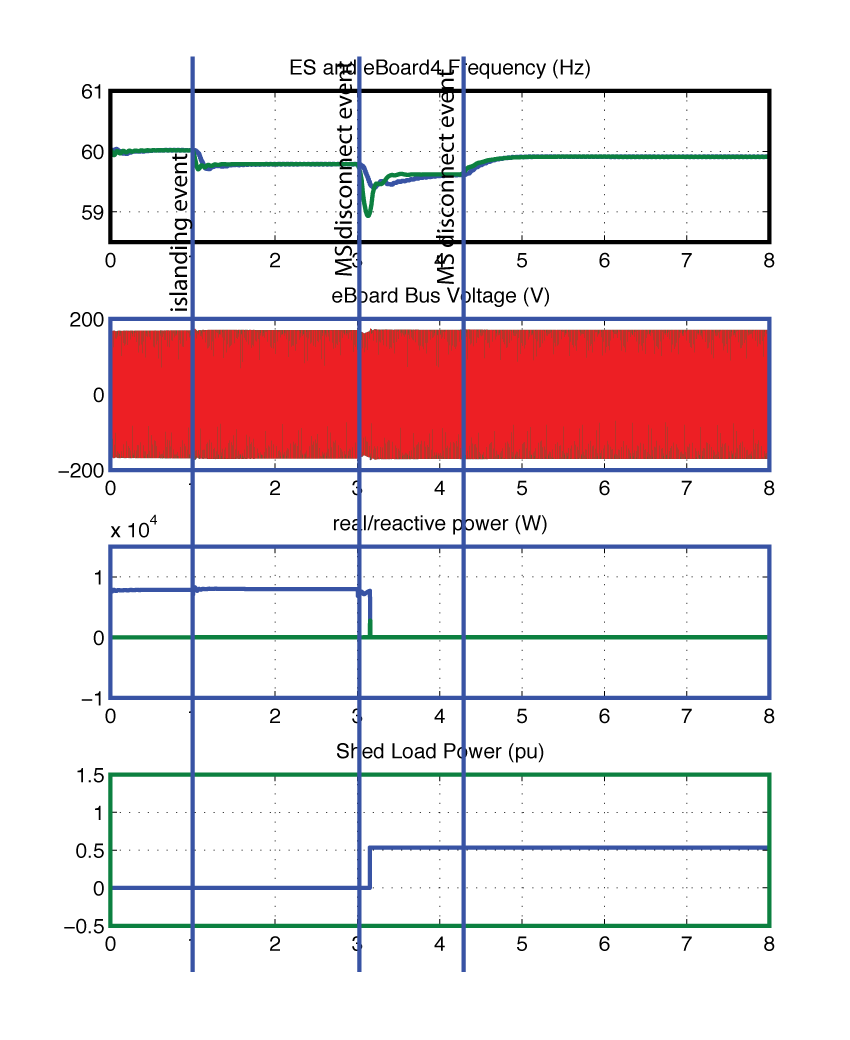
In this simulation case, *Preq* , is initially set to -0.1 pu for the ES, 0.5 pu for the MS, and .8 pu for the GS. These values are held constant through the simulation since the dispatcher is not used. The results are shown in figure 39.

Figure 39: Simulation results for case 1

In this simulation, the e-board’s shed their loads if the line frequency at the e-board’s terminal drops below 59.5 Hz. The loads automatically reconnect if the estimated line frequency exceeds 60 Hz.

In this case, the eboard loads are dropped when the estimated line frequency drops below 59.5 Hz. After the MS reconnects to the microgrid, however, the line frequency never rises about 60 Hz, so the shed loads never reconnect.

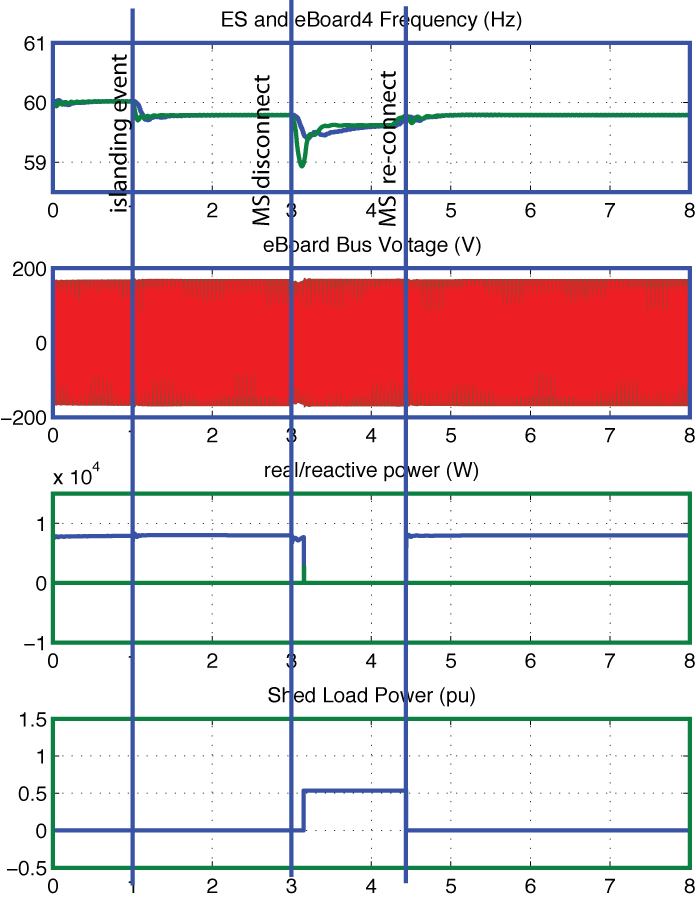
**Case 2:** *no dispatcher with adaptive reconnection of shed loads*

Figure 40: Simulation results for case 2

In this simulation case, *Preq* , is initially set to -0.1 pu for the ES, 0.5 pu for the MS, and .8 pu for the GS. These values are held constant through the simulation since the dispatcher is not used. The results are shown in figure 40.

In this simulation, the e-board’s shed their loads if the line frequency at the e-board’s terminal drops below 59.5 Hz. The loads automatically reconnect at a frequency that is determined by the total amount of load that has already been shed.

In this case, the load reconnects almost as soon as the MS reconnects to the microgrid.

**Case 3:**  *dispatcher with reconnection of shed loads at 60 Hz*

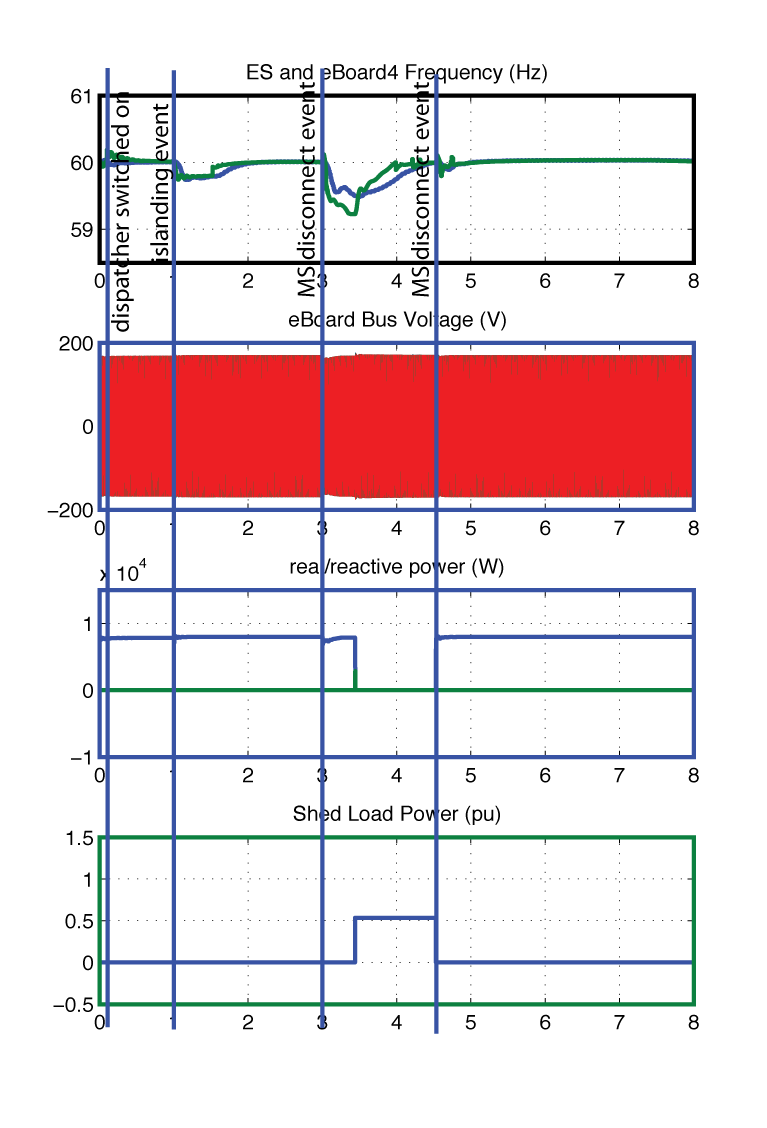
In this simulation case, *Preq* , is initially set to -0.1 pu for the ES, 0.5 pu for the MS, and .8 pu for the GS. At 0.1 seconds, the dispatcher is turned on and the dispatcher adaptively manages *Preq* throughout the rest of the simulation. The results are shown in figure 41 and 42.

Figure 41: simulation results for case 3

From figure 41, we see that the e-boards shed their loads if the line frequency at the e-board’s terminal drops below 59.5 Hz. The loads automatically reconnect if the estimated line frequency exceeds 60 Hz.

In this case the e-board loads are shed when the estimated line frequency drops below 59.5 Hz. The loads reconnect when the line frequency exceeds 60 Hz. This differs from case 1, because the dispatcher returns the system to its 60 Hz frequency. This can actually be seen in the frequency traces after the islanding event.

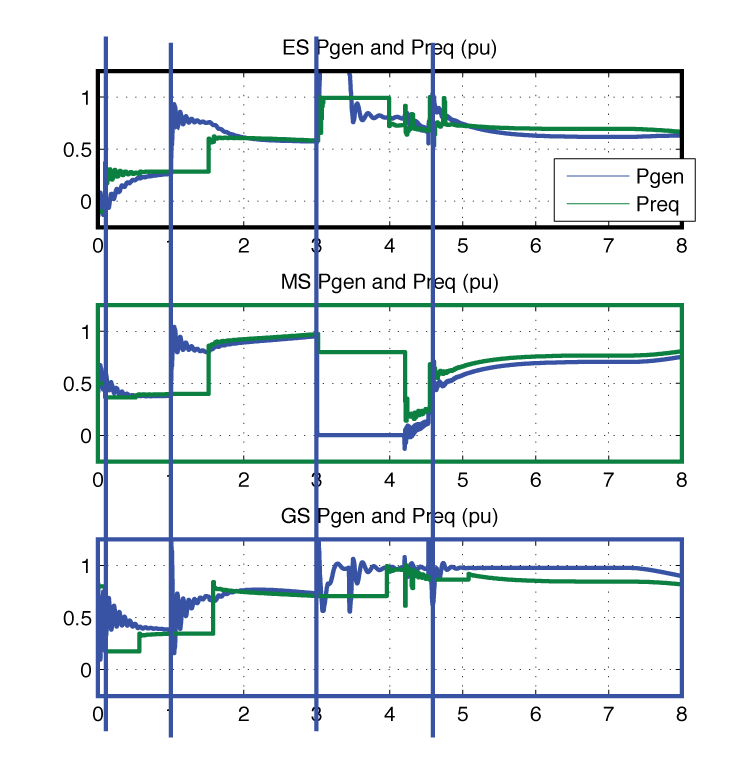
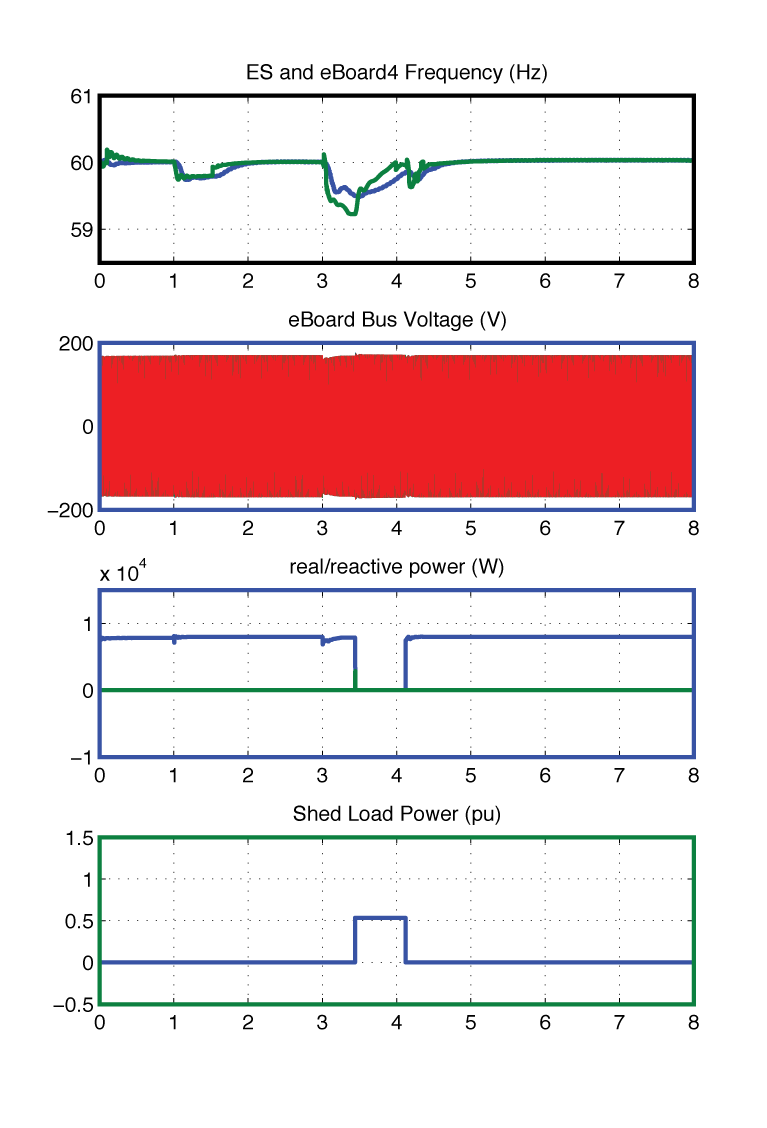


Figure 42 shows *Pgen* and *Preq* for all three sources. We see the generated power trying to track the commanded (requested) power computed by the dispatcher. Since these powers track each other the dispatcher appears to be working correctly.

Figure 42: simulation results for case 3 showing how the requested and generated power track each other

**Case 4:**  *dispatcher with adaptive reconnection of shed loads*

In this simulation case, *Preq* , is initially set to -0.1 pu for the ES, 0.5 pu for the MS, and .8 pu for the GS. At 0.1 seconds, the dispatcher is turned on and the dispatcher adaptively manages *Preq* throughout the rest of the simulation. The simulation results are shown in figure 43 and 44.

From figure 43 we see that the e-boards shed their loads if the line frequency at the e-board’s terminal drops below 59.5 Hz. The loads automatically reconnect at a frequency that is determined by the total amount of load that has already been shed. Figure 44 shows that the commanded and actual power levels track each other. The performance of these simulations is comparable to that in case 3 thereby indicating that the load shedding and dispatch logic are working as expected.

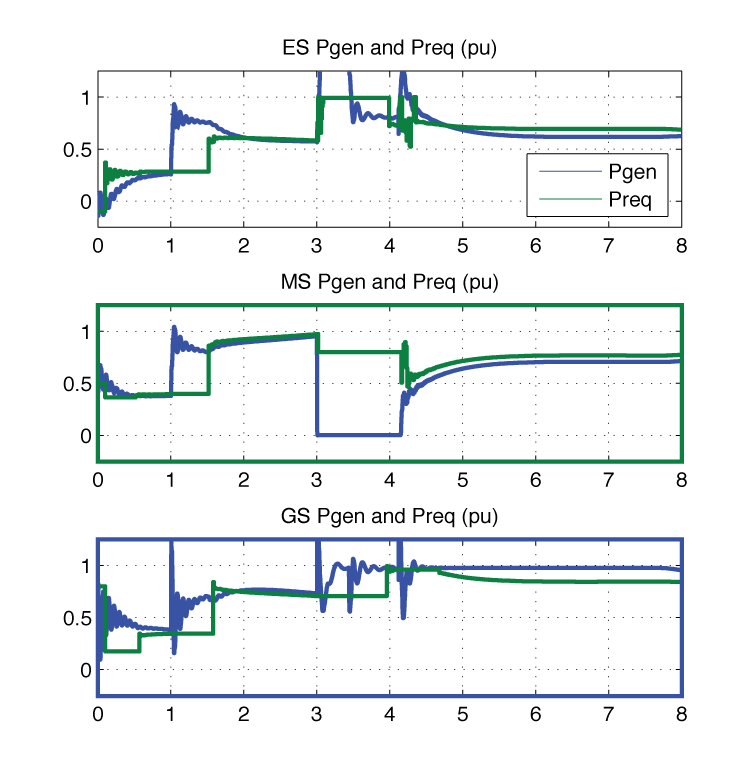


Figure 44: simulation results for case 4 showing how requested and generated power track each other

Figure 43: simulation results for case 4

**8.2 Large Scale Simulation Results**

A “large-scale” 3-phase of an inverter-based microgrid was built and tested. 15 of ND’s simPower models for the UWM microsources were interconnected into the mesh topology shown below. Each microsource had a capacity of 15 kW with *Preq* = 0.3 p.u. and *Ereq*=1.0 p.u. A total of 12 kW was attached to each generator bus, with 4 kW of load being an e-board bus that could be automatically shed if the frequency drooped below 59.5 Hz. Figure 45 shows the layout of this “large” microgrid.

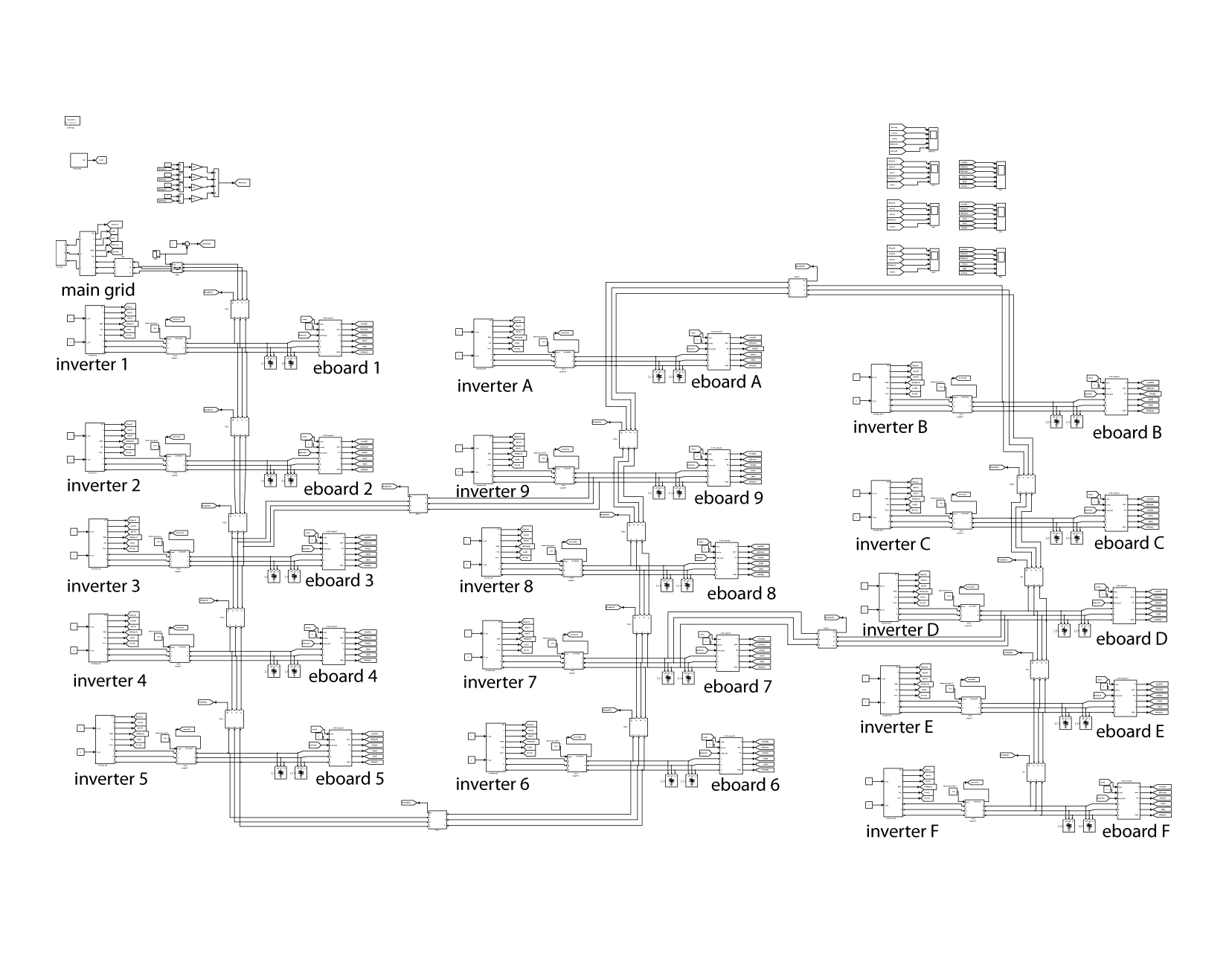
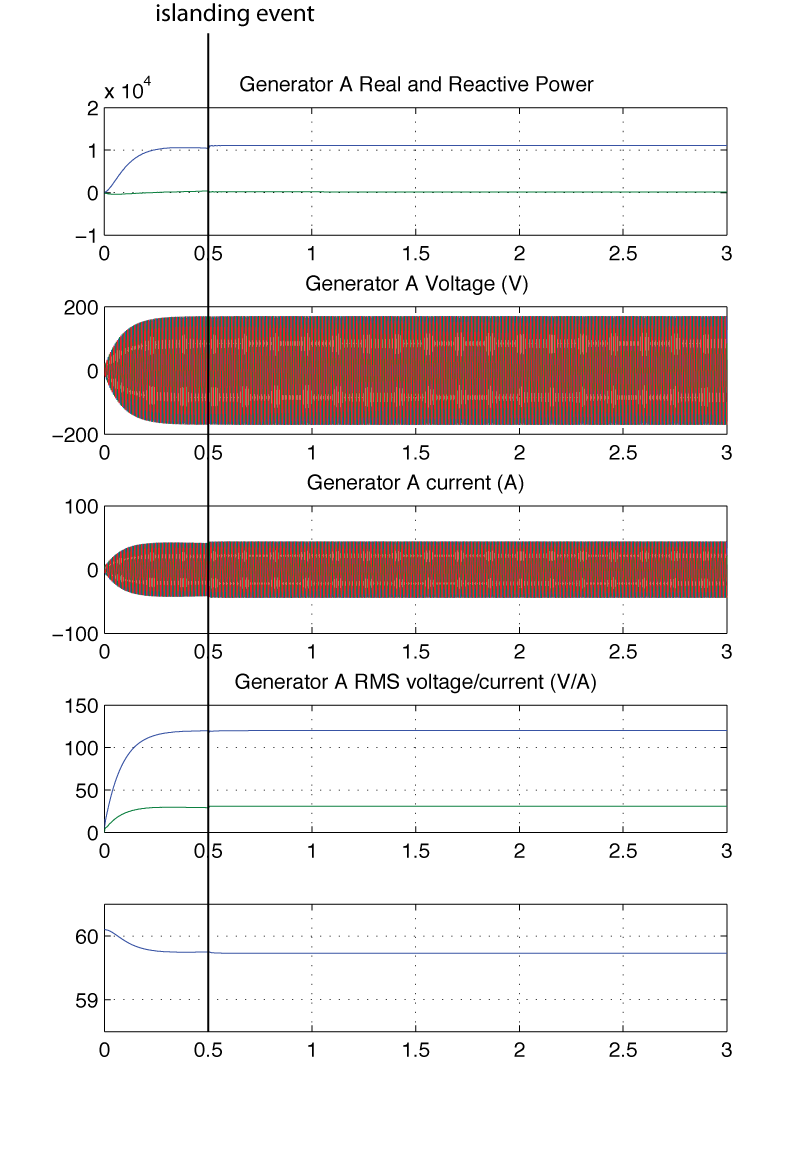


Figure 45: large-scale 3-phase microgrid and simulation results

The simulation results are shown in figure 45. The simulation starts with all sources (including the main grid) connected. At 0.5 seconds the microgrid islands. As can be seen from the plots, the voltage magnitude and frequency are preserved through the islanding event. These results suggest that the fast microsource inverter controls will retain their stability over a wide range of grid topologies.

# Chapter 9: Conclusions

This report documents the work performed by the University of Notre Dame from July 1st 2010 to October 1st 2011. During this period the following tasks were completed

* Develop single-phase and three-phase simulation for candidate microgrids
* Develop models of the e-Board load controllers in consultation with Odyssian Technology Integrate load controller model into the microgrid layouts specified by UW
* Develop distributed power dispatch algorithms for Odyssian’s lab-scale microgrid system
* Extend power dispatch algorithms to account for limits on transmission line power
* Work with Odyssian to continue developing load shedding algorithm
* Integrate distributed load-shedding and power dispatch algorithms into the simulation using realistic model of communication network
* Evaluate scalability of the proposed intelligent hybrid control architecture
* Specify the interface between the power dispatch agents and the laboratory test inverter
* Specify the interface between the load shedding agents and the e-Board load controller
* Assist Odyssian to integrate communication network with algorithms and improve reliability and performance
* Assist in developing the embedded software implementing the power dispatch and load shedding algorithms on the wireless agent modules
* Determine what information must be transmitted between cooperating agents in the system

Two tasks described in the statement of work were not fully completed.

* Validate simulation against a bench-scale hardware microgrid:  
  The UND’s simulator of the UWM testbed was validated against UWM’s own simulation of its hardware. We were, however, unable to validate the simulation of Odyssian’s bench scale microgrid because as of September 2011, we had not been supplied with a precise characterization of that microgrid. UND had been consulting with UIUC up to August, helping them understand how the UWM controller might function with their micro-inverters. But no complete model with hardware test data was received from UIUC.
* Develop automated load forecasting algorithms in concert with power quality measurements to ensure stability:  
  The development of automated load forecasting algorithms required additional information regarding the variable nature of the loads. This information was not available and so there was no basis for developing load forecasting.

References

[1] R. Lasseter and P. Piagi, “Microgrid: a conceptual solution”, in *Power Electronics Specialists Conference, 2004, PESC 04, 2004 IEEE 35th Annual,* volume 6, June 2004, pages 4285-4290

[2] R. Lasseter, “Control and Design of Microgrid Components”, Final Project Report, PSERC publication 06-03, January 2006

[3] F. Katiraei, M.R. Iravani, and P.W. Lehn, “Micro-grid autonomous operation during and subsequent to islanding process”, IEEE Transactions on Power Delivery, volume 20, number 1, pp 248-257, 2005

[4] P. Wan and M.D. Lemmon (2010), Optimal power flow in microgrids using event-triggered optimization , American Control Conference, Baltimore, USA, 2010

[5] F. Kelly, A. Maulloo, and D. Tan, “Rate control for communication networks: shadow prices, proportional fairness and stability,” *Journal of the Operational Research Society*, vol. 49, no. 3, pp. 237–252, 1998.

[6] S. Low and D. Lapsley, “Optimization flow control, I: basic algorithm and convergence,” *IEEE/ACM Transactions on Networking (TON)*, vol. 7, no. 6, pp. 861–874, 1999.

[7] J. Wen and M. Arcak, “A unifying passivity framework for network flow control,” *Automatic Control, IEEE Transactions on*, vol. 49, no. 2, pp. 162–174, 2004.

[8] D. Palomar and M. Chiang, “Alternative Distributed Algorithms for Network Utility Maximization: Framework and Applications,” *Automatic Control, IEEE Transactions on*, vol. 52, no. 12, pp. 2254–2269, 2007.

[9] Tabuada, “Event-triggered real-time scheduling of stabilizing control tasks,” *IEEE transactions on automatic control*, vol. 52, no. 9, p. 1680, 2007.

[10] X. Wang and M. Lemmon, “Self-triggered feedback control systems with finite-gain l 2 stability,” *IEEE transactions on automatic control*, vol. 54, p. 452, 2009.

[11] X. Wang and M. Lemmon, “Event-triggering in distributed networked systems with data dropouts and delays,” in *Proceedings of Hybrid Systems: computation and control*, 2009.

[12] P. Wan and M.D. Lemmon (2009), An event-triggered distributed primal-dual algorithm for network utility maximization, IEEE Conference on Decision and Control (CDC), Shanghai, PRC, December 2009.

P. Wan and M. Lemmon (2009), Event triggered distributed optimization in sensor networks , Information Processing in Sensor Networks (IPSN), 2009.

 P. Wan and M. Lemmon (2009), Distributed Network Utility Maximization using Event-triggered augmented Lagrangian methods, American Control Conference, 2009.

Appendix: Matlab/Simulink/SimPower Components

This appendix documents the Matlab version number and simulation components that were used in this project. The version of Matlab that was used for these simulations is given below. The key components are marked in bold.

------------------------------------------------------------------------------

MATLAB Version 7.10.0.499 (R2010a)

MATLAB License Number: 553528

Operating System: Mac OS X Version: 10.6.8 Build: 10K549

Java VM Version: Java 1.6.0\_26-b03-384-10M3425 with Apple Inc. Java HotSpot(TM) 64-Bit Server VM mixed mode

-------------------------------------------------------------------------------------

MATLAB Version 7.10 (R2010a)

Simulink Version 7.5 (R2010a)

Bioinformatics Toolbox Version 3.5 (R2010a)

Communications Blockset Version 4.4 (R2010a)

Communications Toolbox Version 4.5 (R2010a)

Control System Toolbox Version 8.5 (R2010a)

Curve Fitting Toolbox Version 2.2 (R2010a)

Database Toolbox Version 3.7 (R2010a)

Datafeed Toolbox Version 3.5 (R2010a)

Econometrics Toolbox Version 1.3 (R2010a)

Filter Design HDL Coder Version 2.6 (R2010a)

Filter Design Toolbox Version 4.7 (R2010a)

Financial Derivatives Toolbox Version 5.5.1 (R2010a)

Financial Toolbox Version 3.7.1 (R2010a)

Fixed-Income Toolbox Version 1.9 (R2010a)

Fixed-Point Toolbox Version 3.1 (R2010a)

Fuzzy Logic Toolbox Version 2.2.11 (R2010a)

Global Optimization Toolbox Version 3.0 (R2010a)

Image Processing Toolbox Version 7.0 (R2010a)

Instrument Control Toolbox Version 2.10 (R2010a)

MATLAB Builder JA Version 2.1 (R2010a)

MATLAB Compiler Version 4.13 (R2010a)

MATLAB Report Generator Version 3.8 (R2010a)

Mapping Toolbox Version 3.1 (R2010a)

Model Predictive Control Toolbox Version 3.2 (R2010a)

Neural Network Toolbox Version 6.0.4 (R2010a)

Optimization Toolbox Version 5.0 (R2010a)

Parallel Computing Toolbox Version 4.3 (R2010a)

Partial Differential Equation Toolbox Version 1.0.16 (R2010a)

RF Toolbox Version 2.7 (R2010a)

Real-Time Workshop Version 7.5 (R2010a)

Robust Control Toolbox Version 3.4.1 (R2010a)

Signal Processing Blockset Version 7.0 (R2010a)

Signal Processing Toolbox Version 6.13 (R2010a)

SimEvents Version 3.1 (R2010a)

SimMechanics Version 3.2 (R2010a)

SimPowerSystems Version 5.2.1 (R2010a)

Simscape Version 3.3 (R2010a)

Simulink 3D Animation Version 5.1.1 (R2010a)

Simulink Control Design Version 3.1 (R2010a)

Simulink Design Optimization Version 1.1.1 (R2010a)

Simulink Fixed Point Version 6.3 (R2010a)

Simulink Report Generator Version 3.8 (R2010a)

Spline Toolbox Version 3.3.8 (R2010a)

Stateflow Version 7.5 (R2010a)

Statistics Toolbox Version 7.3 (R2010a)

Symbolic Math Toolbox Version 5.4 (R2010a)

System Identification Toolbox Version 7.4 (R2010a)

Video and Image Processing Blockset Version 3.0 (R2010a)

Wavelet Toolbox Version 4.5 (R2010a)

The simulations were developed using simPower (SimPowerSystems) and the directories used in these simulations will be found in the attached file (power\_simulations.zip).

This is a large file (221 MB) and will be posted temporarily to ND’s project website

<https://www.nd.edu/~lemmon/projects/Odyssian-2009/vault.html>

Within this file there are the following subdirectories

* event\_trigger  
  This directory contains the simPower models used in generating the results for the event-triggered dispatch algorithm described in chapter 2 and 3.
* uwm\_verification  
  This directory contains the simPower models used to validate UND’s simulation against the UWM microgrid. This directory contains subdirectories case1, case2, and case3, which are the three test cases we validated against. The other subdirectories es\_test, ms\_test, and genset\_test simulate the control of a single microsource (ms), external storage (es), or generator (genset). These are the simulations described in chapter 4 of this report.
* load\_shedding  
  This directory contains the simPower models used in developing the PLL frequency estimator, early versions of the load shedding logic, the e-board simulation model, and the smart-coupler component. The base files from which these components were built will be found in subdirectory case3A. The other two subdirectories, ms\_test, ms\_test\_module contain these new components. These simulation components are described in chapter 6 (sections 6.1 and 6.2).
* single-phase-sims  
  This directory contains the simPower models for the single-phase Odyssian bench scale testbed. The first version of these simulations will be found in subdirectory odyssian-single-phase. A later version, which contains the files used in generating the project reports will be, find in subdirectory odyssian-single-phase2. These simulations are described in chapter 5.
* uwm-final-simulation  
  This directory contains the simPower models used to test the distributed dispatch and load shedding algorithms on the UWM testbed. The load shedding algorithms are described in section 6.3, the dispatcher is described in chapter 7, and the simulations results are described in chapter 8. The subdirectory case3A is the full simulation with the centralized dispatcher. A condensed form of this directory (removing unnecessary files) will be found in subdirectory case4. The subdirectory case5-scaling contains the models used in the large-scale network described in section 8.2. The subdirectory contains the simulations that were run based on updated information about the UWM microgrid for the components Odyssian actually delivered to UWM.