I2SIM FINANCIAL MODEL AND ITS APPLICATION TO UBC'S LIVING LAB PROJECTS

by

Rui Ren

B.A.Sc., The University of British Columbia, 2006

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

The Faculty of Graduate Studies

(Electrical and Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA
(Vancouver)

December 2011

Abstract

The Infrastructure Interdependencies Simulator (I2Sim) enables the user to explore the relationship and interdependencies among different infrastructures. Based on the same I2Sim ontology, and derived from a regular production cell, the I2Sim financial production cell (FPC) is designed to simulate and present the financial behaviors.

UBC's living lab model and living lab battery model provided an ideal testing ground for the I2Sim environment and its new financial production cell. In the living lab model, two of the financial production cells were integrated into the system, and provided with real time dynamic inputs. As expected, the financial production cell outputs correctly the cost of purchased resources as well as the ongoing real time costs. The results of this simulation successfully demonstrated the strength of the I2Sim environment and the capabilities of the financial production cell.

In the living lab battery model simulation, two cases were tested. In the first test case, with demand peak-shaving as the main objective, an entire battery system model was developed from the ground up and finally integrated with the financial production cell. The simulation provided accurate financial analysis towards three different battery types: Flow batteries, lithium-ion batteries and sodium sulfur batteries. Although the results suggest that the battery system is not the most economical electric energy storage system at the moment, it again proved that the financial production cell is a suitable tool for many different business analysis cases. In the second case, delaying south transmission line's upgrading process became the main objective; the test results showed that the flow battery was the best choice. The project was proven to be both technically and financially feasible.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank my supervisor, Professor Dr. Jose R. Marti, for his supervision and guidance during my graduate study years. Dr. Marti has not only broadened my knowledge towards the power engineering field, but introduced the multi-infrastructure interdependencies simulation concept to me as well. I also like to thank Dr. K.D. Srivastava for his help and valuable suggestions during our weekly team meetings over several projects with other undergraduate students.

I would like to thank Cesar Lopez for helping me to implement the Financial Production Cell. During the past two years, I spent many memorable hours with the people who study and work in the Electric Power and Energy System (EPES) lab, they are: Dr. Paul Lusina, Mohammed Talat Khouj, Marco Gonzalez-Rojas, Dr. Hafiz Abdur Rahman, Alexander De Maeseneer, Dr. Tom De Rybel, Mehrdad Chapariha, Hamid Atighechi, Amir Rasuli, Sina Chiniforoosh, Mehmet Sucu and others. We had a good time together.

I am also feeling very grateful for the help from Kristina Welch. She has spent many of her valuable hours to help me out with the financial modeling questions, as well as the important data that she provided us for the living lab project.

I appreciate Brook Dermody, Mohammed Sami Rahyyem, Amit Mahal and Sunil Bhati, as well as the students from material engineering department for their excellent work on the UBC Living Lab battery testing project.

Finally, I appreciate UBC's Living Lab committee for supporting the living lab project we were working on.

DEDICATED TO MY PARENTS

1 Introduction

The subject of this thesis is I2Sim's financial model and its application to UBC's living lab project. The financial model introduces a new toolbox module called Financial Production Cell. With this new member added to the I2Sim toolbox family, it enhances I2Sim's capability of modeling financial infrastructures.

1.1 Background

With the advance of new technologies and increase of living standards, modern society has become more and more dependent on resources like electricity, clean water, heat, natural gas, etc. In order to deliver these resources to every corner of society, the system of critical infrastructures (power grid, water network, natural gas pipes) has become the artery system of our everyday life. On top of these, the health system, the telecommunications network and the traffic system are also vital. Although each system has its very own emergency response procedures, most of time when a disaster strikes, all these systems fail to collaborate with each other and do not respond as effectively as planned. The reason is that even though every single system has its private data, it is not willing to share it, which makes it very difficult to monitor and control the entire disaster response operation.

1.2 I2Sim

The idea behind the Infrastructures Interdependencies Simulation (I2Sim) is to build a dynamic multi-system simulator to test all infrastructure interactions under disaster situations. During these tests, system weaknesses are exposed, which will provide first responders good knowledge

of how to plan ahead. I2Sim can also be used to explore critical interdependencies between particular systems, which can provide valuable reference during the emergency time [**1**].

1.2.1 I2Sim Ontology

Different infrastructures have their own characteristics and natures, which makes it difficult to model many infrastructures with one principle. I2Sim defines a fundamental ontology that is very general and can be applied to many different processes [**2**].

The basic elements of the I2Sim ontology include: token, cell and channel.

- Token represents the unit that travels through the I2Sim model. It can be consumed as input (electricity, natural gas, water, etc.) or produced as output (steam, merchandise, etc.).
- Cell is a production unit that receives inputs and produces outputs (a steam plant, a factory, etc.)
- Channel carries and transports the tokens in I2Sim models (electric transmission line, water pipe, transit system, etc.)

In I2Sim modeling, cells and channels are the basic infrastructure elements [**3**]. One individual cell can represent one physical building or a group of buildings, which have the same functionality. For instance, a hospital or three steam boilers can be modeled by using only one cell. If a building or an infrastructure possesses multiple functionalities, it is too complex to be modeled by one cell. In this case, multiple cells and channels can be grouped together to forge a sub-system to fulfill the purpose.

Based on the ontology, the existing I2Sim toolbox includes the following units [**4**]:

- Production Cell
- Delay Channel
- Distributor
- Modifier Cell
- Visualization Panel
- Storage
- Aggregator
- Source
- Control Panel
- Probe

Production Cell Produces output tokens based on input resources. The input and output relationship is governed by the Human Readable Table (HRT).

Storage stores tokens and releases them at a user-defined rate.

Delay Channel delays token travel speed from one block to another.

Aggregator aggregates tokens of the same kind into a single input.

Distributor distributes an output token to multiple other cells.

Source creates tokens, which are the inputs for other cells.

Modifier Cell applies weight factors to inputs and relates the result to a known curve to produce output.

Control Panel is the user interface that controls the simulation.

Visualization Panel shows the simulation results to users.

Probe collects data, which are plotted in the visualization panel.

In this thesis, a new module called **Financial Production Cell** will be introduced. The details of the Financial Production Cell will be discussed later in Chapter 2.6.

In the I2Sim ontology, the model contains multiple layers. The components on the same layer share the same functionality and server the same purpose. For example, all the components in the physical layer represent the physical conditions of the model; all the components on the communication layer are responsible for the communications network; and the decision layer gives commands to the other layers. The financial production cell introduced in this thesis belongs to the financial layer, and it simulates financial activities.

1.2.2 2010 Vancouver Winter Olympics I2Sim Model

One recent example of I2Sim application is the 2010 Winter Olympics Simulator. In this simulator, the Olympic Venues and critical buildings (BC Place, GM Place, Vancouver General Hospital, St. Paul's Hospital, etc.) were modeled using the I2Sim environment; Figure 1 shows the entire model in detail. Several testing conditions were applied to the model, and by analyzing the simulation results, the vulnerabilities of the system as well as the interdependencies between them were discovered [**5**].

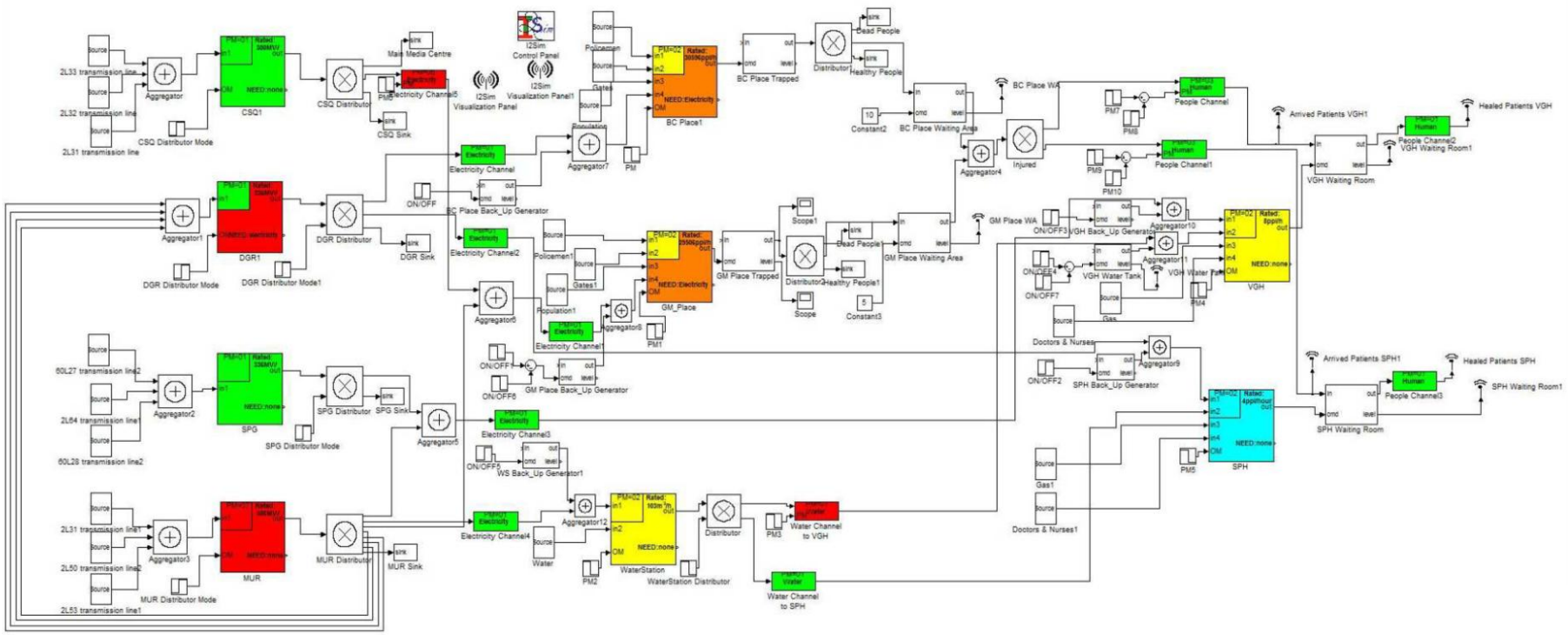


Figure 1: Vancouver winter Olympics I2Sim model [5]

2 UBC's Living Lab Project

With the growing concern of green house gas (GHG) emission, climate change and the menace of global warming, the province of British Columbia is committed to participate in aggressive GHG reduction. The goal is to reduce 33% of GHG emission by the year 2020, and 80% by the year 2025. As the top university in BC, UBC takes a role as international leader in developing and demonstrating energy savings and clean technology initiatives [**6**], which focus on environmentally sustainable, energy self-sufficient and carbon emission free initiatives.

Located on Point Grey of the west coast of Canada, UBC's Vancouver campus has a gorgeous ocean view and more than 400 hectares in size. The beauty of the campus and the reputation of the University (UBC is consistently ranked in the top 40 universities in the world) has attracted 37,000 undergraduate and 9,000 graduate students to come to study in the Vancouver campus [**7**]. In order to provide a comfortable academic and research environment, UBC offers housing services to its students, faculty and staff. In the year 2010, UBC had 15,500 residents, and by the year 2025, the population is expected to reach 28,100 [**6**].

Figure 2: UBC Vancouver campus [8**]**

The large population requires many services. Thus, UBC hospital, police station, north and south electric substations and steam house are all located on campus. This self-sufficient environment makes UBC the perfect testing ground to conduct urban sustainability experiments. Under these circumstances, UBC's living lab project was born, one of its main objective is to reduce campus greenhouse gas emissions from the year 2007 level to 33% by year 2015, 67% by year 2020, and 100% by year 2050 [**6**].

Most GHG emissions created on campus are caused by natural gas usage. A large amount of natural gas is consumed for campus' heating. Currently, the operational heat source in UBC's Vancouver campus is an aged steam plant. Natural gas is the primary fuel to create the steam. This plant suffers not only from large GHG emissions, but also from energy inefficiency [**9**]. If UBC is planning to reach its GHG emission reduction goal by the year 2050, natural gas usage must be greatly reduced.

2.1 UBC Hot-Water Based Heating System Project

In 2011, a new $85-million hot water based heating system will be developed to replace the aging steam heating system. This project will take five years to implement and will cover more than 100 buildings around the campus. Compared with the current steam based heating system, which operates at 190℃, the new hot-water based heating system will operate at an average 80℃. This noticeable difference in operating temperature can reduce the campus energy usage by 24% and the GHG emission by 22% [**10**].

At the moment the planned source of energy for the new hot-water heating system is based on a biomass boiler and a natural gas boiler [**11**]. However, an argument can be made for the use of clean electricity to feed electrical boilers instead of using gas boilers. A comprehensive study

analyzing this option is made on a companion Master Thesis by César López [**12**], where the emphasis is on GHG reduction while maintaining fuel costs at similar levels as with gas boilers.

2.1.1 Nexterra Biomass Plant

The Nexterra biomass boiler is a Combined Heat and Power (CHP) boiler. It is fueled with urban wood waste and converted into clean burning, combustible synthetic gas. The synthetic gas is then fed into a high-efficiency GE internal combustion engine to generate 2 MW of electricity, and 4 MW of heat for the campus heating system [**13**].

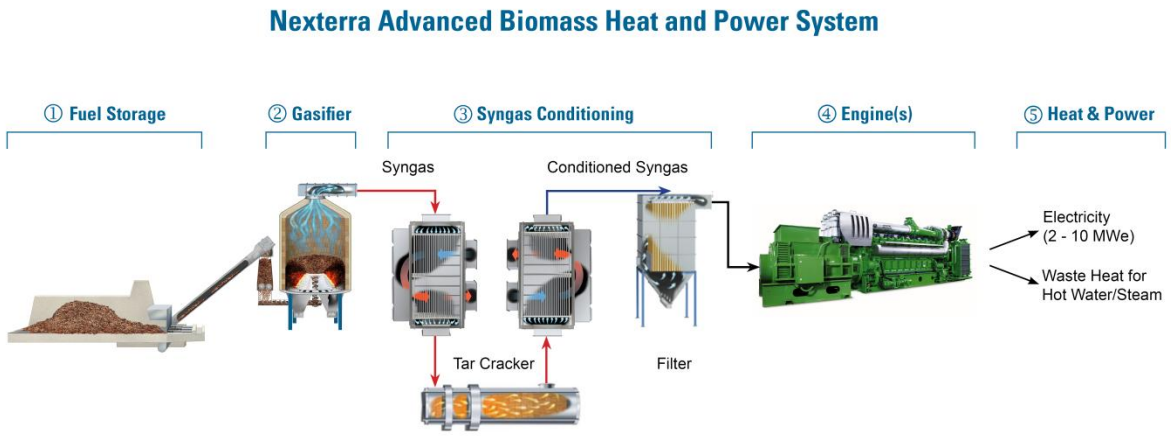


Figure 3: Nexterra advanced biomass heat and power system [13**]**

The Nexterra biomass boiler is the first of its kind in North America. It has two operating modes: the thermal mode, which generates heat for the UBC campus; and the co-generation mode (heat and electric power), which generates heat as well as electricity. After this system starts full operation, it will provide in co-generation mode 12% of the campus' average heat and up to 4.5% of the peak electric power; and in thermal mode 25% of the campus heat [**14**].

The biomass fuel used by Nexterra mostly comes from urban wood waste. In addition, it is considered as carbon neutral [**15**] [**16**]. Introducing this technology to UBC will reduce GHG emission by up to 9,000 tons each year [**14**].

2.1.2 Natural Gas Boilers

The primary boilers used in the old steam plant are natural gas based, and were built between 1922 and 1966. These aged boilers will be decommissioned in 2017 [**10**]. In this thesis, electrical boilers are also considered to replace natural gas boilers, so that the GHG emission can be reduced to a minimum. The electric boiler use off-peak electricity. In addition heat pumps that use waste heat from triumf's particle accelerator are considered.

2.1.3 Electric Boilers

UBC Vancouver campus is powered by two electric substations: The North Substation (UNY) and The South Substation (UNS) [**17**]. These substations are supplied by BC Hydro through two power transmission circuits: the north transmission line, which has a capacity of 62 MVA, and the south transmission line, which has the capacity of 42 MVA. The transmission system is designed so that either of these two lines can supply the entire load of the North substation, in case the other one is out of service [**18**]. Figure 4 shows the connections between the two transmission lines and the two substations.

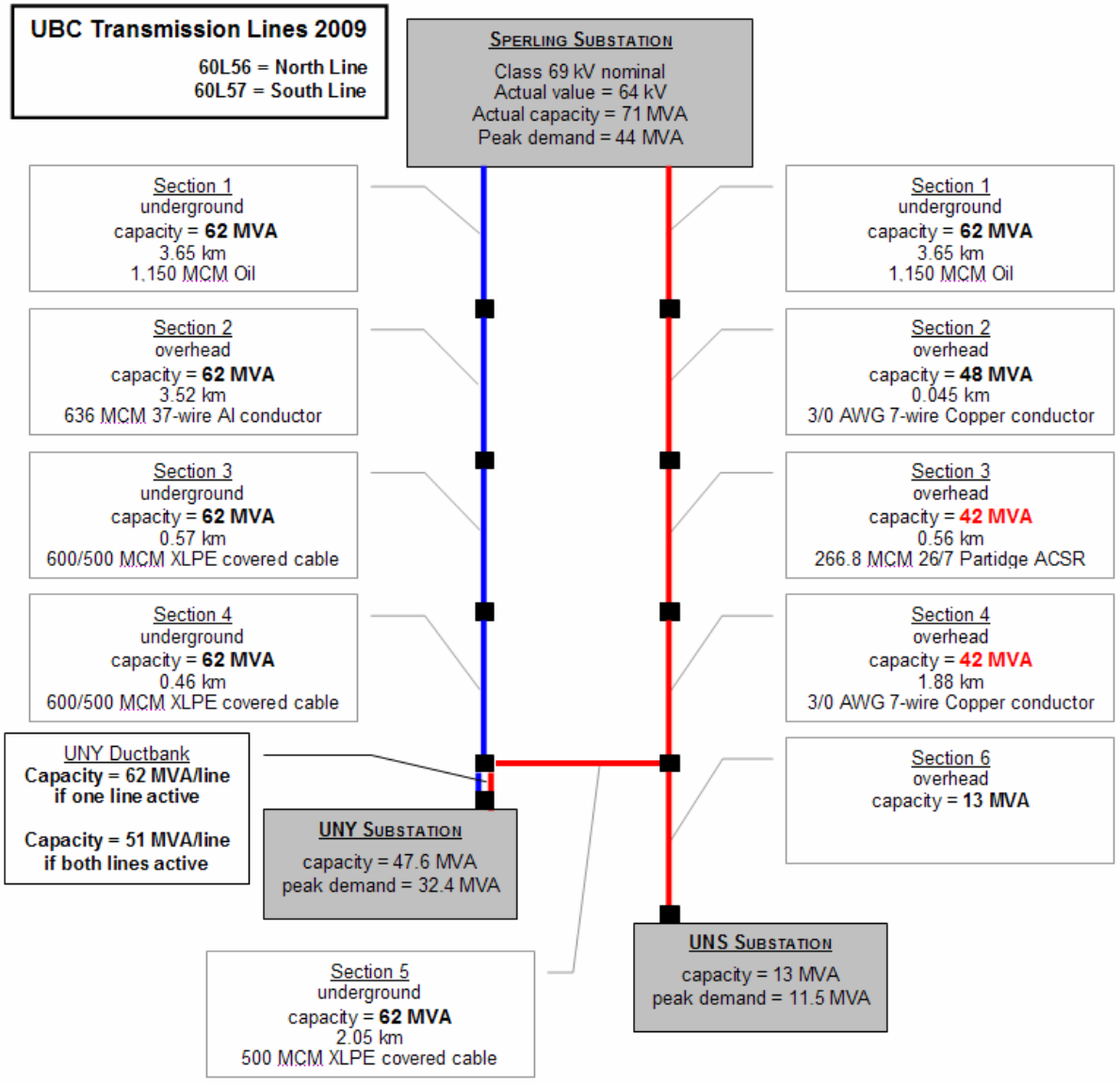


Figure 4: BC Hydro transmission line to UBC campus [18]

The idea of using electric boilers is to take advantage of the off-peak electricity available to the UBC campus. Since electricity in BC is about 90% of hydro, it is considered a clean source of energy. In Figure 5, the blue section is the heat demand that can be supplied by off-peak electricity, the red section is the remaining heat demand supplied by natural gas, and the green

section is the extra electricity usage. The graph shows that the off-peak electricity can supply most of the UBC campus heat demand, and sometimes still has a large portion of unused capacity.

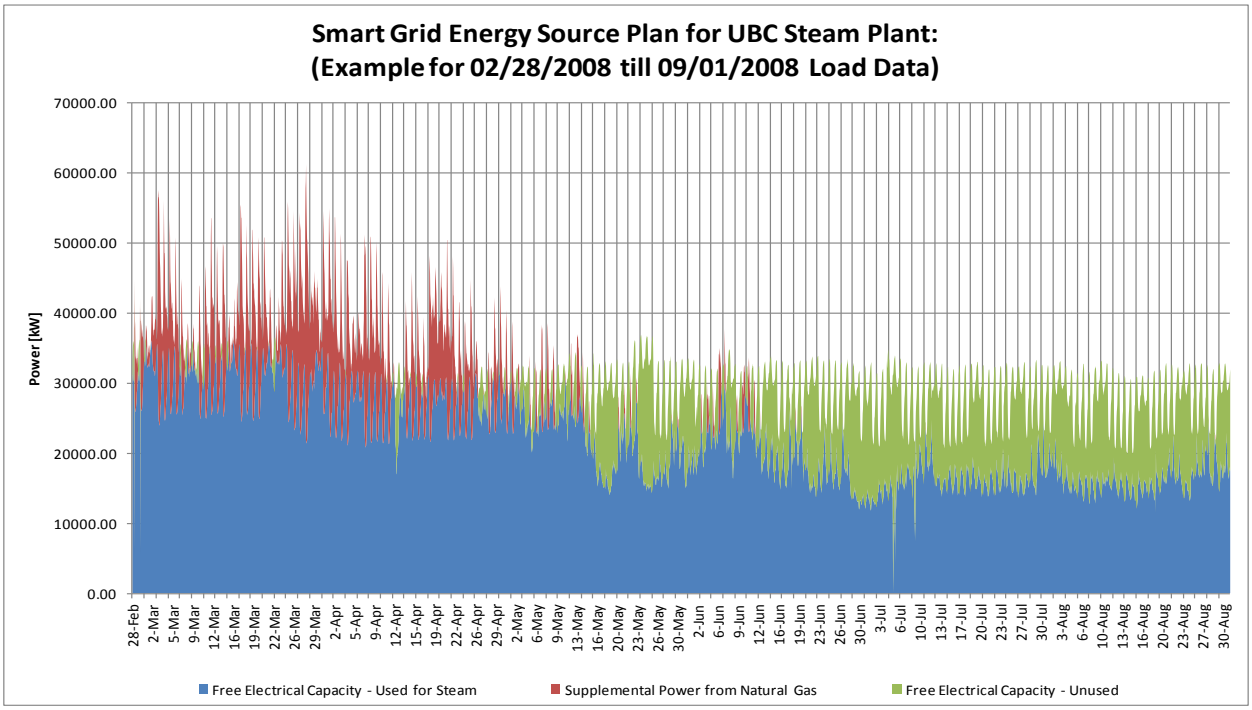


Figure 5: Natural gas and off-peak electricity used for campus heat production as well as extra electricity usage [19**]**

2.1.4 Heat Pumps

Located on UBC campus, TRIUMF is a world-class subatomic physics research laboratory [**20**]. Within its facilities, there is the world's largest cyclotron and it requires 18,500 Amps current to power its magnets [**21**]. The waste heat from the cyclotron could be re-used to heat hot-water for campus heating. One way of cycling the heat is through heat pumps.

A Heat pump extracts heat from one source and transfers it to another place. Common examples of heat pumps are refrigerators and air conditioners [**22**].

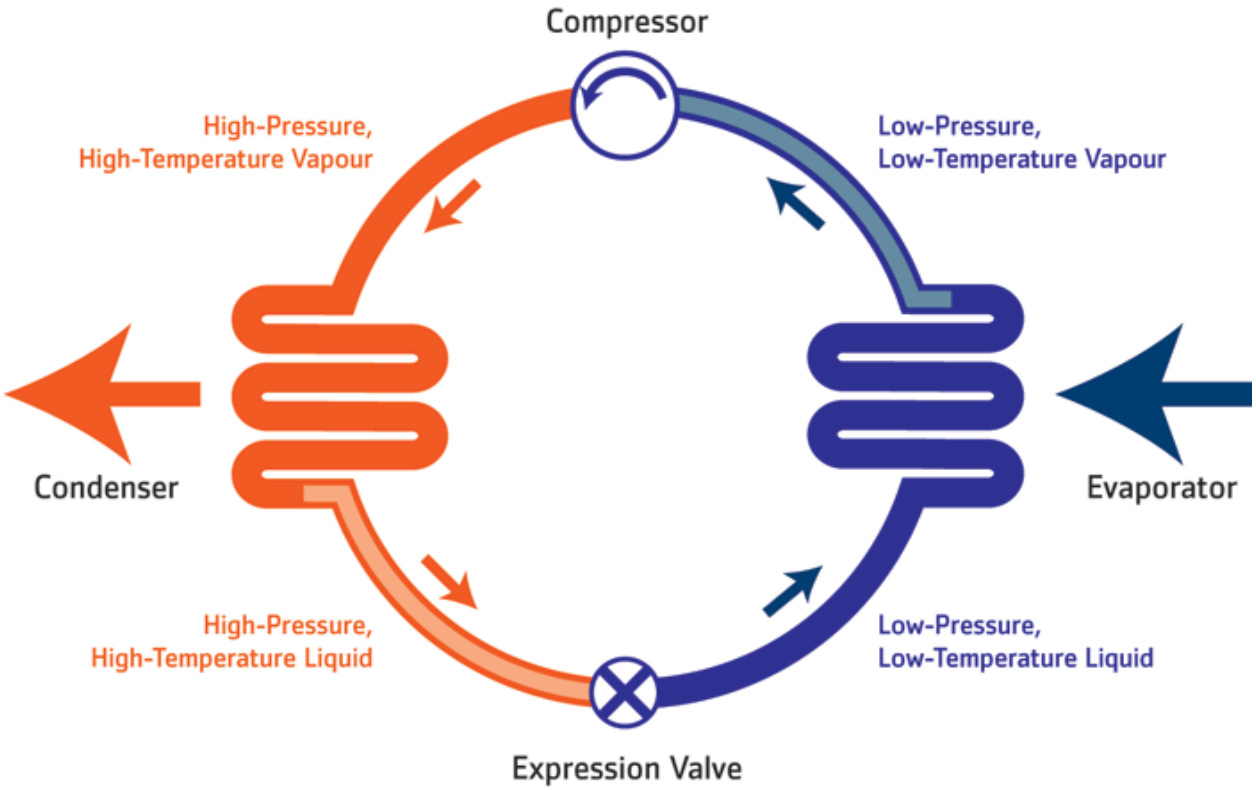


Figure 6: Basic heat pump principle [23**]**

The efficiency of a heat pump is measured in term of coefficient of performance (COP). It is calculated as:

$$Coefficient\ of\ performance\ (COP) = \frac{Energy\ output\ of\ the\ heat\ pump}{Electrical\ energy\ needed\ to\ run\ the\ heat\ pump}$$

The higher the COP, the more efficient the heat pump is [**22**].

2.2 UBC's Living Lab I2Sim Model Overview

The primary objective of UBC's Living Lab is to minimize Green House Gas (GHG) emissions and optimize the financial outcome. As one of the leading modeling tools for multi-system interdependencies, I2Sim is a very good tool for UBC living lab modeling. The existing toolbox is sufficient for many applications, but does not have a financial layer. The new financial production cell (FPC) developed in this thesis is intended to bridge this gap. The details of the financial production cell are discussed in section 2.6 of this chapter.

The living lab I2Sim model discussed in this thesis focuses on the campus heating system. In this case, it is the planned hot-water heating system. Currently, this system is planned to use natural gas and biomass as its main energy sources. Electricity and heat pumps from UBC Triumf particle accelerator are options that could be employed as potential energy sources. In this simulation model, all of these energy sources were integrated together as shown in Figure 7. The purpose of this is to find an optimal solution and use it as a reference for future projects.

In the next several chapters, the key I2Sim components for the living lab model will be discussed in detail. They are:

- Selector
- Production Cell
- Financial User Interface
- Controller
- Financial Production Cell

Since the financial production cell is the centre focus of this thesis, it will be described explicitly in section 2.6.

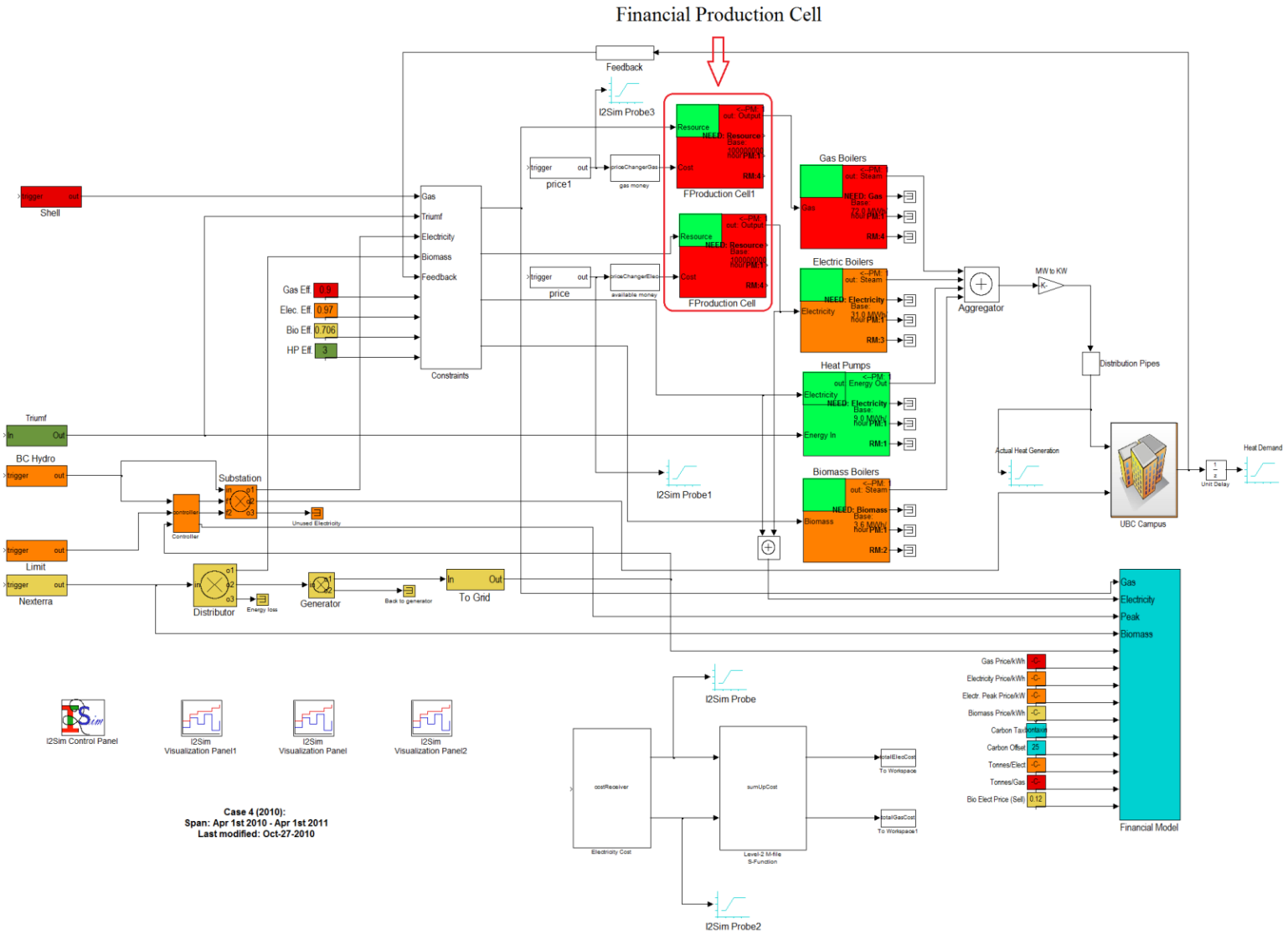


Figure 7: I2Sim living lab model overview

2.3 Selector

In order to reduce the GHG emission, the selector chooses the energy source that generates the least GHG. The flow chart shown in Figure 8 demonstrates the selection process within the selector component. Biomass is chosen first because it is considered as a base plant.

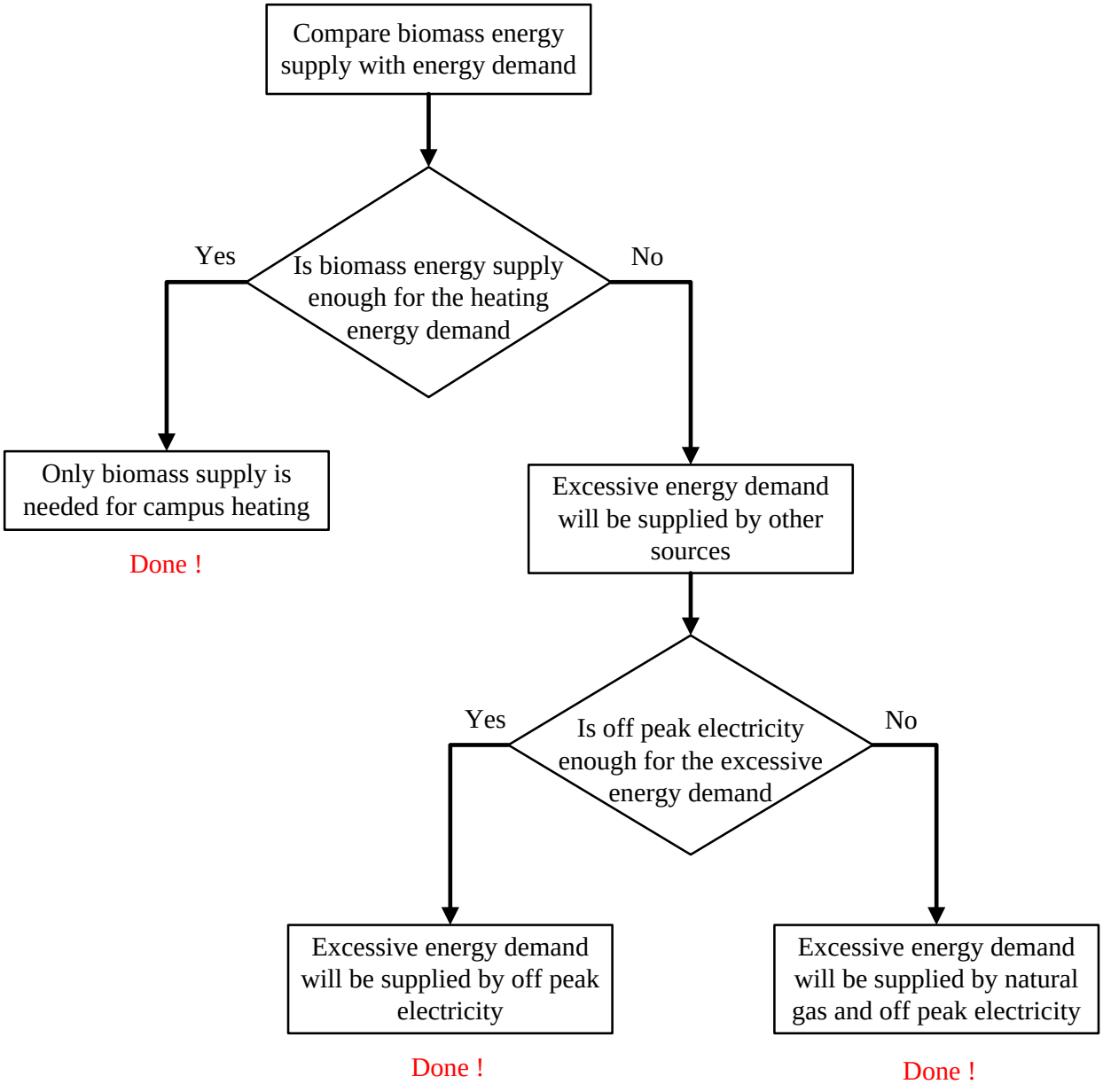


Figure 8: UBC living lab I2Sim model constraints flowchart

2.4 Controller

The Controller in the living lab model controls the electricity distribution between campus electrical usage and campus heat usage. The Controller has two main tasks:

- Find the peak electricity usage during each month.
- Control the distributor to send electricity to both the UBC campus existing electrical system and the proposed electrical boiler.

2.5 Production Cell

The name of production cell is based on the functionality of the cell, which is producing output tokens with input resources [**4**]. By design a production cell can have several inputs, but it can only have one output.

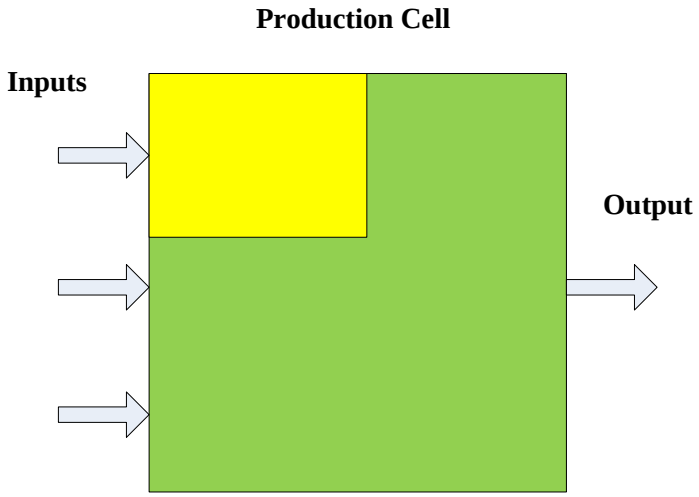


Figure 9: Production cell overview

Production cells can model many different facilities; one example is a steam plant. The steam plant needs multiple available resources to maintain its functionality. These resources will include natural gas for its gas boilers, electricity for its electrical boilers, and last but not least, water to generate the steam.

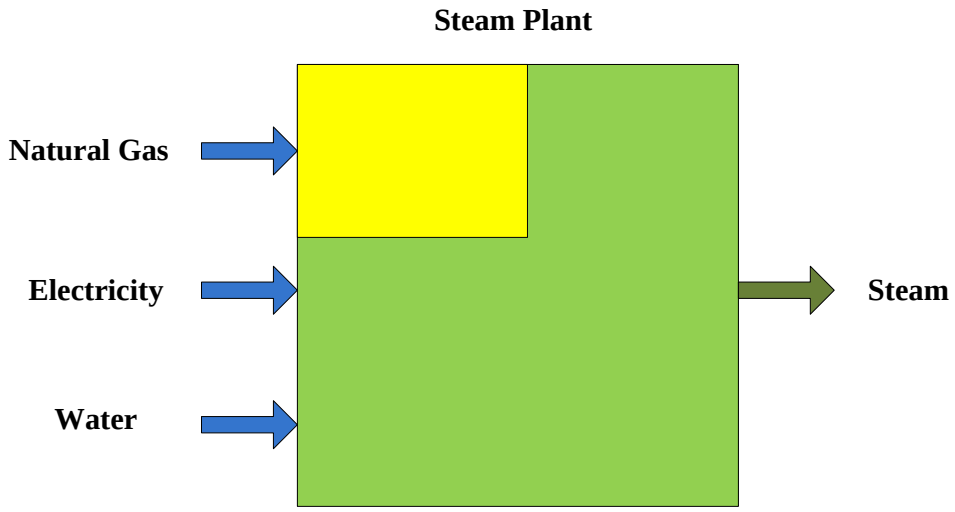
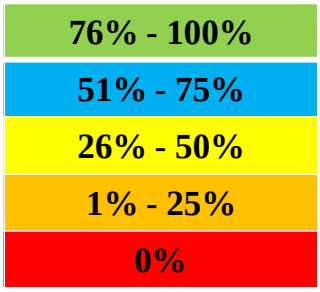


Figure 10: Production cell steam plant sample

2.5.1 Physical Mode (PM)

In I2Sim, a regular production cell has a physical mode (PM) and a resource mode (RM). The physical mode represents the physical condition of the entity it is modeling, and it also limits the maximum resource mode level [**5**]. For example, when the steam station is at its peak condition, everything is working perfectly; the steam output can reach its 100% capacity if all the resources are available. However, if the steam station is damaged and its physical condition is reduced to 75%, no matter how much resource it receives, the steam output cannot exceed more than 75% of its full capacity.

There are five different colour codes associates with five different physical mode conditions, which are shown in the Figure 11.



Physical Mode			
Output	Inputs		
Steam	Natural Gas	Electricity	Water
100%	100%	100%	100%
75%	75%	75%	75%
50%	50%	50%	50%
25%	25%	25%	25%
0%	0%	0%	0%

Physical Mode			
Output	Inputs		
Steam	Natural Gas	Electricity	Water
75%	75%	75%	75%
50%	50%	50%	50%
25%	25%	25%	25%
0%	0%	0%	0%

Physical Mode			
Output	Inputs		
Steam	Natural Gas	Electricity	Water
50%	50%	50%	50%
25%	25%	25%	25%
0%	0%	0%	0%

Physical Mode			
Output	Inputs		
Steam	Natural Gas	Electricity	Water
25%	25%	25%	25%
0%	0%	0%	0%

Physical Mode			
Output	Inputs		
Steam	Natural Gas	Electricity	Water
0%	0%	0%	0%

Figure 11: Production cell physical modes

2.5.2 Resource Mode (RM)

The resource mode (RM) indicates the availabilities of different resources. In each physical mode, the relationship of output and inputs, which also reflects the resource mode, is determined by the Human Readable Table (HRT). The HRT expresses the outputs as a set of non-linear functions: $y_{i1} = f_1(x_{i1}, x_{i2}, x_{i3}, \dots\dots, x_{in})$ in terms of inputs [**24**]. The Human Readable Table (HRT) contains multiple rows and columns. The rows represent different resource conditions,

and the columns represent output and inputs. The first column is always the output and the rest are inputs.

In order to produce the required output, the inputs have to meet the minimal requirement. For example, as shown in the following table, if 100 units of steam need to be produced, the cell needs 5000 units of natural gas, 3000 units of electricity, and 6000 units of water. However, if only 5000 units of natural gas, 800 units of electricity, and 5000 units of water are provided, only 25 units of steam will be produced, because the electricity, in this case, is the limiting factor.

	Steam	Natural Gas	Electricity	Water
	100	5000	3000	6000
	75	3500	2500	5000
	50	2500	1500	3000
Output Row	25	2000	800	1500
	3	200	50	150

Figure 12: Production cell HRT

The resource mode also employs the principle of colour coding, Figure 13 shows that the status of both the physical mode and resource mode are visible to the user. They are changing in real time simulation based on the cell condition. They give user a visual feedback of what is happening during the simulation.

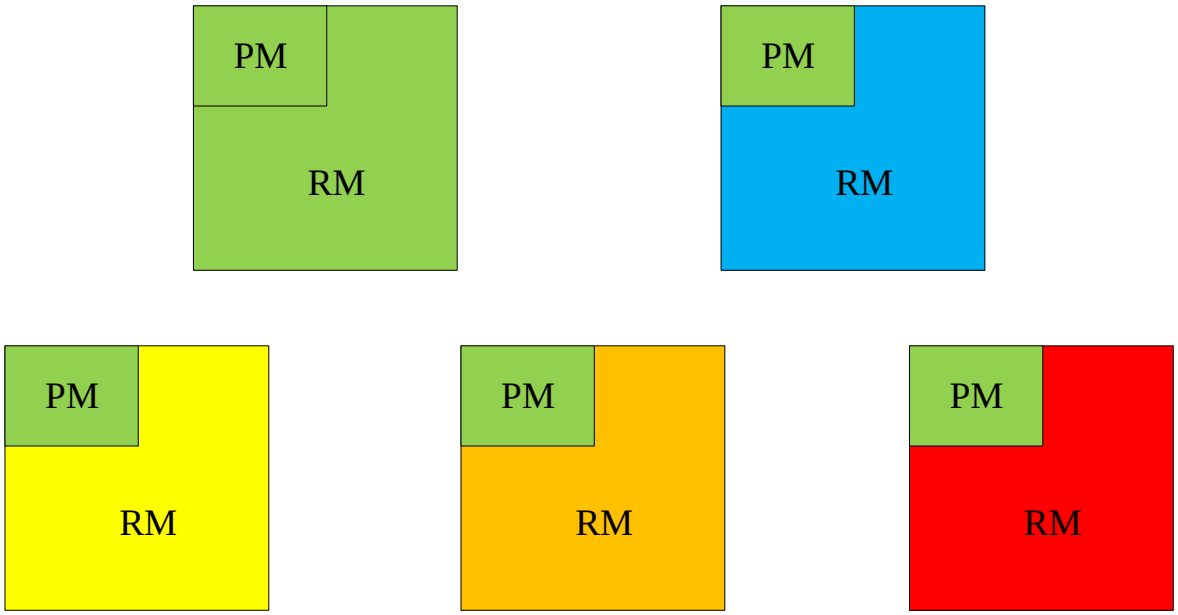


Figure 13: Production cell resource mode colour code

2.6 Financial Production Cell

The Financial Production Cell (FPC) is derived from a regular production cell, so they share the same basic ontology and design philosophy. Both types of production cells have a physical mode and resource mode. However, depending on the model itself, the regular production cell usually has two or more inputs. The financial production cell, on the other hand, only has two inputs: the available resource, and the available money that is allocated to the cell.

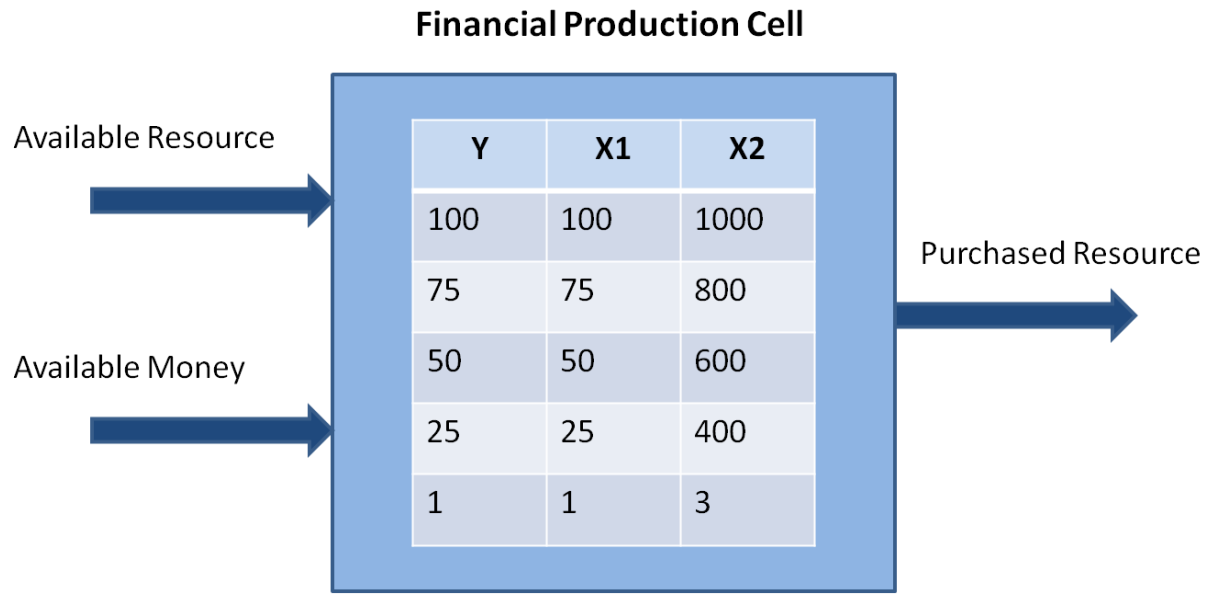


Figure 14: Financial production cell overview

The financial production cell focuses on the financial aspect of the model. It simulates the commodity trading process of resources, and this process is more of a virtual concept than a physical infrastructure. For instance, the purchasing of electricity is settled with a contract agreement. This process does not involve a physical building. Therefore, the physical mode (PM)

of the financial production cell does not represent the physical condition; neither will it suffer physical damage.

The resource mode of the financial production cell governs the relationship between demand and supply. The detail of the PM and RM regarding a financial production cell is discussed in the next section.

2.6.1 Physical Mode (PM) of the Financial Production Cell

The physical mode (PM) in the financial production cell has a similar look as the regular production cell. However, it represents a different concept. The differences between physical modes in the financial production cell and regular production cells are:

- The PM in the regular production cell represents the physical condition of the infrastructure, while the PM in the financial production cell indicates the cost of the resource.
- The PM in the regular production cell limits the maximum RM level, but the PM in the financial production cell always stays at the maximum RM level.

The idea of employing the concept of physical mode (PM) is to apply the same ontology through the entire I2Sim toolbox, which minimizes the complexity of the toolbox. On a higher level point of view, this approach greatly simplifies the complexity of the real-world case into manageable simulated system [**1**].

The physical mode in the financial production cells provides a frame to hold resource cost tables, which is the HRT. When the physical mode changes, the resource cost table is replaced by a new one. In real life trading cases, the prices of different resources change all the time. Therefore, the resource cost table needs to be updated continuously in real time during the simulation. In the

model, this is by accessing the HRT directly during the simulation. In chapter 3.1, this feature was fully tested and verified.

2.6.2 Resource Mode (RM) of the Financial Production Cell

The core structure of the resource mode is the human readable table (HRT); it defines the purchasing behavior of the financial production cell. There are three columns in the HRT. The entities in the each column are:

- First column (Y): purchased resource based on the available resource and money needed.
- Second column (X_1): the available resource at a particular price.
- Third column (X_2): the corresponding cost for the available resource (X_1) on the same row.

On each row, the relationship between available resource (X_1) and cost (X_2) is a non-linear curve. The reason is that the more resource is purchased, the larger the cost will be. However, per unit price wise, the larger the amount of resource is purchased, the cheaper the per-unit price will be. Therefore, the total cost is increasing with the growth of purchasing, but slope of the curve is decreasing. This is consistent with real world cases. The example in Table 1 is plotted in Figure 15 and Figure 16. It illustrates that this non-linear relationship has a logarithmic characteristic.

Table 1: Relationship among resource, total cost and per unit price

Amount of Resource	Total Cost	Per Unit Price
100	20000	200
50	15000	300
20	10000	500
10	6000	600
5	4000	800
1	1000	1000

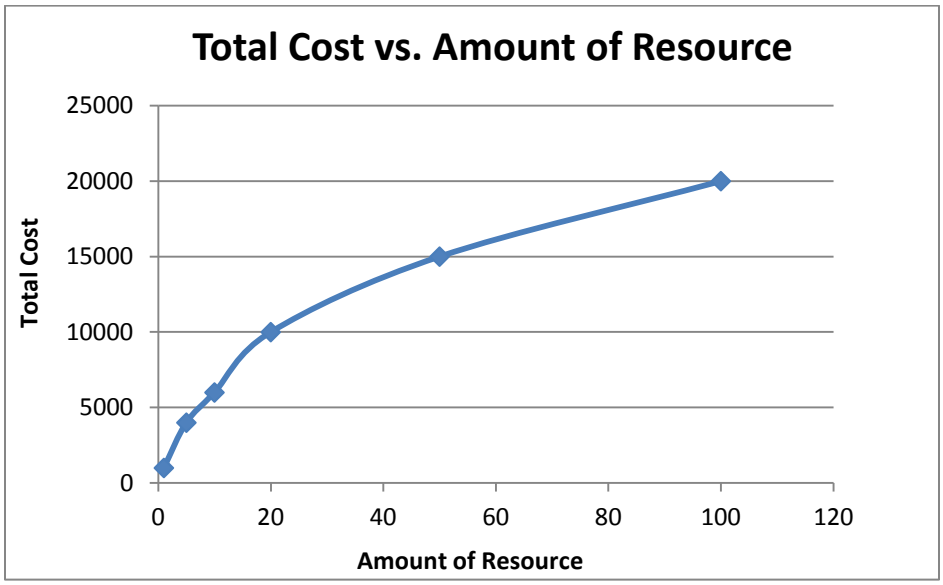


Figure 15: The relationship between total cost and amount of resource

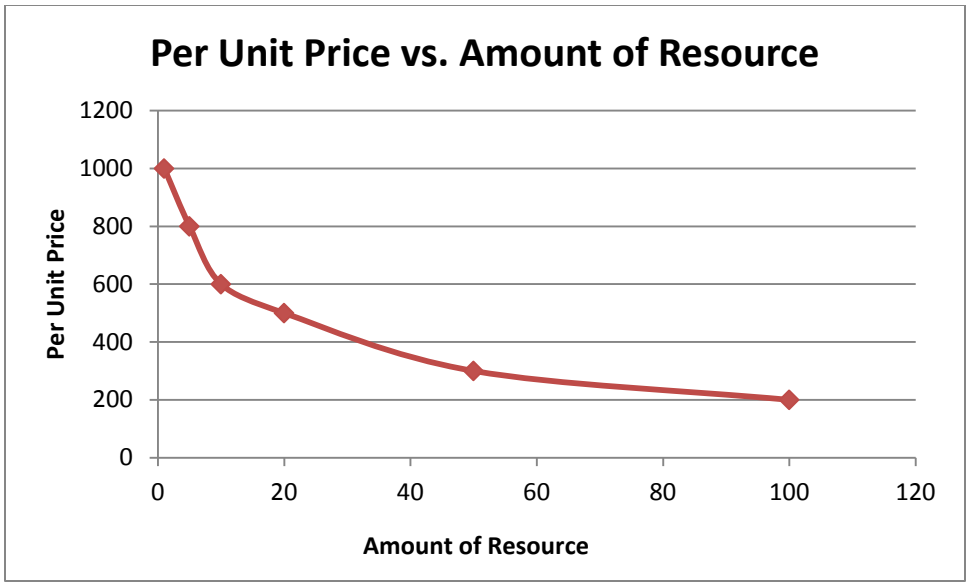


Figure 16: The relationship between per unit price and amount of resource

When the two inputs (available resource and available money) are fed into the financial production cell, the HRT automatically launches a search sequence. This sequence starts searching from the X_1 column first and then moves to the X_2 column. The purpose of this search is to locate and identify which input is the limiting factor. The flowchart in Figure 17 shows how the search is performed. The allocated money in the flow chart indicates the available money that is allocated to the financial production cell.

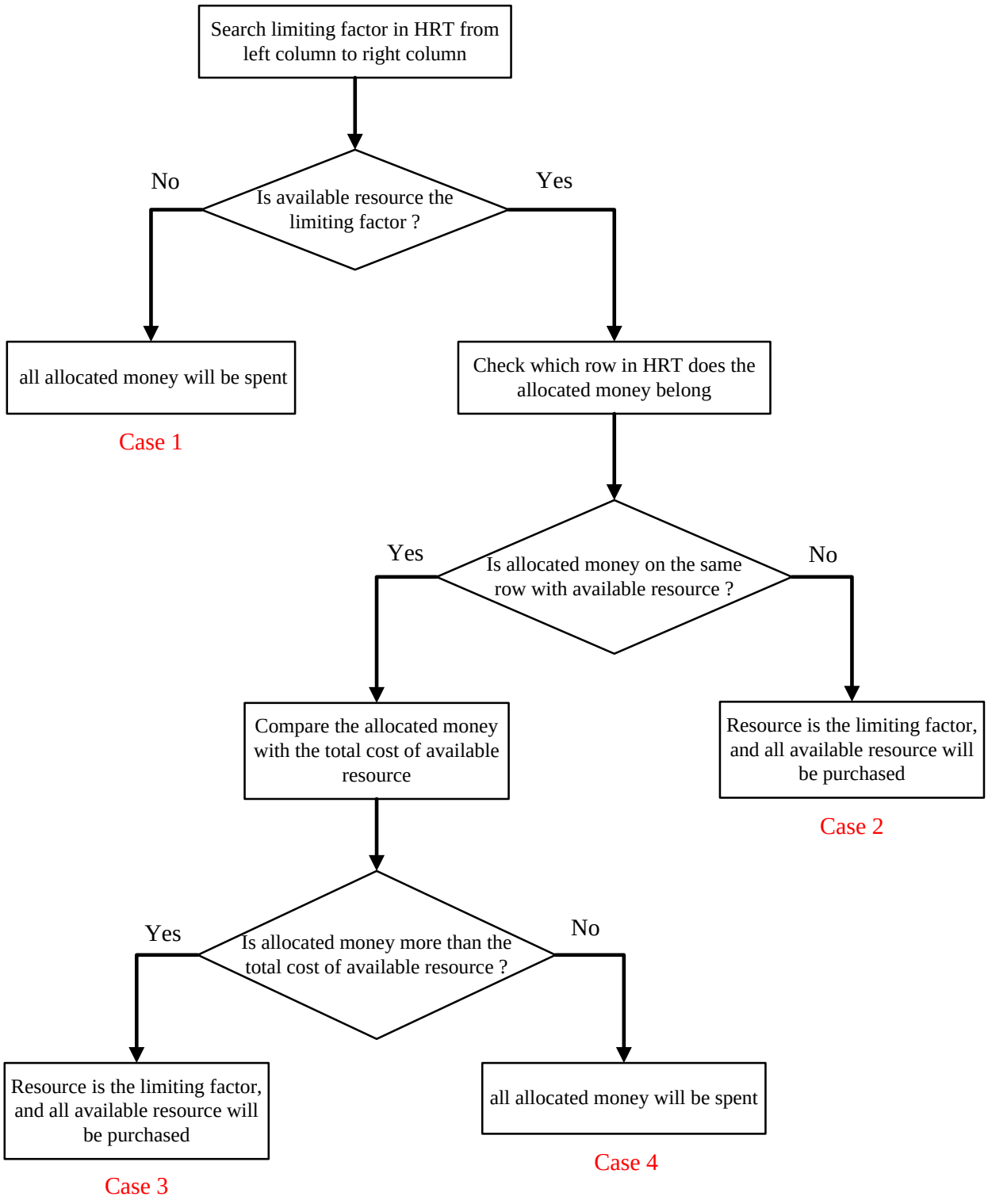


Figure 17: Financial production cell HRT flow chart

In the flowchart, among all possible combinations, four general cases can be defined. In each case, the amount of purchased resource and spent money are calculated as follows:

- Case 1: Available resource and available money belong to different rows in the HRT, and available money is the limiting factor.

$$\frac{X_1}{Purchased\ Resource} = \frac{X_2}{Available\ Money}$$

$$\boldsymbol{Purchased\ Resource} = \frac{(X_1) \times (Available\ Money)}{X_2}$$

$\boldsymbol{Spent\ Money} = Available\ Money$ (all allocated money will be spent)

- Case 2: Available resource and available money belong to different rows in the HRT, and the availability of the resource is the limiting factor.

$$\frac{X_1}{Available\ Resource} = \frac{X_2}{Spent\ Money}$$

$$\boldsymbol{Spent\ Money} = \frac{(Available\ Resource) \times (X_2)}{X_1}$$

$\boldsymbol{Purchased\ Resource} = Available\ Resource$ (all the available resource will be purchased)

- Case 3: Available resource and available money belong to the same row in the HRT, and the availability of the resource is the limiting factor.

$$\frac{X_1}{Available\ Resource} = \frac{X_2}{Spent\ Money}$$

$$\boldsymbol{Spent\ Money} = \frac{(Available\ Resource) \times (X_2)}{X_1}$$

$\boldsymbol{Purchased\ Resource} = Available\ Resource$ (all the available resource will be purchased)

- Case 4: Available resource and available money belong to the same row in the HRT, and the available money is the limiting factor.

$$\frac{X_1}{Purchased\ Resource} = \frac{X_2}{Available\ Money}$$

$$\boldsymbol{Purchased\ Resource} = \frac{(X_1) \times (Available\ Money)}{X_2}$$

$\boldsymbol{Spent\ Money} = Available\ Money$ (all allocated money will be spent)

The output of the financial production cell is the purchased resource. In a regular production cell, once the limiting factor is identified, no matter the value of the input, the output of the

production cell will be determined by the output row in HRT, which is illustrated in Figure 12. However, in the financial production cell, the outputs are calculated instead of being controlled by the HRT row.

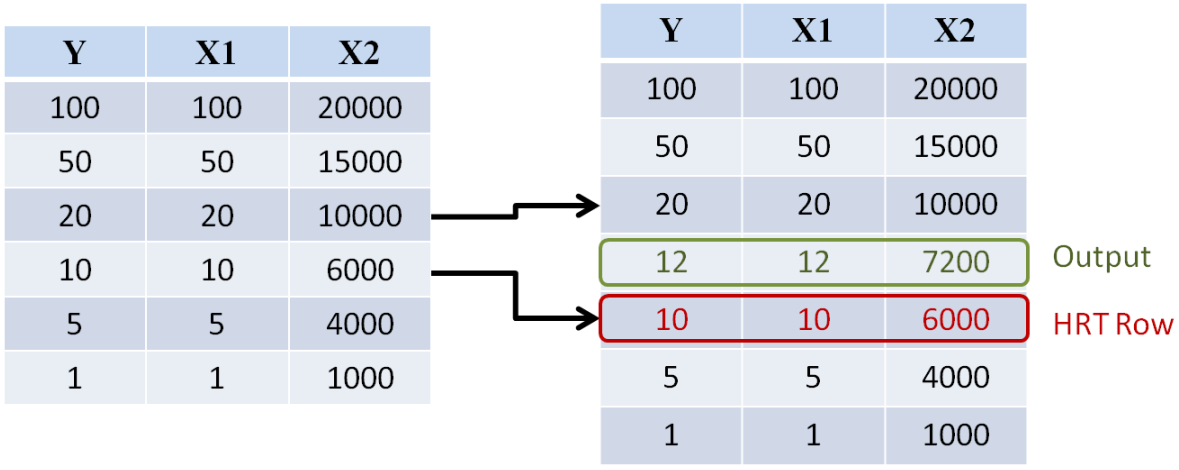


Figure 18: Financial production cell HRT

Figure 18 shows a sample case of how the financial production cell HRT produces output when the limiting factor is found and it is above the HRT row (the red row). In this case, the output is calculated based on the formula in one of the four previous cases.

Although, according to I2Sim ontology, there is only one output from a financial production cell, which is the purchased resource, the amount of spent money is also needed by many other applications. For instance, the financial user interface in next chapter will need this information to calculate the Net Present Value (NPV). One way to obtain this value is to use a Matlab command get_param. It provides a direct access to this parameter in computer memory. The behavior of FPC's HRT was tested in chapter 3.1.

2.7 Financial User Interface

I2Sim and Simulink are powerful mathematical tools to create and simulate complex real time models, but not all users are experts in programming. Therefore, a clear user interface is required. During the period of creating such interface, many similar programs or toolboxes were studied; for example, the EnergyPLAN toolbox [**25**]. The financial user interface provides the user a friendly environment to post-process the financial data generated from the model.

The user interface block implemented is a Matlab GUI function. It consists of two parts; the first part is a ".fig" picture file. This picture file defines layout, frame work and function buttons of the interface. The second part is ".m" Matlab program file. It implements the functions of all the buttons and displays in the interface.

The annual cash flow and the NPV user interfaces calculate and display the annual cash flow of the model according to the life-time of the project. Then it computes the NPV based on an appropriate interest rate. Figure 19 provides a look of the interface.

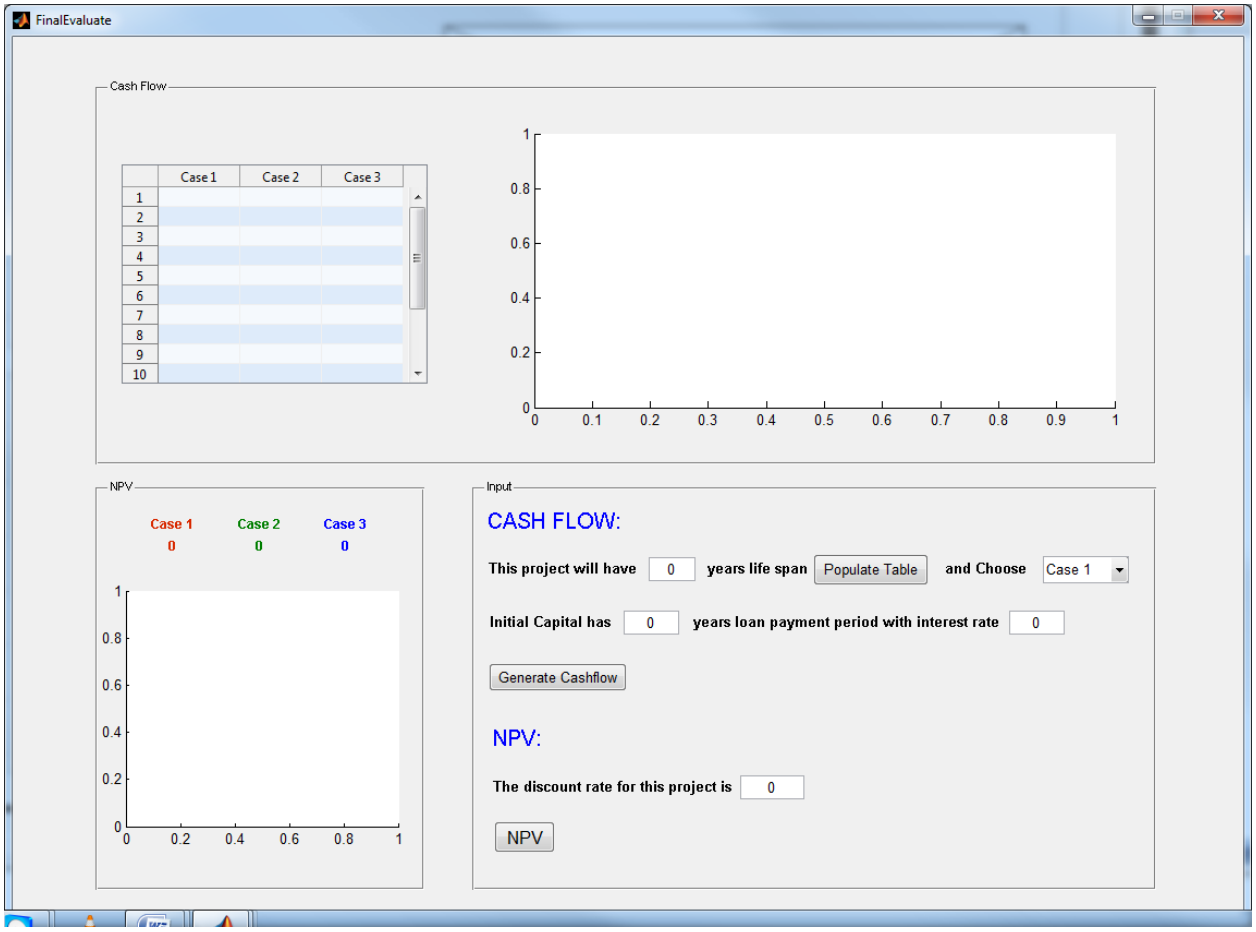


Figure 19: Annual cash flow and NPV user interface

The financial interface in Figure 19 contains two major parts: Cash Flow calculation and Net Present Value (NPV) calculation.

In the cash flow part, it obtains the annual cash flow information from the living lab model and displays this information on the cash flow table. The user has full control of the table content, which can be added, deleted or modified.

The cash flow of the project in each year consists of:

- Annual annuity of the initial capital cost

- Annual total maintenance cost
- Annual total operation cost
- Annual total fuel cost
- Annual revenue

In the I2Sim model, each production cell represents a physical structure (E.g. a hospital, an electrical substation or a water station). The initial capital cost, annual total maintenance cost and annual total operation cost of that structure are directly input by the user as attributes.

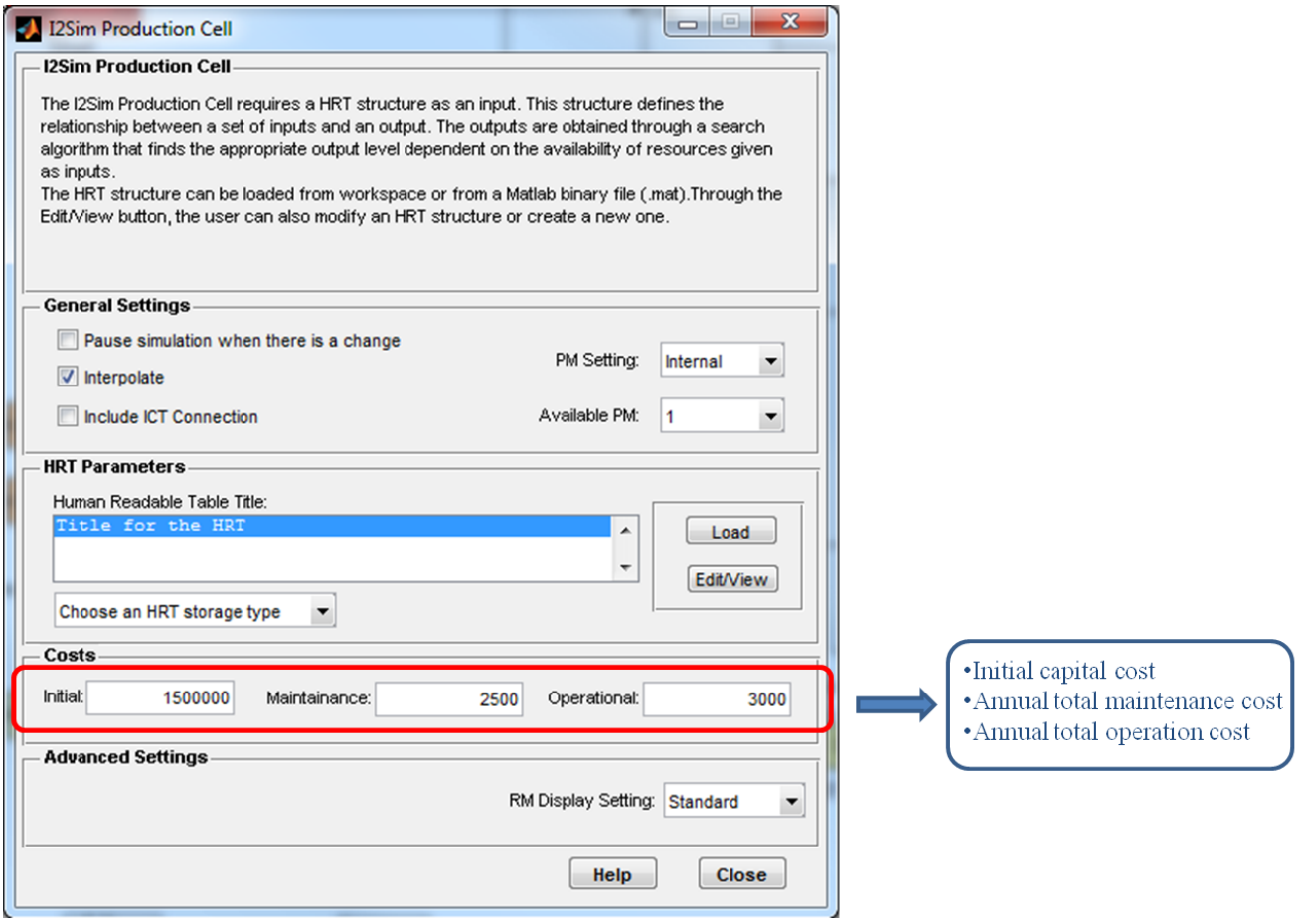


Figure 20: Initial condition of the financial production cell

When a facility is built, the initial capital cost is paid by a loan broken down into several annual annuity payments. The loan payback period and the loan interest rate could be different from the life of the facility and from the inflation interest rate; therefore, they have to be specified. The annual annuity of the initial capital cost is calculated as [**26**]:

$$Annuity = \frac{Initial\ Capital\ Cost}{[\frac{1-\left(1+i_p\right)^{-n}}{i_p}]}$$

Where i_p is the interest rate for the payment period, and n is the number of payment periods.

The annual cash flow is then calculated as:

$$cash\ flow = -(annuity + maintenance\ cost + operation\ cost) + cost\ reduction$$

In step two, the NPV (net present value) is calculated by an additional function calculateNPV.m and plotted on the bottom left of the interface.

In this case, the NPV (Net Present Value) of the project is calculated as [**26**]:

$$NPV = CF_0 + \frac{CF_1}{(1+r)^1} + \frac{CF_2}{(1+r)^2} + \cdots + \frac{CF_n}{(1+r)^n}$$

Where CF_j is the cash flow at time j, and n is the lifetime of the investment.

The "calculateNPV.m" file is an additional function added onto the financial user interface; its content can be easily modified at a later time, which provides the end user great flexibility to apply a more sophisticated method or function in the future.

3 Simulation Results and Discussion

This chapter focuses on the application of the financial production cell (FPC) and the simulation of various living lab models. A simple test case is first conducted on a single FPC alone, so that its functionality and features can be verified. Later, three living lab models are simulated and tested.

3.1 Financial Production Cell Testing

In this section, two main features were examined: the HRT behavior and the capability of updating HRTs during real time simulation. The first test model is shown in Figure 21 and includes one financial production cell, two inputs and one output probe. The purpose of this testing is to verify that the HRT's actual output matches the expected (calculated by hand) outputs.

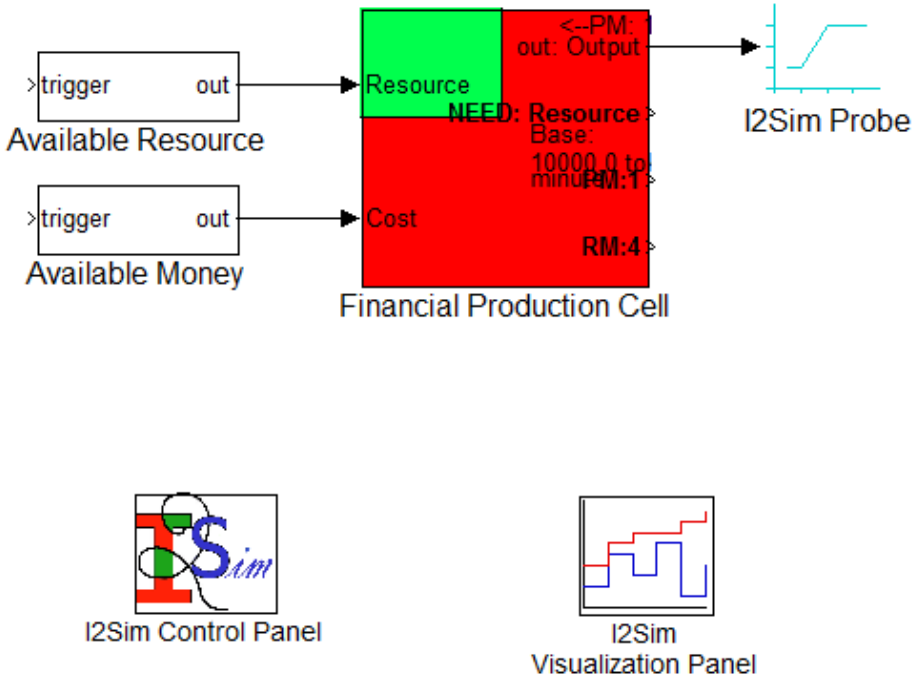


Figure 21: Financial production cell HRT behavior testing model

The parameters for the test HRT are assigned as in Table 2. They are chosen to simplify the testing.

Table 2: Testing HRT

Output	Resource	Cost
10000	10000	10000
1000	1000	2000
100	100	300
10	10	40
1	1	5

The test data in Table 3 covers all four different cases mentioned in section 2.6.2. The expected output column is highlighted in green.

Table 3: Financial production cell testing table

Test Data				
	Input Data		Output Data	
#	Available Resource	Available Money	Purchased Resource	Money Spent
1	5	1000	5	25
2	100	100	25	100
3	200	5000	200	600
4	5000	200	50	200
5	20000	1000	333	1000
6	300	100000	300	900
7	100	10000	100	300
8	1000	25	5	25
9	900	1900	633	1900
10	90	1000	90	360
11	1200	8000	1200	2400
12	600	5000	600	1800
13	50	1000	50	200

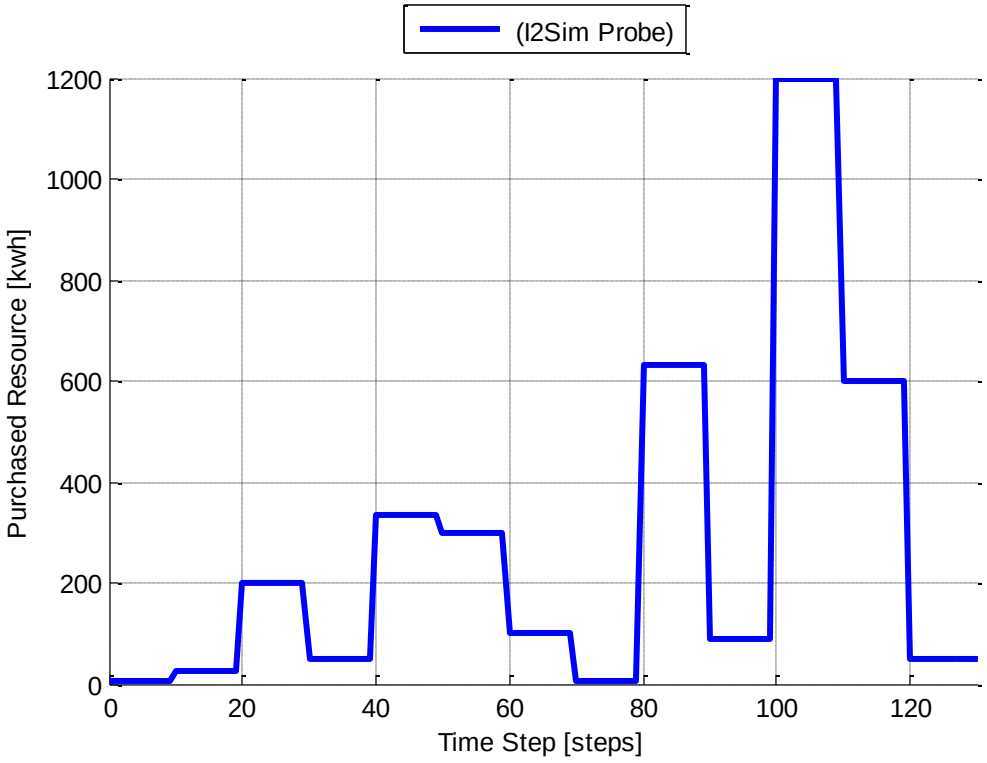


Figure 22: Financial production cell behavior test results

Each test case listed in Table 3 runs 10 steps during the simulation. In the end, the test results are displayed in Figure 22 and are consistent with the expected outputs listed in Table 3. The second test model is shown in Figure 23. The objective of this simulation is to prove that the HRT can be updated in real time during the simulation. The inputs of the simulation are shown in Figure 24; both available resource and available money were changed through time. The expected output in Figure 25 is the product of the resource and its corresponding price at the time.

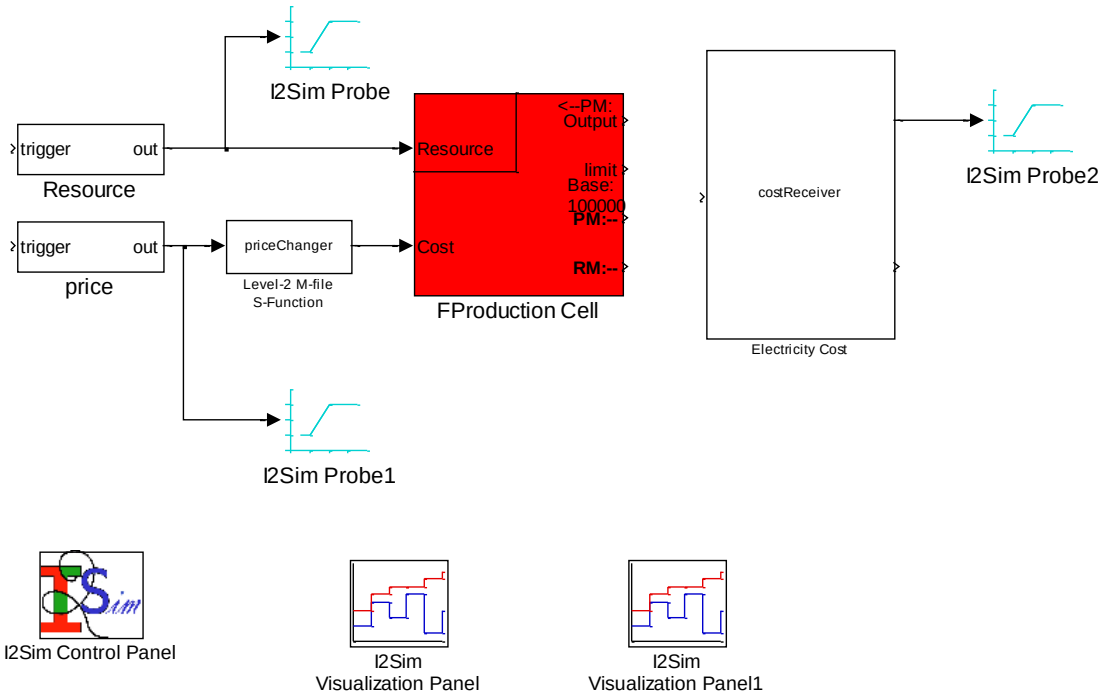


Figure 23. Financial production cell HRT dynamic update test model

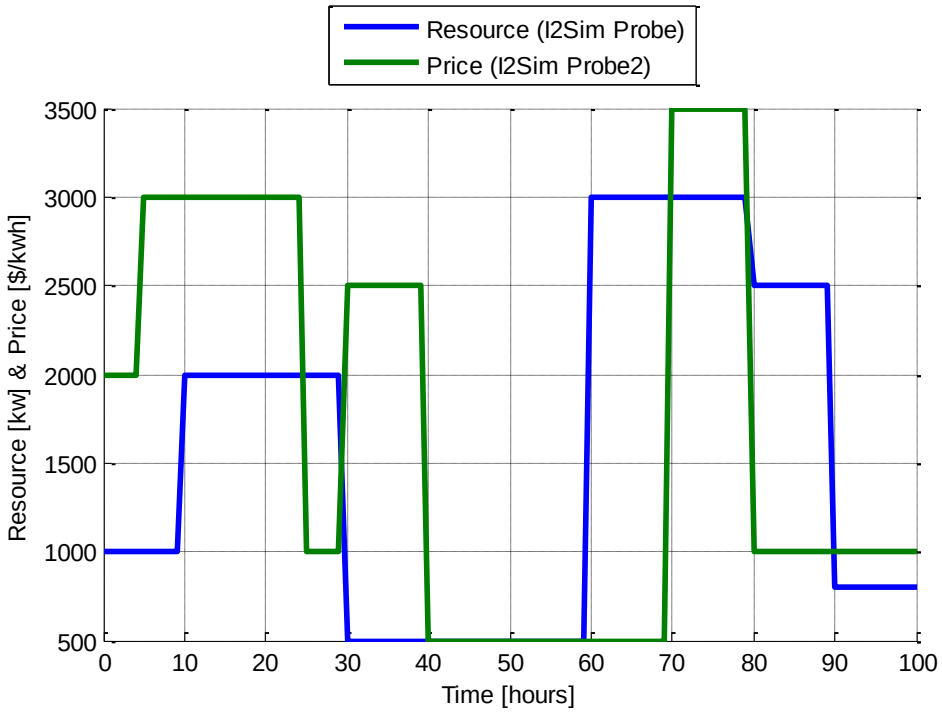


Figure 24: Resource vs. price

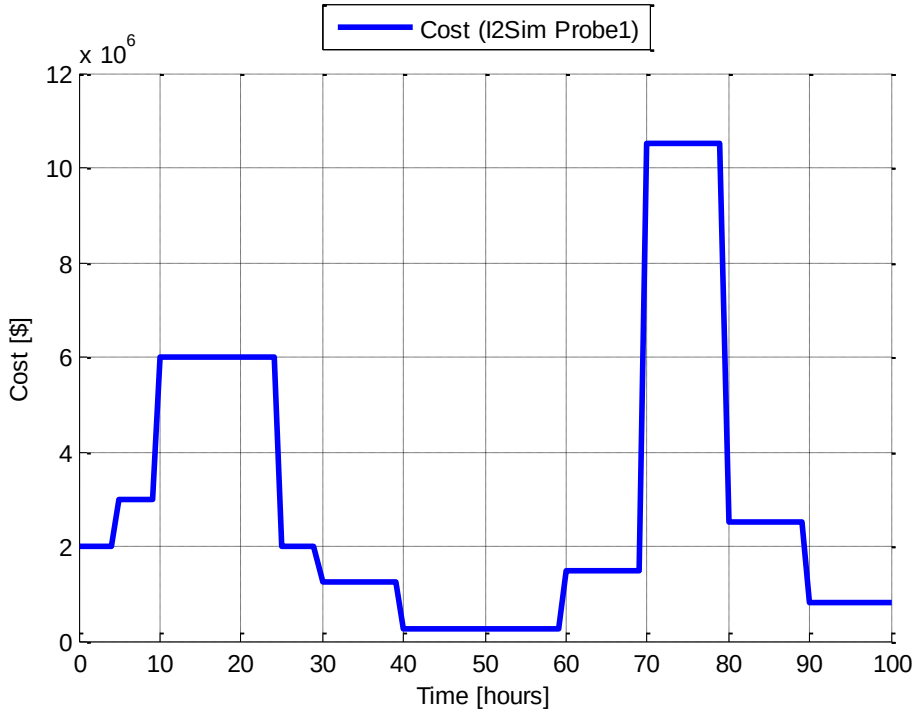


Figure 25: Expected resource cost

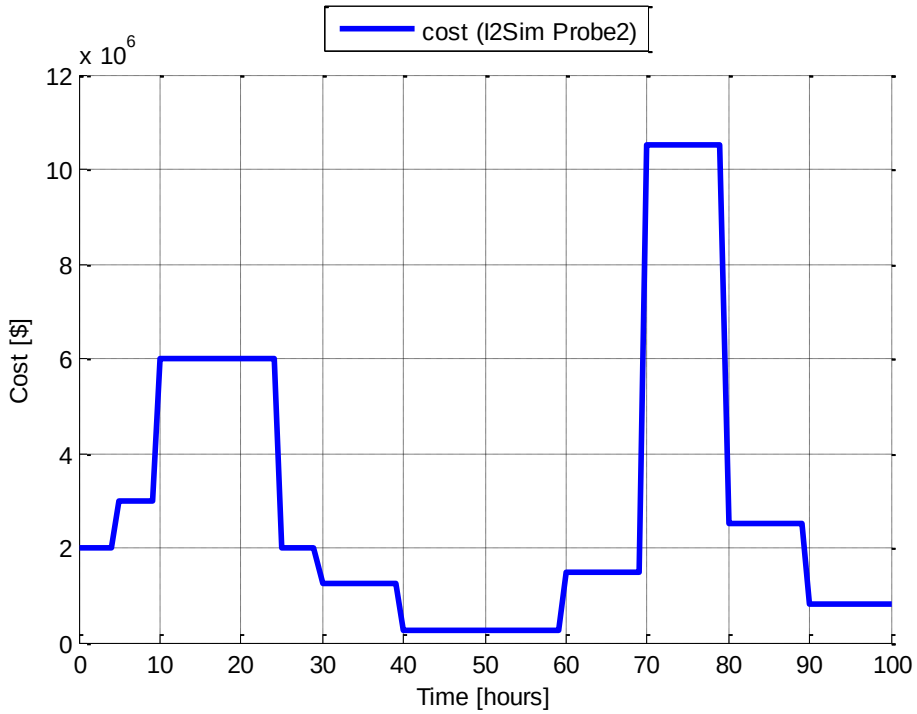


Figure 26: Resource cost from FPC test simulation

It can be observed that the simulation results match perfectly the expected results, which verifies the second major feature of the FPC: the capability of updating the HRT in real time during the simulation.

3.2 Living Lab Model Simulation

The data used for the UBC living lab simulation was UBC's electricity yearly usage data and UBC's steam plant yearly energy consumption data. The electricity usage data is from May 7th 2007 to January 1st 2010; the steam data is from April 20th 2005 to August 15th 2007, and from February 28th 2008 to September 1st 2008. In order to set up a base year for testing, a complete year's data is required. However, the overlapping periods of two sets of data are from May 7th 2007 to August 15 2007 and from February 28th 2008 to September 1st 2008, and neither of them is sufficient for a full base year.

To complete a full base year, it was observed that the average steam consumption during years 2007 and 2008 are similar. Taking advantage of this condition, part of the year 2007 steam consumption data was extrapolated and then filled into 2008 gap, with the remaining data from 2008, so that a complete year of steam demand data was obtained. The procedure of this setup is shown in Figure 27.

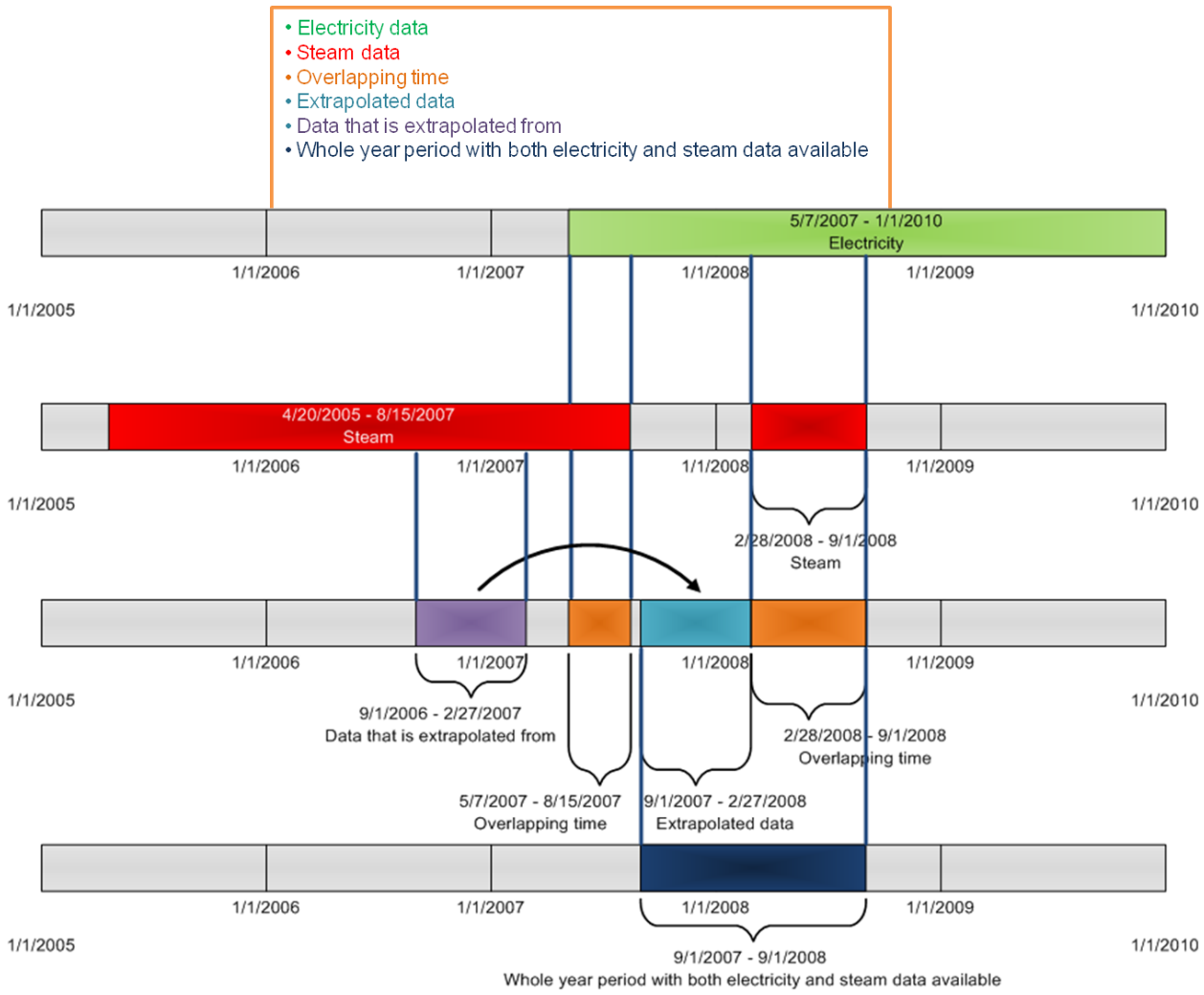


Figure 27: Obtaining a base year of data from the real data

The derived year 2008 data serves as a base year; the later years' data are derived from this data set by applying a projected growth factor.

$$Projected\ Growth\ Factor = \frac{Demand\ in\ 2009}{Demand\ in\ 2008} = 1.09$$

Then in the following years, the projected heat demand is calculated as:

$$Projected\ Heat\ Demand\ in\ 2009 = (Demand\ in\ 2008) \times (1.09)$$

$$Projected\ Heat\ Demand\ in\ 2010 = (Demand\ in\ 2008) \times (1.09)^2$$

$$Projected\ Heat\ Demand\ in\ 2011 = (Demand\ in\ 2008) \times (1.09)^3$$

$$Projected\ Heat\ Demand\ in\ 2012 = (Demand\ in\ 2008) \times (1.09)^4$$

•

•

•

And so on

3.2.1 Living Lab Model with Fluctuating Resource Price

The financial production cells were integrated into the living lab model as shown in Figure 28. The primary objective of the simulation is to demonstrate that the financial production cell is able to dynamically re-assign the resource price and calculate the corresponding cost in a complex system during the simulation. A static price curve and a dynamic price curve for Electricity and Natural gas are displayed in Figure 29 and Figure 30, respectively the cost comparisons of each resource are illustrated in Figure 31, Figure 32, Figure 33 and Figure 34.

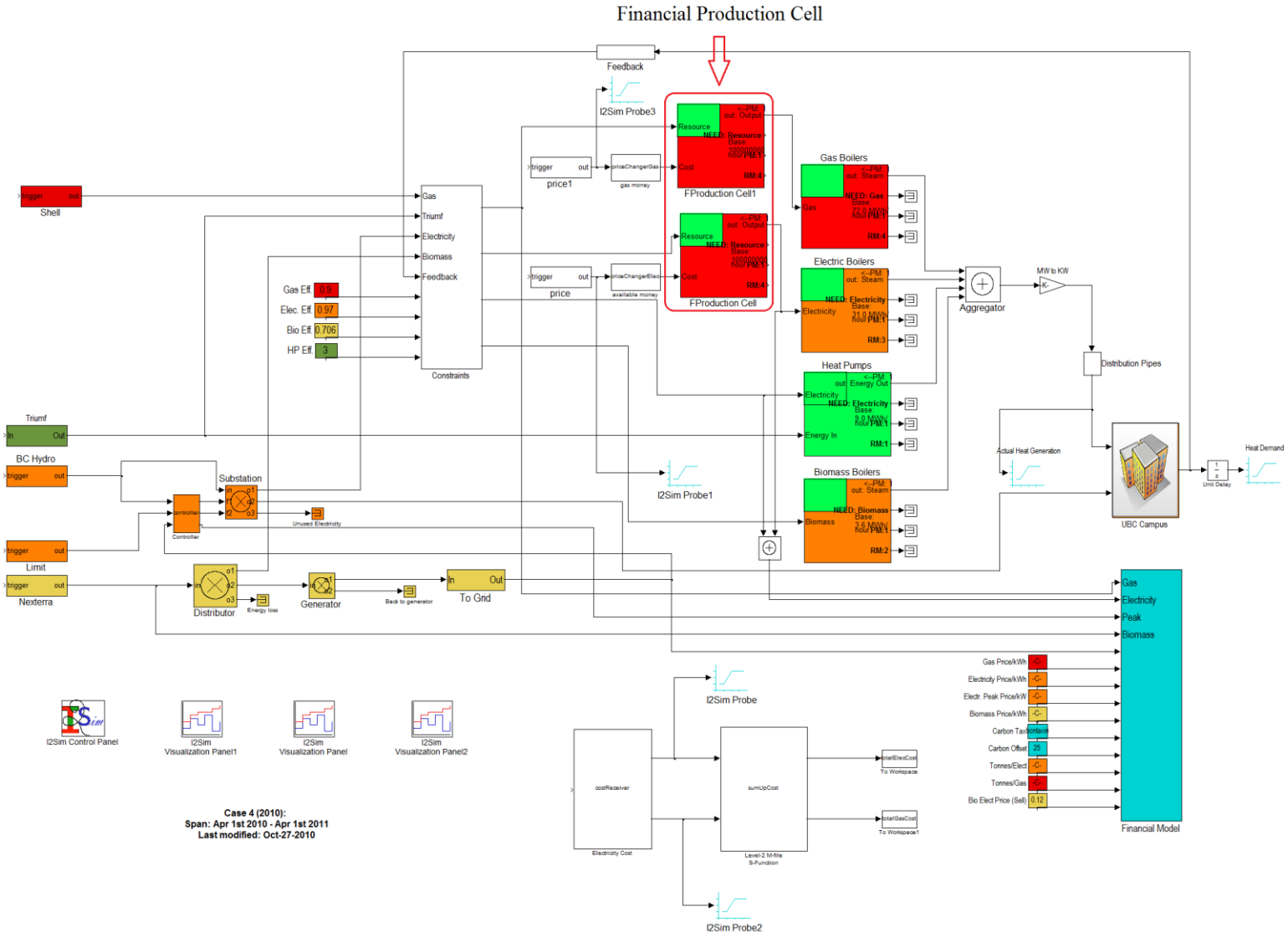


Figure 28: UBC living lab simulation model with financial production cell

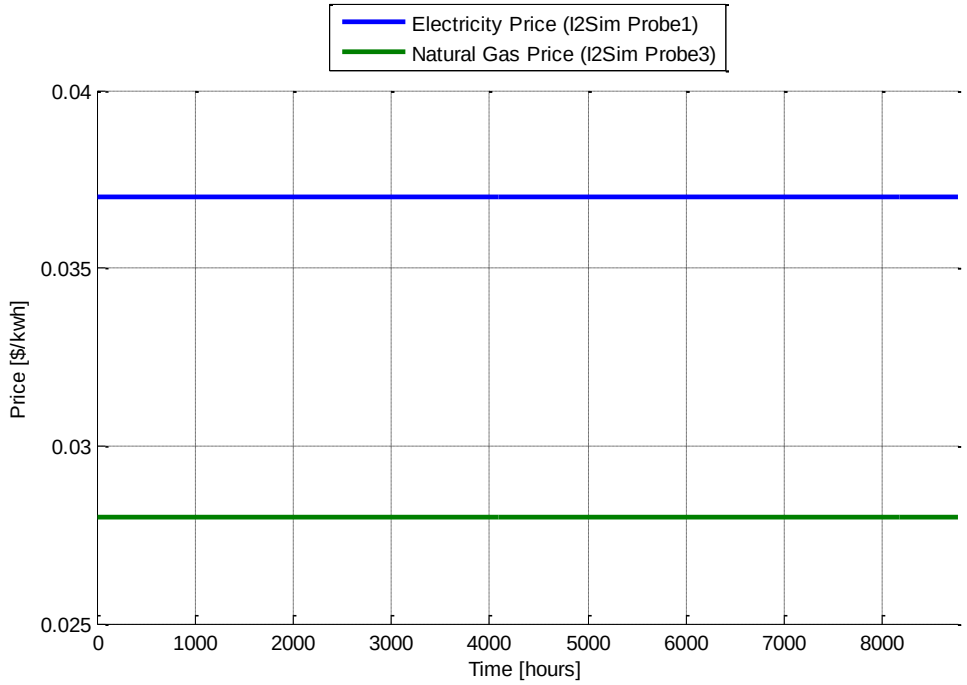


Figure 29: Static price of electricity and natural gas

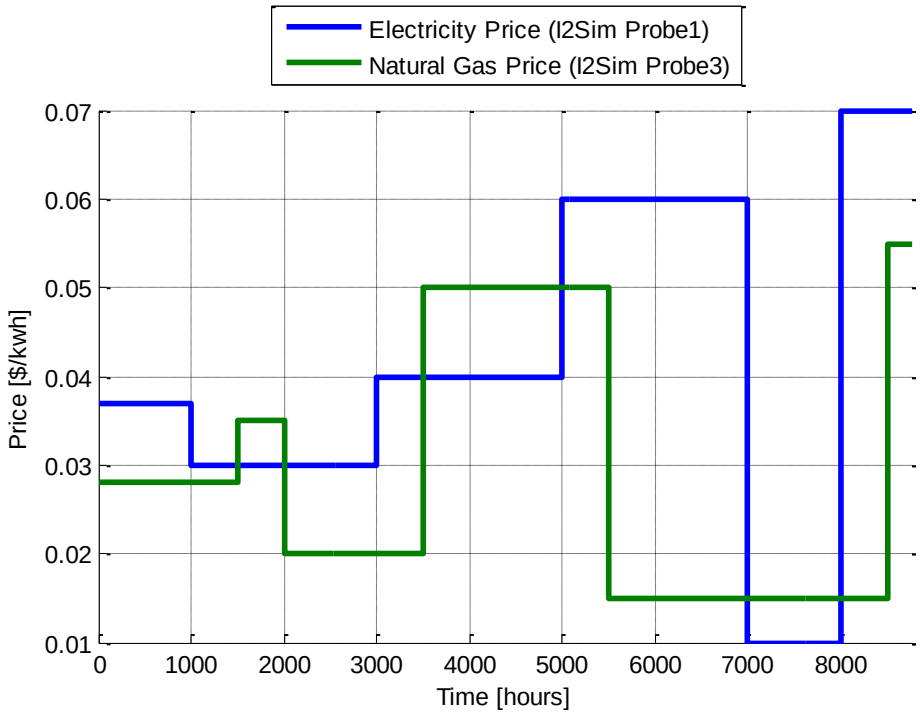


Figure 30: Dynamic price of electricity and natural gas

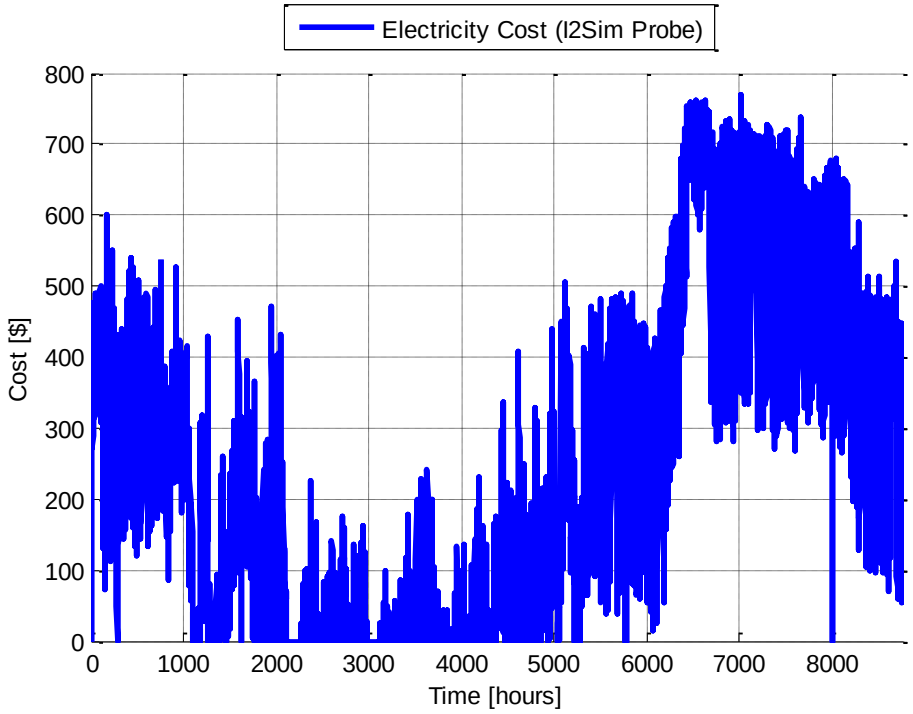


Figure 31: Electricity consumption cost with static price

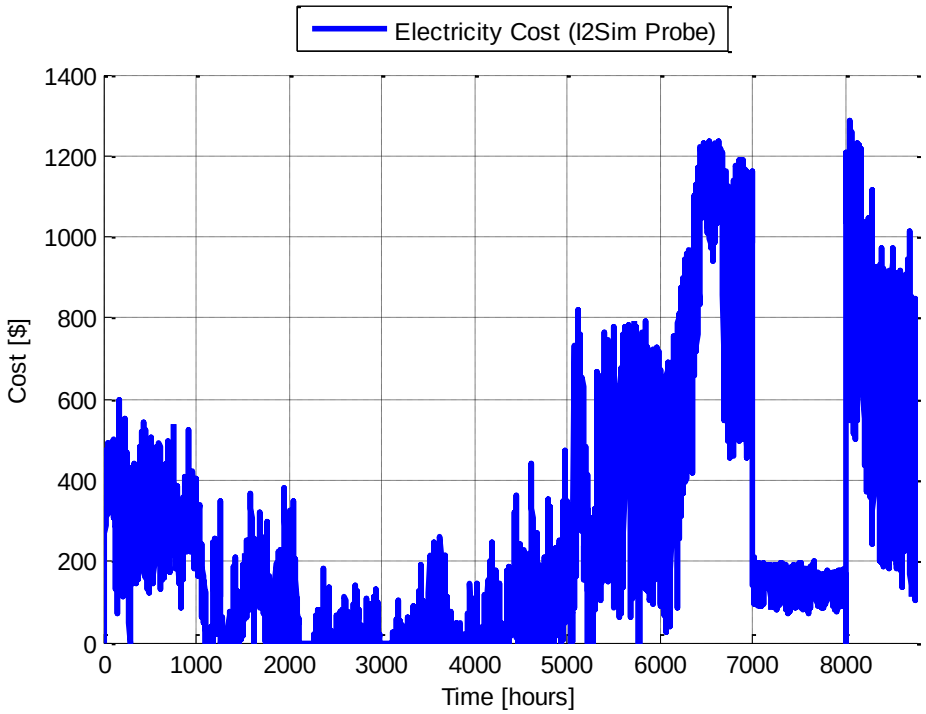


Figure 32: Electricity consumption cost with dynamic price

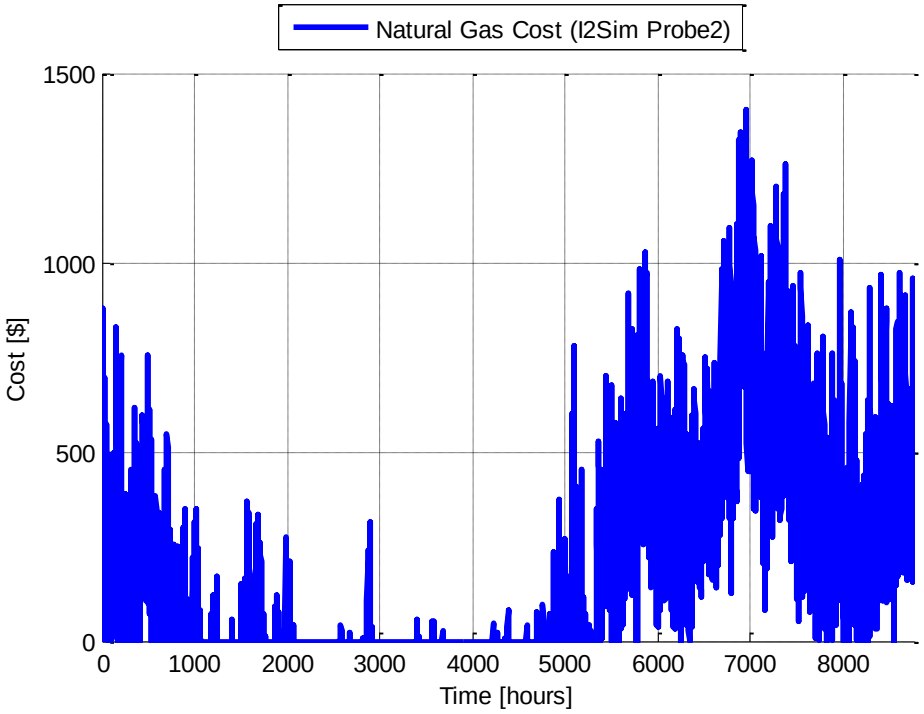


Figure 33: Natural gas consumption cost with static price

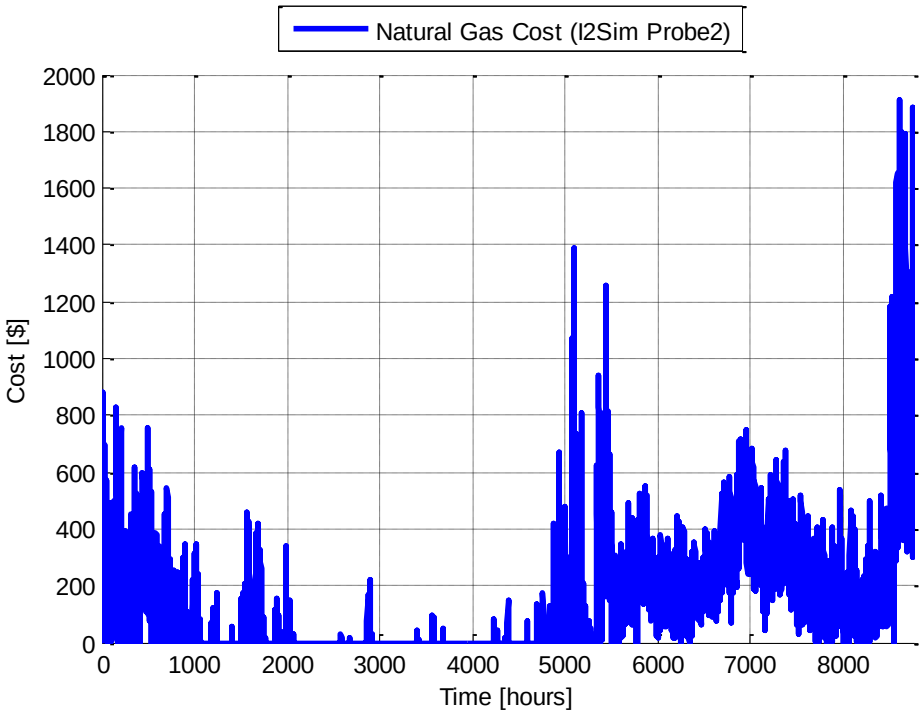


Figure 34: Natural gas consumption cost with dynamic price

Compared with the static price, the cost of both electricity and natural gas with dynamic price adjustment changed drastically. This was caused by the fluctuation of price between electricity and natural gas shown in Figure 30. The price and cost of the resources were handled by the HRT in the financial production cell. The process of updating prices was achieved by dynamically updating the HRT table and base value during the simulation. The results of this simulation fulfilled the initial goal of the testing, which provided solid evidence that the financial production cell is capable of handling dynamic data.

3.3 Living Lab Battery System

The UBC campus consumes large amount of electricity due to its dense population and various research facilities. UBC's hydro bill includes two parts: the energy used and the energy demand [**27**]. The energy use charge is the energy consumption cost, which is based on how much electrical energy has been consumed; the energy demand charge is the peak demand cost, which is based on the highest peak demand of the month. On the BC Hydro bill for the energy demand charge corresponds to 30% of the total cost. The objective of this simulation is to test whether battery storage could be used to shave the peak and be financially feasible compared with the batteries capital cost.

3.3.1 General Battery Model

The general battery model represents the common battery behavior, it discharges electricity when it is needed and charges electricity when it is available. A battery age factor is also added to the model in order to make it more realistic. Figure 35 shows the overview of the model. And

Figure 36 shows the relationship among three function blocks: controller, discharge and charge.

The Inputs and outputs of the model are:

- Power demand: the amount of electric power needed from the battery.
- Power available: the amount of electric power available to charge the battery.
- Battery age factor: how well the battery is preserved since the initial stage (e.g. 0.8 means the battery can only be charged up to 80% of its initial maximum capacity.)
- Power output: the amount of electric power drained from the battery.
- Power stored: the amount of electric power stored in the battery.
- Battery capacity: the level of battery charged capacity (e.g. fully charged or 60% charged)

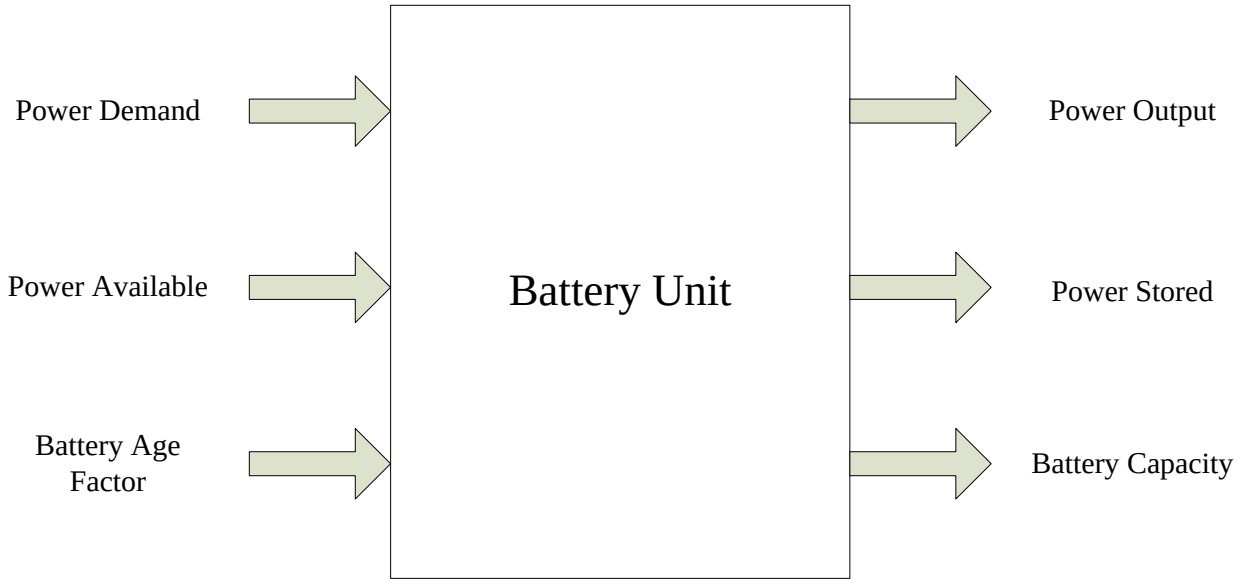


Figure 35: Battery unit

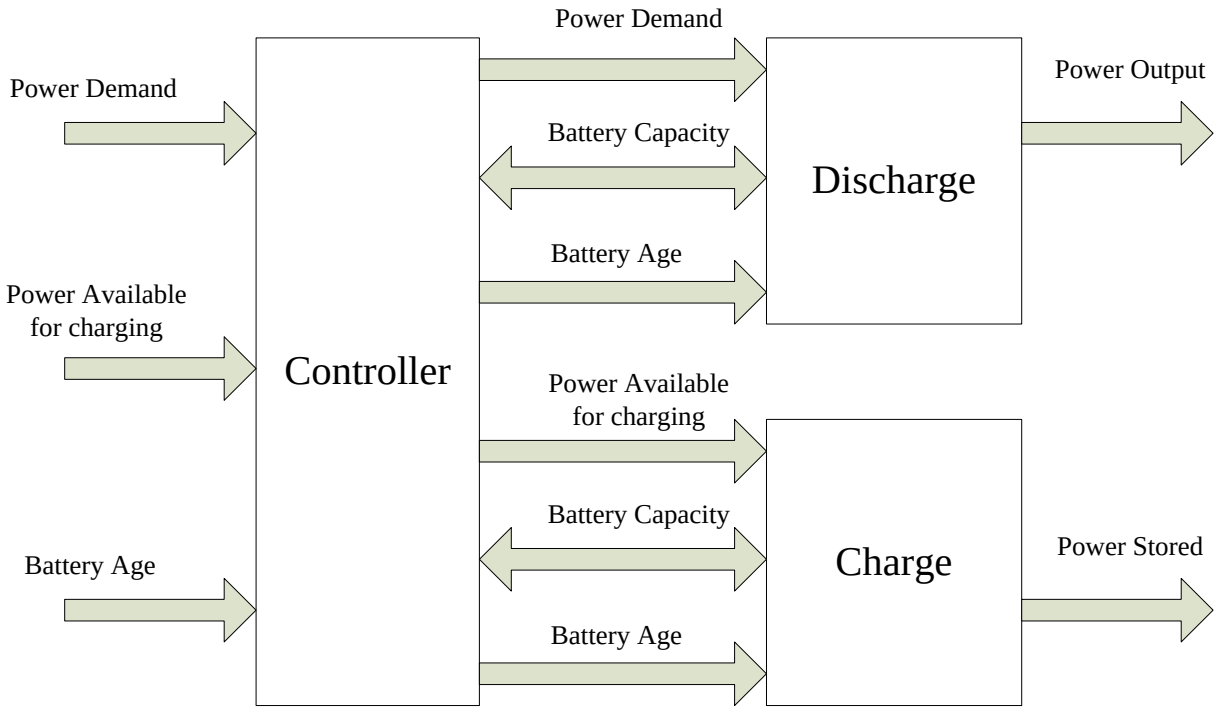


Figure 36: Battery functions

The battery charge and discharge share similar characteristic curves. The curves contain two major areas: nominal area and exponential area [**28**]. In the nominal area, the battery charge and discharge occur at a constant nominal voltage, which depends on the battery type; while in the exponential area, the battery's operating voltage varies between nominal voltage and maximum voltage [**29**]. A typical battery discharge characteristic is shown in Figure 37, and a typical battery charge characteristic is shown in Figure 38.

Battery efficiency includes both charge efficiency and discharge efficiency. In the battery model, they are controlled by efficiency module.

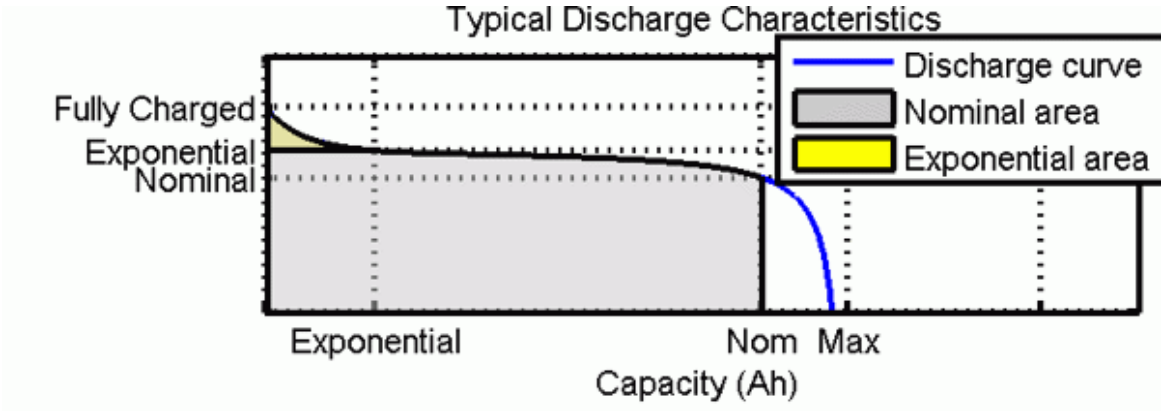


Figure 37: Typical battery discharge characteristics [30**]**

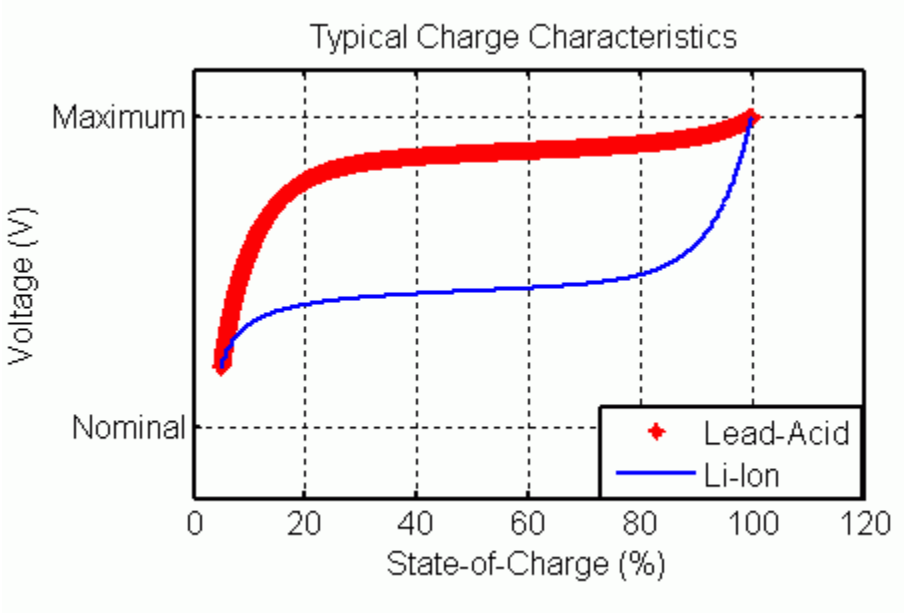


Figure 38: Typical battery charge characteristics [30**]**

In order to design a simple but sufficient battery model in I2Sim, the battery characteristic curves shown in Figure 37 and Figure 38 are represented by two areas: nominal area and exponential area, which is shown in Figure 39.

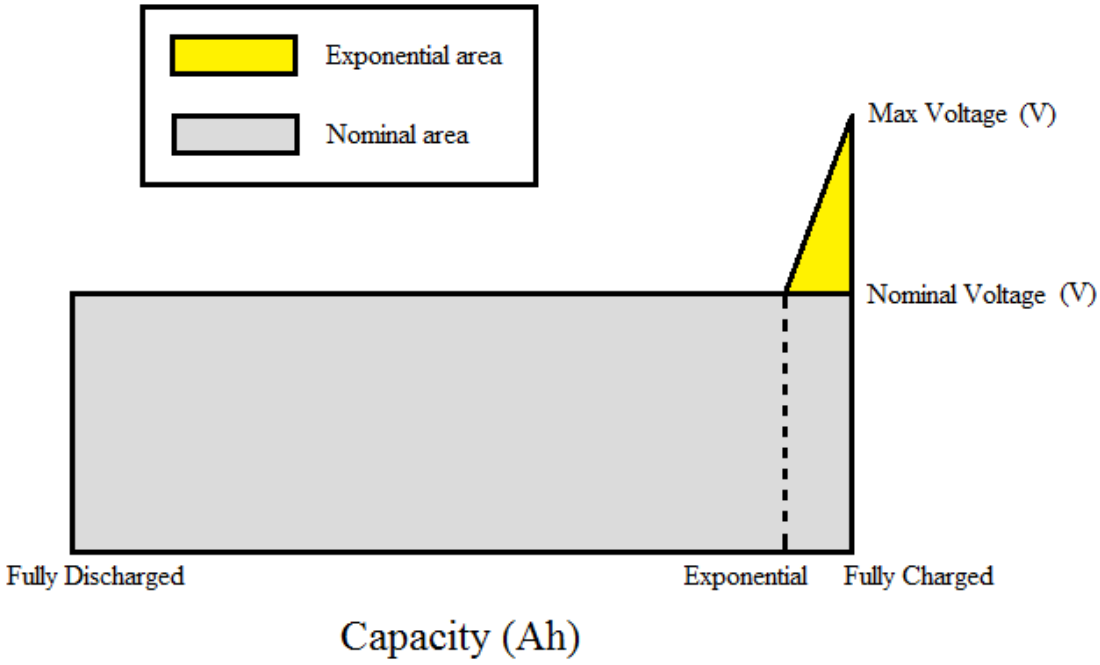


Figure 39: Battery model discharge and charge characteristic

3.3.2 Simulation Results

Three different battery types were simulated and tested. They are: Flow Battery, Lithium-ion Battery and Sodium Sulfur Battery. The test data, including battery discharge and charge characteristics, total energy capacity and battery costs were provided by students from the Materials Engineering Department. The electricity consumption data used in this simulation is the same consumption data used in the previous living lab model simulation. The peak-shaving threshold was initially set at 38MW, which means that when UBC's campus electricity demand is beyond 38MW, the battery system will step in and deliver the power as much as it is able to; when the electricity demand is below 38MW, BC Hydro will supply UBC's campus demand, and in the mean time, the battery system will recharge itself for the next power demand period.

3.3.2.1 Flow Battery

The flow battery stores electro-active materials externally and circulates these reactants through an electro-chemical cell which converts the chemical energy directly into electric energy [**31**] [**32**].

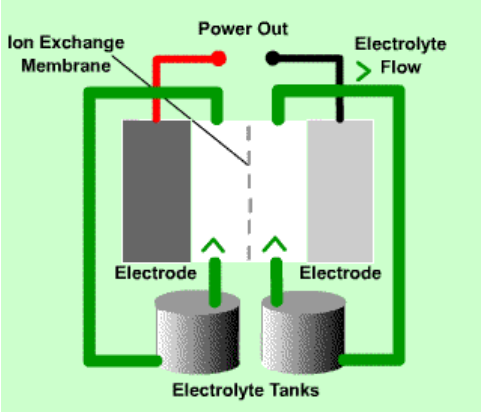


Figure 40: Flow battery diagram [31**]**

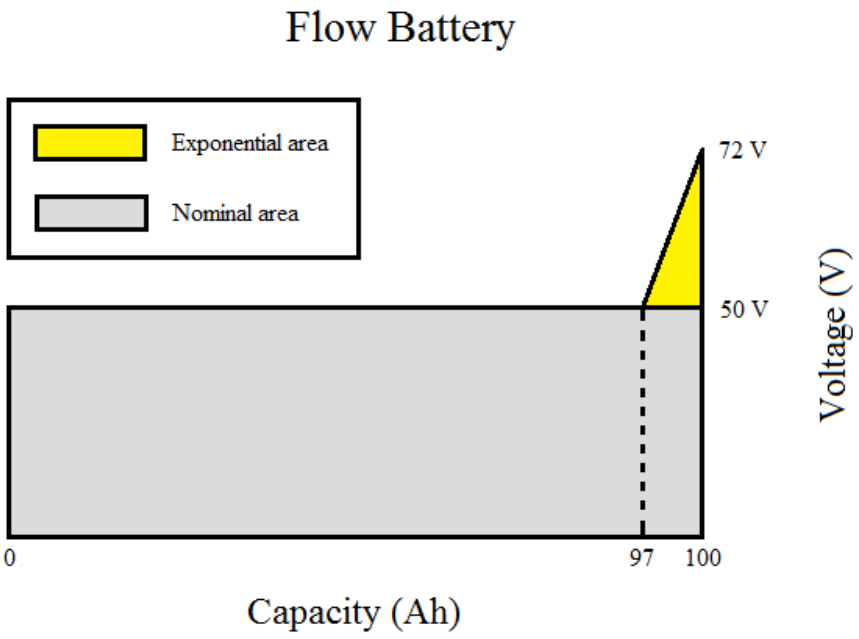


Figure 41: Flow battery discharge and charge characteristic

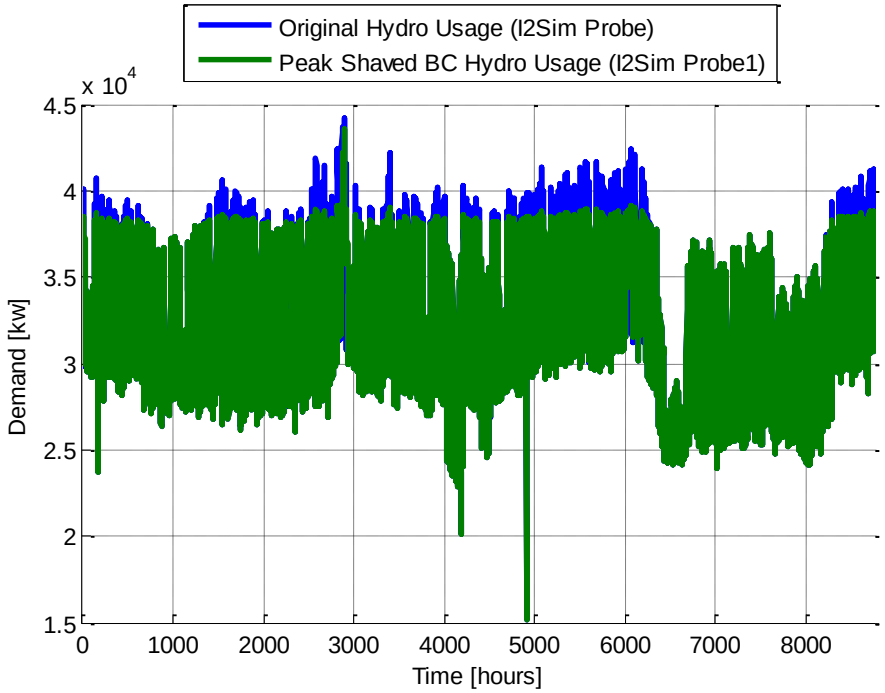


Figure 42: UBC electricity consumption with and without flow battery

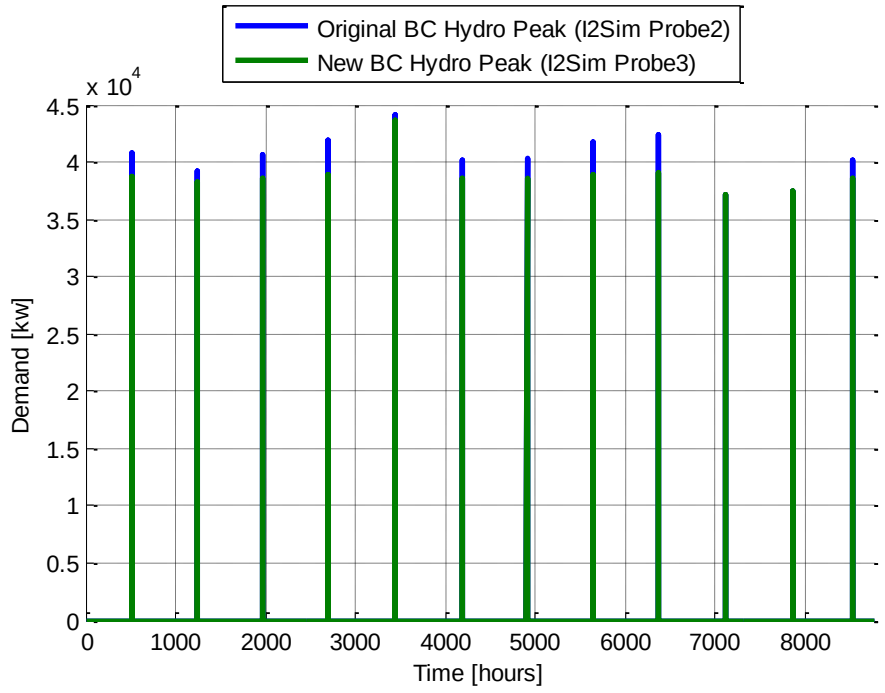


Figure 43: UBC peak demand with and without flow battery

3.3.2.2 Lithium-ion Battery

The lithium-ion battery is one of the most widely used batteries in current new applications. In this type of battery, lithium ions travel from negative to positive during discharge, and vice-versa during charge [**33**].

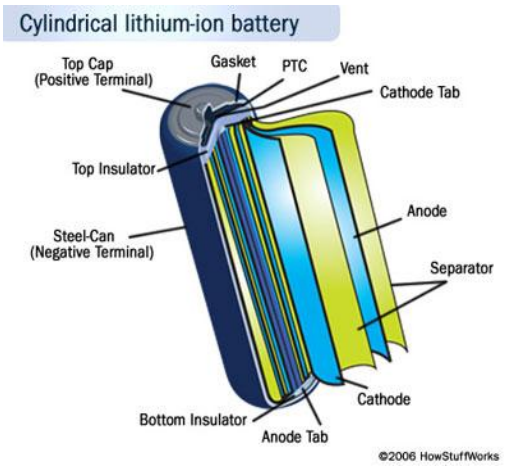


Figure 44: Lithium-ion battery diagram [33**]**

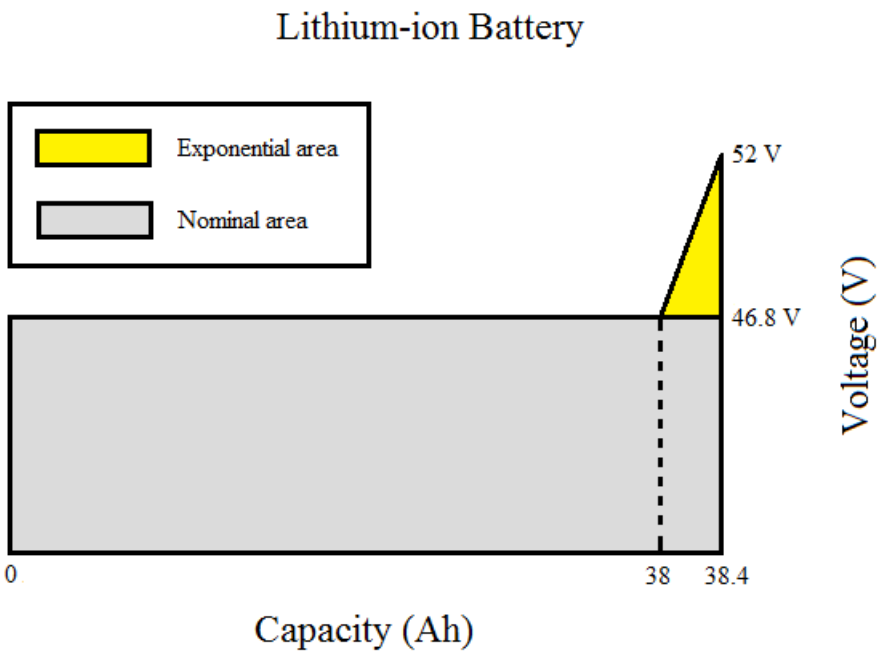


Figure 45: Lithium-ion battery discharge and charge characteristic

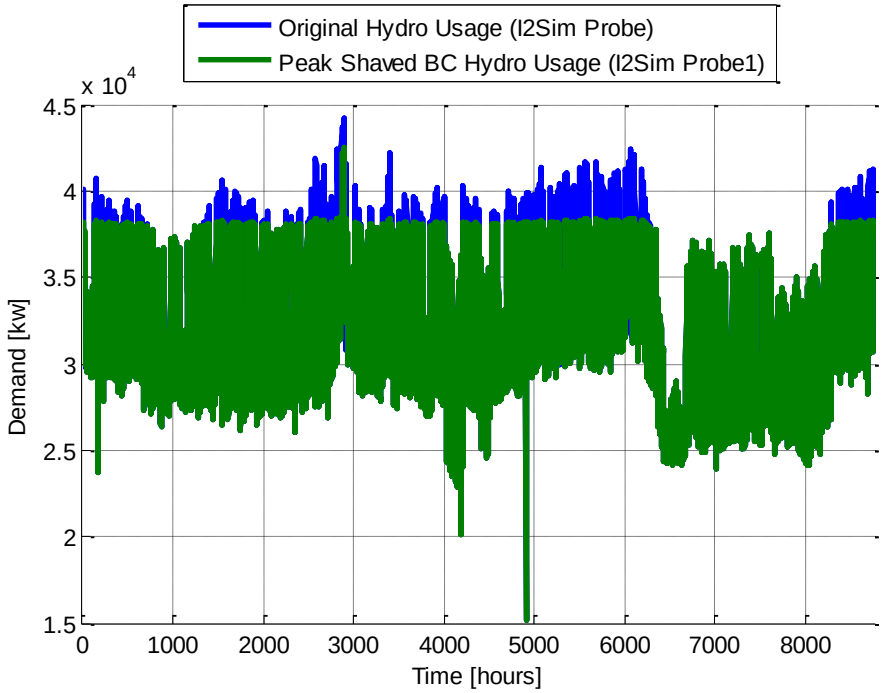


Figure 46: UBC electricity consumption with and without lithium-ion battery

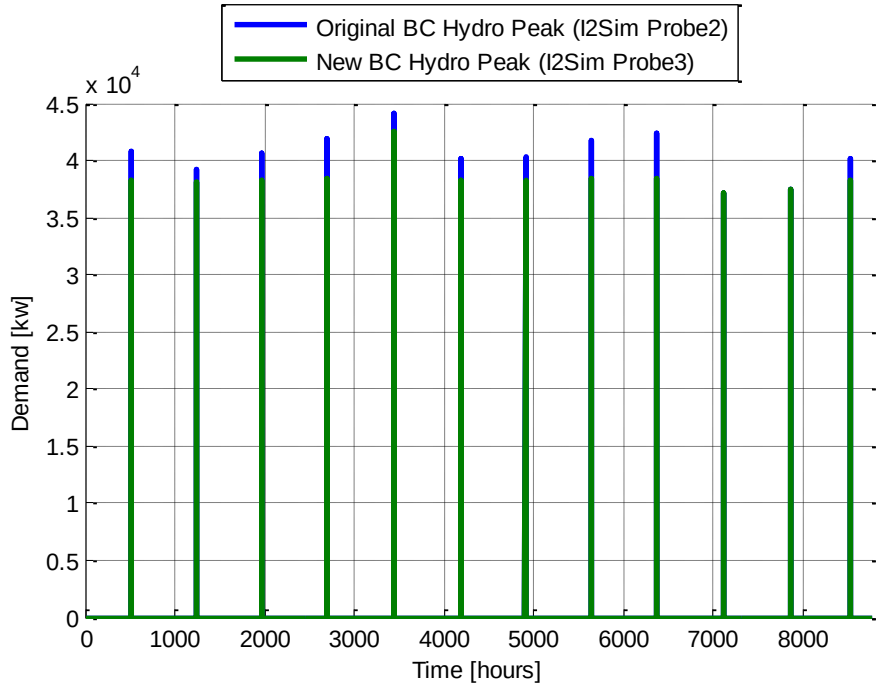


Figure 47: UBC peak with and without lithium-ion battery

3.3.2.3 Sodium Sulfur (NaS) Battery

The Sodium Sulfur (NaS) Battery has liquid sulfur at the positive terminal and liquid sodium at the negative terminal. The active materials are separated by a solid beta alumina ceramic electrolyte [**34**] [**35**].

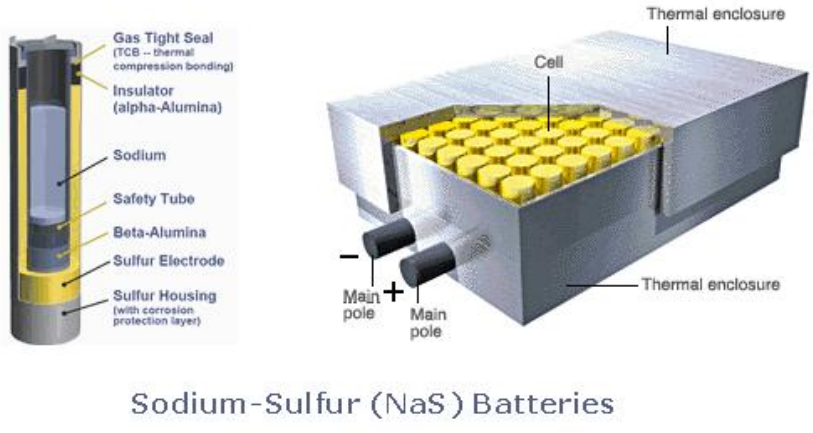


Figure 48: Sodium-sulfur (Nas) battery diagram [36**]**

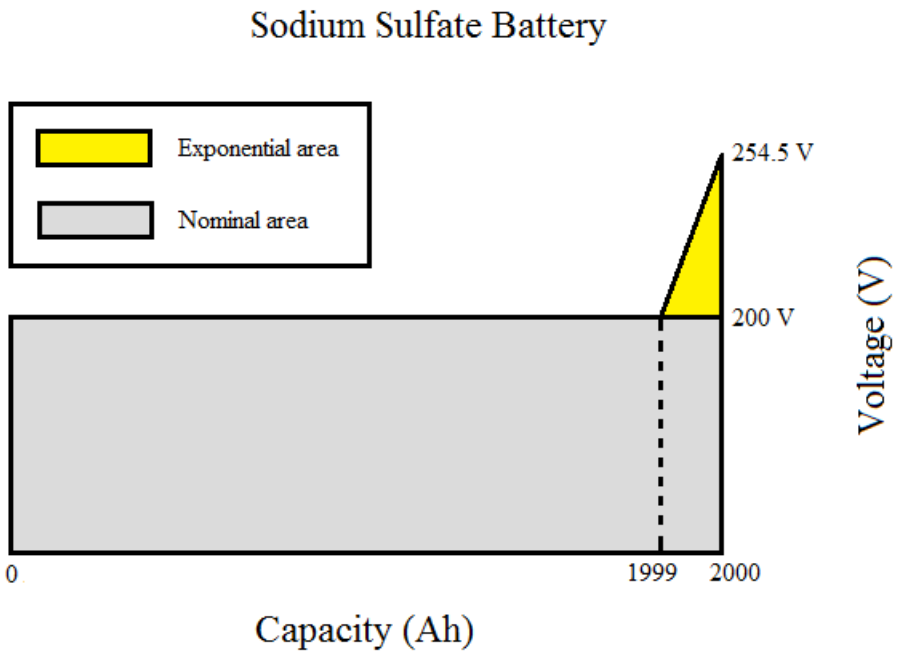


Figure 49: Sodium sulfur battery discharge and charge characteristic

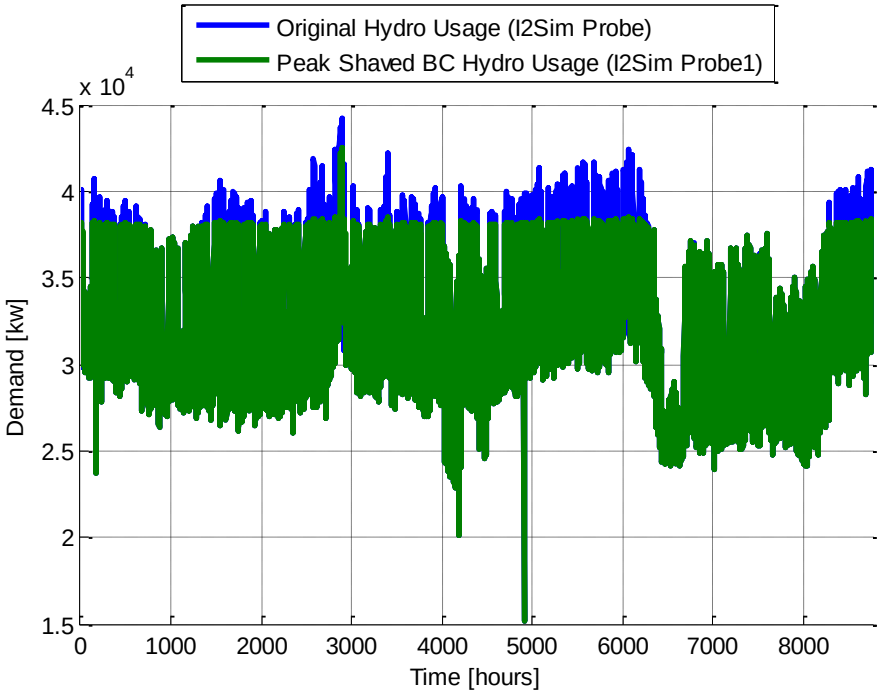


Figure 50: UBC consumption with and without sodium sulfur battery

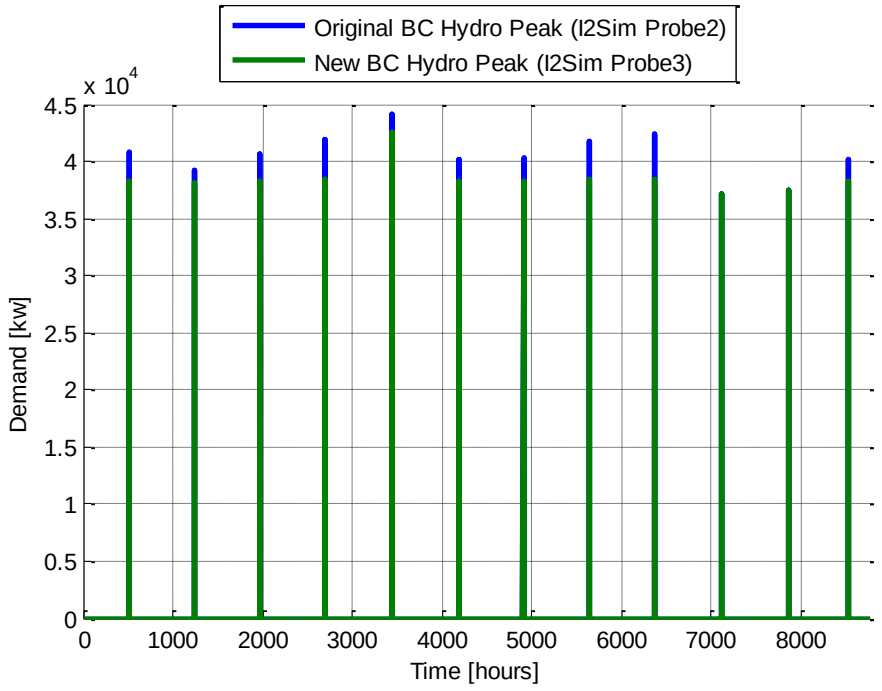


Figure 51: UBC peak with and without sodium sulfur battery

The Simulation results presented in Figure 42, Figure 46 and Figure 50 are the electricity consumption comparison of three battery models. The blue curve shows how much BC Hydro electricity is originally needed to supply the UBC campus. The green curve shows that after employing the battery system, how much electricity is still needed from BC Hydro to supply the campus load. In most of the period, the battery system was capable of shaving off the peak demand. However, in the time line around 3000 hours, the battery system reached its full capacity. Therefore, BC Hydro had to take over and supply that period.

In Figure 43, Figure 47 and Figure 51, the blue lines represent how much were the original peak demands of each month. The green lines represent how much are the new peak demands after applying the battery system. In the graph, the green lines are on top of the blue lines. Therefore, the visible blue sections are the shaved peaks.

For the whole UBC Campus, the capacities of all three batteries were set at 37.5 MWh. The interest rate for the financing of the batteries is 2.5%. The financial performance and analysis of the three battery types are shown in Table 4.

Table 4: Battery system financial analysis

Battery Type	Battery Life Span (year)	Battery Cost ($)	Annual Cost Reduction from Peak Shaving ($)	Net Present Value ($)
Flow Battery	31	$ 18,750,000	$ 101,678	− $ 15,609,771
Lithium-ion Battery	3.5	$ 15,000,000	$ 139,526	− $ 14,511,812
Sodium Sulfur Battery	8	$ 18,750,000	$ 137,484	− $ 17,651,090

Although each battery has indeed reduced the hydro cost by shaving off the peak demand, the final results show that they all have negative net present value. Therefore, they are not financially feasible.

3.3.2.4 Delay South Transmission Line's Upgrading Process with Battery System

The south transmission line's emergency line capacity (protection setting) is 55.39 MVA [**18**]. Each year UBC's electrical power demand grows at a rate of 109%. As a result, the south transmission line will exceed its emergency capacity by the year 2013, which is shown in Table 5.

Table 5: Expected peak demands from 2010 to 2013

Year	UBC's Electricity Peak Demand of Each Month (MVA)											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
2010	40.70	39.17	40.65	41.86	44.18	40.16	40.34	41.71	42.36	37.13	37.54	40.12
2011	44.36	42.70	44.31	45.63	48.16	43.77	43.97	45.46	46.17	40.47	40.92	43.73
2012	48.36	46.54	48.30	49.73	52.49	47.71	47.93	49.56	50.33	44.11	44.60	47.67
2013	52.71	50.73	52.64	54.21	57.21	52.01	52.24	54.02	54.86	48.08	48.62	51.96

The cost of upgrading existing south transmission line is about 50 Million Canadian Dollars. If the upgrading process could be delayed for one year, the 50 Million initial capitals would be available for investing in elsewhere. The purpose of this test is to investigate whether it is financially feasible to use the battery system to delay the upgrading process.

In this test, the battery capacity is re-set to 68.75 MWh, so that it will reduce the highest peak demand below 55.39 MVA. The threshold of peak shaving is also dynamically set to shave off 30% of peak for each month, which will take advantage of the full capacity of the battery system. The test results of three battery types are shown in Figure 52, Figure 53, Figure 54, Figure 55, Figure 56 and Figure 57.

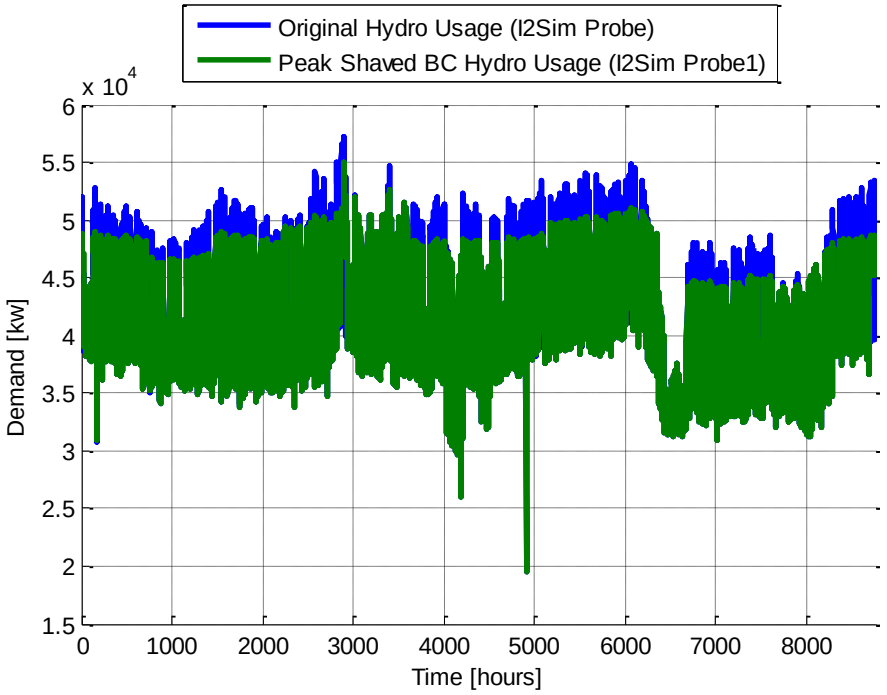


Figure 52: UBC consumption with and without flow battery's dynamic peak shaving

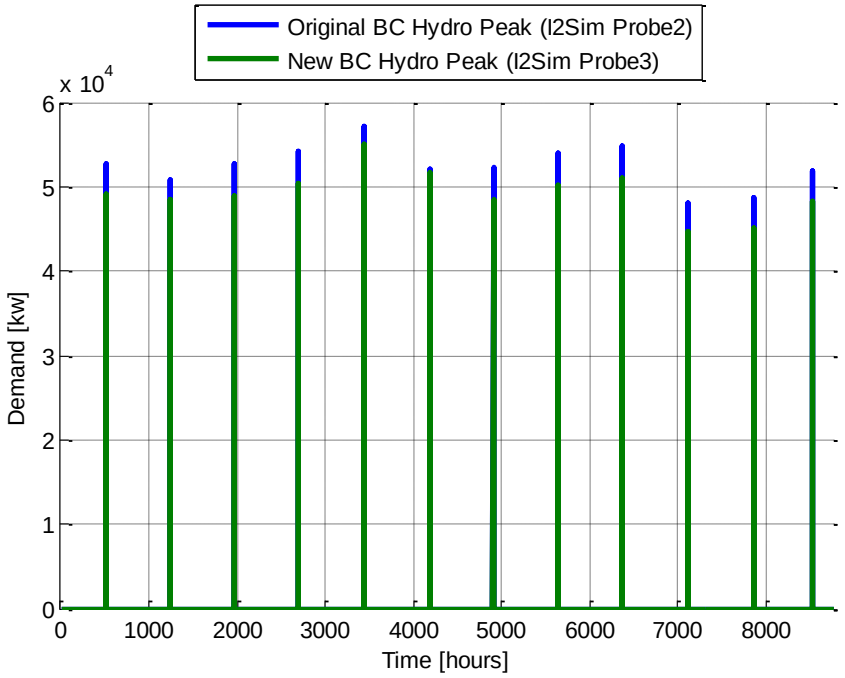


Figure 53: Dynamic peak shaving by using flow battery

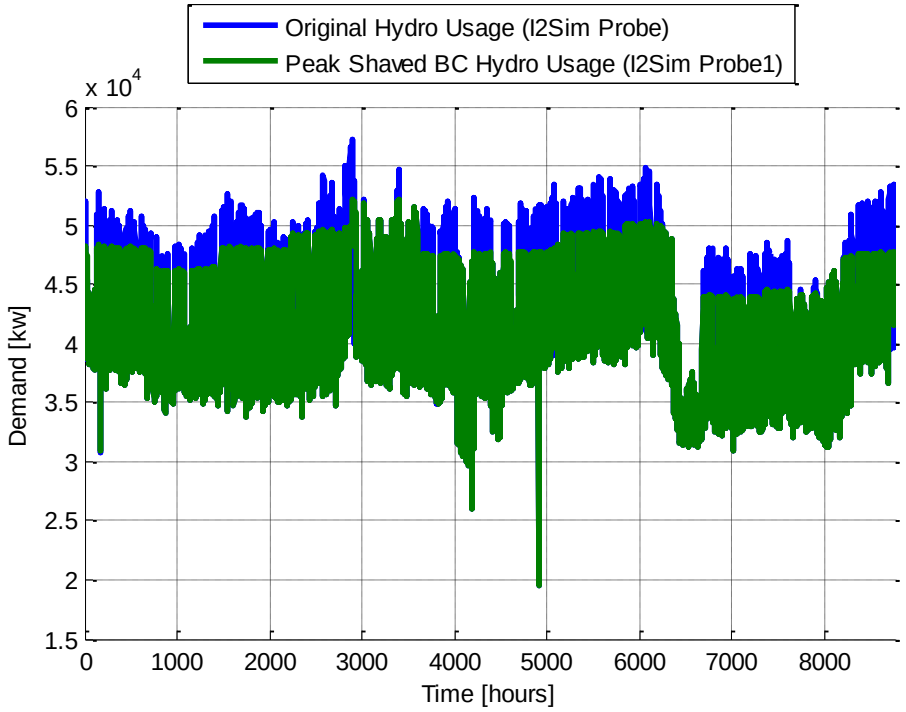


Figure 54: UBC consumption with and without lithium-ion battery's dynamic peak shaving

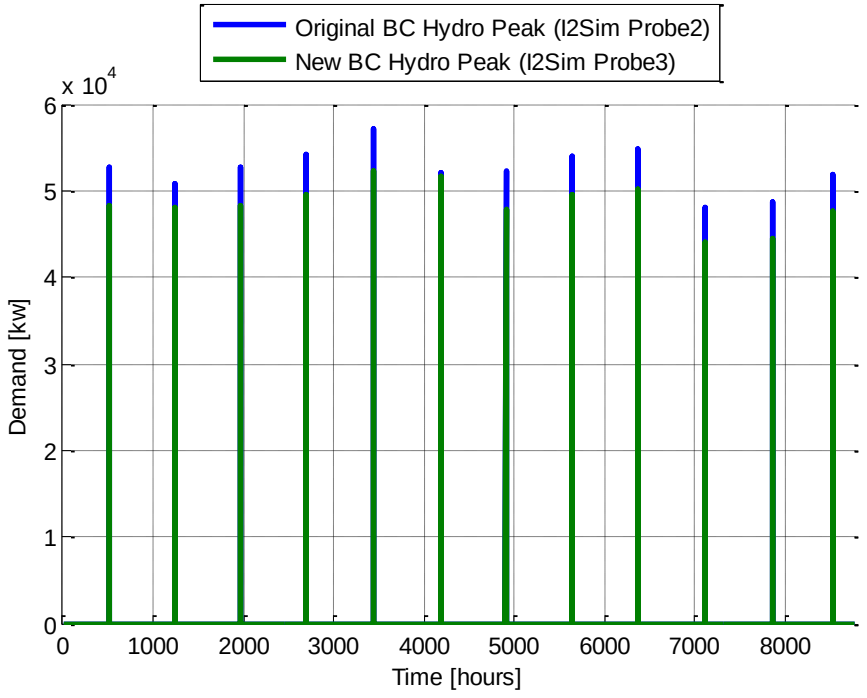


Figure 55: Dynamic peak shaving by using lithium-ion battery

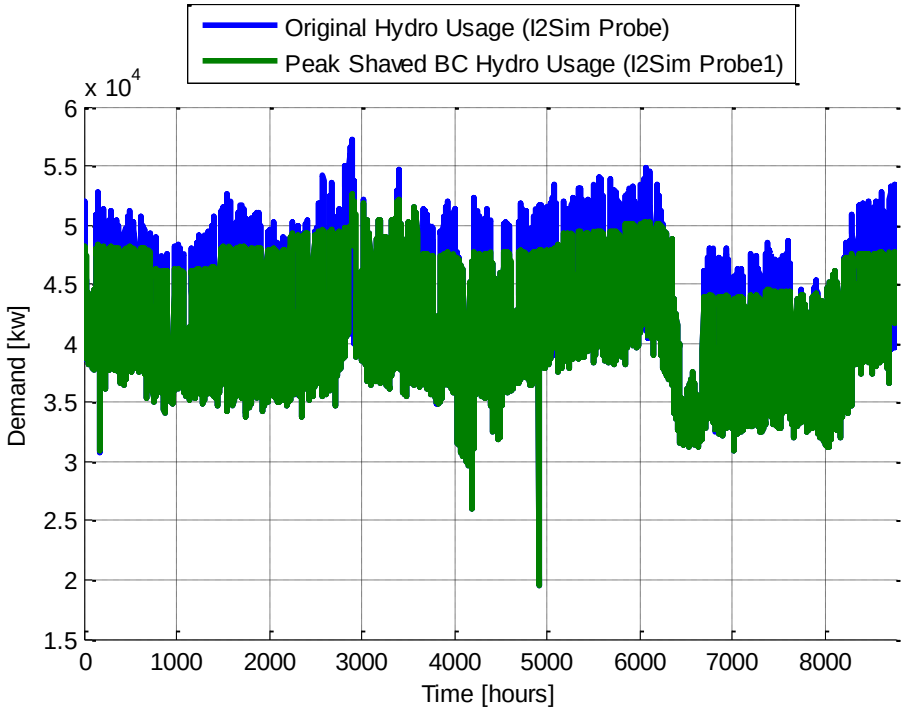


Figure 56: UBC consumption with and without NaS battery's dynamic peak shaving

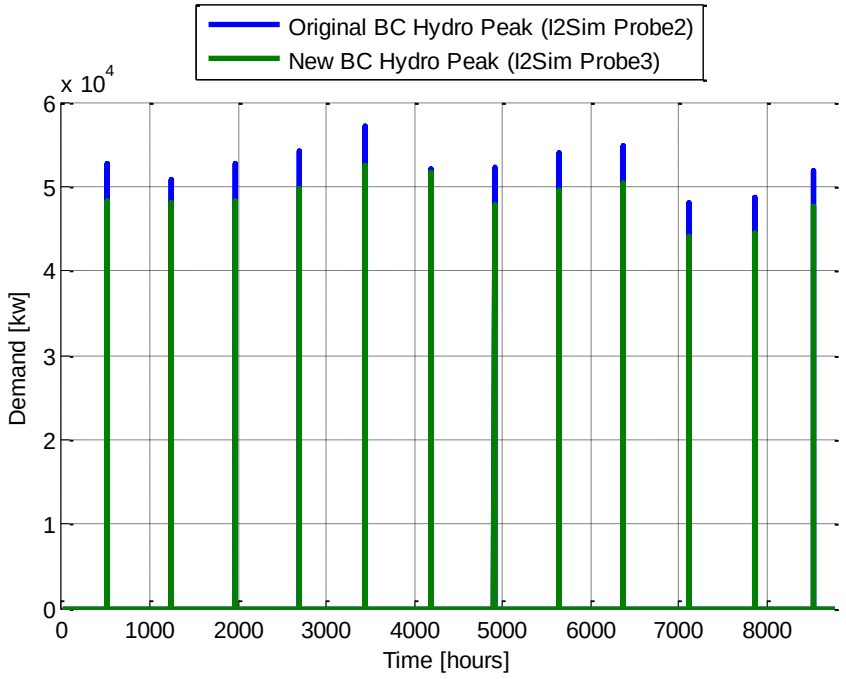


Figure 57: Dynamic peak shaving by using NaS battery

In the simulation results, the new peak demands of the year 2013 are reduced significantly and all stay below 55.39 MVA as shown in Table 6.

Table 6: New peak demands after dynamic peak shaving

Battery Type	UBC's Electricity Peak Demand of Each Month (MVA)											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Flow	49.03	48.52	48.95	50.41	55.06	51.57	48.46	50.23	51.01	44.71	45.21	48.32
Lithium-ion	48.28	48.08	48.21	49.65	52.21	51.57	47.8	49.47	50.24	44.03	44.53	47.59
Sodium Sulfur	48.33	48.11	48.26	49.7	52.59	51.57	47.84	49.52	50.29	44.08	44.57	47.64

The life span, total capital cost and annual cost reduction from each battery type are shown in Table 7.

Table 7: Battery dynamic peak shaving

Battery Type	Battery Life Span (year)	Battery Cost ($)	Annual Cost Reduction from Peak Shaving ($)
Flow Battery	24	$ 34,375,000	$ 190,660
Lithium-ion Battery	3	$ 27,500,000	$ 272,203
Sodium Sulfur Battery	6	$ 34,375,000	$ 265,739

Since the upgrading process of the south transmission line could be delayed, if 50 million initial capitals were deposited in the bank with a 3% interest rate, some interest could be gained each year. The total capital cost of the battery system is paid by an annuity loan with 9% growth factor and 5% interest rate, because in each year, an additional 9% of battery has to be purchased to keep up with the campus power demand growth.

Table 8: Financial analysis of flow battery's dynamic peak shaving

year	Flow Battery		Annual Interest Gain from Bank (million $)	Total Annual Gain (million $)
	Annual Capital Cost (million $)	Annual Cost Reduction from Peak Shaving (million $)		
1	-1.43	0.19	1.50	0.26
2	-1.64	0.21	3.05	1.62
3	-1.87	0.23	4.64	2.99
4	-2.14	0.25	6.28	4.38
5	-2.45	0.27	7.96	5.78
6	-2.81	0.29	9.70	7.19
7	-3.21	0.32	11.49	8.60
8	-3.68	0.35	13.34	10.01
9	-4.21	0.38	15.24	11.41
10	-4.82	0.41	17.20	12.79
11	-5.51	0.45	19.21	14.15
12	-6.31	0.49	21.29	15.47
13	-7.22	0.53	23.43	16.74
14	-8.26	0.58	25.63	17.95
15	-9.46	0.63	27.90	19.08
16	-10.82	0.69	30.24	20.11
17	-12.39	0.75	32.64	21.01
18	-14.18	0.82	35.12	21.77
19	-16.22	0.90	37.68	22.35
20	-18.57	0.98	40.31	22.71
21	-21.25	1.06	43.01	22.83
22	-24.32	1.16	45.81	22.64
23	-27.84	1.27	48.68	22.11
24	-31.86	1.38	51.64	21.16

Table 9: Financial analysis of lithium-ion battery's dynamic peak shaving

year	Lithium-ion Battery		Annual Interest Gain from Bank (million $)	Total Annual Gain (million $)
	Annual Capital Cost (million $)	Annual Cost Reduction from Peak Shaving (million $)		
1	-9.17	0.27	1.50	-7.40
2	-10.49	0.29	3.05	-7.15
3	-12.01	0.32	4.64	-7.05

Table 10: Financial analysis of NaS battery's dynamic peak shaving

year	NaS Battery		Annual Interest Gain from Bank (million $)	Total Annual Gain (million $)
	Annual Capital Cost (million $)	Annual Cost Reduction from Peak Shaving (million $)		
1	-5.73	0.27	1.50	-3.96
2	-6.56	0.29	3.05	-3.22
3	-7.51	0.32	4.64	-2.55
4	-8.59	0.34	6.28	-1.97
5	-9.83	0.38	7.96	-1.49
6	-11.25	0.41	9.70	-1.14

The financial analysis results shown in Table 8, Table 9 and Table 10 indicate that the flow battery has successfully generated a positive financial gain in its 24 year life span. It proved that by employing the flow batteries, not only can the peak demands be reduced below emergency capacity, but also be financially feasible.

3.4 Simulation Conclusion

The purpose of the four different test simulations was to test the key features of the financial production cell (FPC) model, and further explore the potential application of the FPC in various complex models. The results of the first test case demonstrate the capabilities of the FPC that are important for portraying the real life financial behavior.

The second test case integrated the financial production cell with the UBC Living Lab model, which was a large scaled complex model. The results of the simulation proved that the model FPC was able to process large amount of dynamic data in real time.

In the third test case, three independent battery models were tested with the FPC. Even though the final financial results did not favor the battery solution, they showed that the FPC is a good tool for project financial analysis.

The forth test case investigates the possibilities of using battery system to delay upgrading the south transmission line. The results indicate that the project is both technically and financially feasible.

4 Conclusion and Future Work

The Infrastructure Interdependencies Simulator (I2Sim) framework provides a sophisticated platform to explore the interdependencies among different infrastructures under disaster situation. However, in most of peace time, all the infrastructures are free from the damage caused by disasters. The interdependencies among them are mostly business and operational relationships. In order to simulate financial interdependencies and extend I2Sim's capabilities, the Financial Production Cell (FPC) was developed in this thesis.

The financial production cell was derived from the regular production cell; it comes with all the features in the regular production cell, and adds its own flavor. The new features include the capability of outputting accurate financial results and updating HRTs during real time simulation. Besides its internal operation features, the FPC is connected with several user interfaces, which provide the end users an intuitive experience when they interact with the I2Sim environment.

The simulation results documented in this thesis provide solid evidence to prove the financial production cell's capabilities and demonstrate the strength of the I2Sim simulation environment. Some other features of the FPC, which need to be improved in the future, have also been discovered during the simulation. These features are:

- The financial production cell only reads the HRT table's parameters when there is a change in the inputs, regardless of whether the HRT itself is updated or not. This could cause potential errors when the HRT table is updated, but the outputs are still being calculated based on an old HRT table, because the inputs are static and it do not trigger the FPC to read the new HRT table.

- The majority of the end users who use the financial production cell probably will not have sufficient knowledge about I2Sim. However, in the current stage of development, in order to set up the initial HRT tables, it is required that the end user manually input the FPC table content and base value.
- The financial production cell physical mode still changes its color based on the input. This feature is inherited from the regular production cell's physical mode, which indicates the physical condition of the cell. In the financial production cell, the physical condition does not exist. Therefore, the color code is meaningless in this case.

The financial production cell has proved to be a valuable asset to the I2Sim tool-box family. With the future improvements, it will greatly enhance the I2sim environment.

Bibliography

[1] Jose R. Marti, Carlos E. Ventura, Jorge A. Hollman, K.D. Srivastava, and Hugon Juarez, "I2Sim Modelling and Simulation Framework for Scenario Development, Traning, and Real-Time Decision Support of Multiple Interdependent Critical Infrastructures during Large Emergencies," Applied Science, British Columbia, Vancouver,.

[2] Philippe Kruchten, Carson Woo, Kafui Monu, and Mandana Sotoodeh, "A Human-Centered Conceptual Model of Disasters Affecting Critical Infrastructures," The University of British Columbia, Vancouver, 2007.

[3] Hafiz Abdur Rahman, Mazana Armstrong, DeTao Mao, and Jose R. Marti, "I2Sim: A Matrix-partition based Framework for Critical Infrastructure Interdependencies Simulation," Applied Science, The University of British Columbia, Vancouver, Conference Paper 2008.

[4] "Infrastructure Interdependencies Simulator Technical Description and User Manual," University of British Columbia, Vancouver, User Manual, Sept 2009.

[5] HyunJung Lee, "Infrastructure Interdependencies Simulation (I2Sim) System Model and Toolbox," University of British Columbia, Vancouver, Master Thesis, 2007.

[6] Brent J. Sauder, "The Living Lab - The next evolution of Industry - University Partnerships," The University of British Columbia, Vancouver, 2010.

[7] The University of British Columbia Homepage. [Online]. http://www.ubc.ca/about/

[8] TechFrontiers. [Online]. http://techfrontiers.blogspot.com/2008/08/schools-back-but-with-twist.html

[9] Cesar Lopez, Paul Lusina, and Jose Marti, "REAL-TIME MONITORING OF ENERGY INFRASTRUCTURE," University of British Columbia, Vancouver, 2011.

[10] (2011) UBC energy conversion saves $4 million a year, cuts GHGs. [Online]. http://www.sustain.ubc.ca/steam-hot-water/ubc-energy-conversion-saves-4-million-year-cuts-ghgs

[11] Stantec Consulting, "Alternative Energy Report For University of British Columbia Phase One - Stage One," Vancouver,.

[12] César López, "MULTI-ENERGY SYSTEMS SIMULATIOR FOR HOURLY MANAGEMENT AND OPTIMIZATION OF GHG EMISSIONS AND FUEL COSTS," The University of British Columbia, Vancouver , Master Thesis 2011.

[13] (2010, August) Nexterra to power UBC. [Online]. http://www.globe-net.com/articles/2010/august/17/nexterra-to-power-ubc.aspx

[14] UBC Bioenergy Research and Demonstration Project. [Online]. http://www.nexterra.ca/news/100215_backgrounder.html

[15] Dawn R. Gallagher, "BIOMASS AND CARBON NEUTRALITY," Department of Environmental Protection, 2004.

[16] (2011, July,) Bioenergy Wiki. [Online]. http://bioenergywiki.net/Carbon_neutral

[17] Shahrzad Rostamirad, "INTELLIGENT LOAD SHEDDING SCHEME FOR FREQUENCY CONTROL IN COMMUNITIES WITH LOCAL ALTERNATIVE GENERATION AND LIMITED MAIN GRID SUPPORT," The University of British Columbia, Vancouver, Master Thesis, 2011.

[18] Stantec Consulting, "Alternative Energy Feasibility Report For University of British Columbia Phase Two - Step Three (Final)," Vancouver,.

[19] CIS Group UBC, "UBC-CIS Living Lab Proposal," The University of British Columbia, Vancouver, 2010.

[20] Physics and Astronomy. [Online]. http://www.physics.ubc.ca/research/particle.php

[21] TRIUMF Backgrounder. [Online]. http://www.triumf.info/public/about/background.php

[22] Natural Resources Canada's Office of Energy Efficiency, "Heating and Cooling With a Heat Pump," Natural Resource Canada, Gatineau, 2004.

[23] Heat Pumps South Africa. [Online]. http://www.heatpumpssouthafrica.co.za/heat-pump-geyser/

[24] Lu Liu, "Prototyping and Cells Modeling of the Infrastructure Interdependencies Simulator I2Sim," University of British Columbia, Vancouver, Master Thesis, August, 2007.

[25] Henrik Lund, "EnergyPLAN Advanced Energy Systems Analysis Computer Model ," Aalborg University, 2010.

[26] Ron Mackinnon, "APSC 540 Business Decisions for Engineering Ventures Formula Sheet," The University of British Columbia, Vancouver, 2010.

[27] (2011) BC Hydro Regeneration Demand Charges. [Online]. https://www.bchydro.com/youraccount/content/demand_charges.jsp

[28] MathWorks. (2009) MathWorks-Product Documentation. [Online]. http://www.mathworks.com/help/toolbox/physmod/powersys/ref/battery.html

[29] Woodbank Communications Ltd. (2005) Battery and Energy Technologies. [Online]. http://www.mpoweruk.com/performance.htm

[30] MathWorks. (2011) MathWorks Product Documentation. [Online]. http://www.mathworks.com/help/toolbox/physmod/powersys/ref/battery.html

[31] Trung Nguyen and Robert F. Savinell, "Flow Batteries," The Electrochemical Society, 2010.

[32] David L. Chandler. (2011, June) MITnews. [Online]. http://web.mit.edu/newsoffice/2011/flow-batteries-0606.html

[33] Marshall Brain. How Lithium-ion Batteries Work. [Online]. http://electronics.howstuffworks.com/everyday-tech/lithium-ion-battery1.htm

[34] (2006, January) About Sodium-Sulfur (NaS) Batteries. [Online]. http://thefraserdomain.typepad.com/energy/2006/01/sodiumsulfur_na.html

[35] Hassan Karami, Abbas Yaghoobi, and Ali Ramazani, "Sodium Sulfate Effects on the Electrochemical Behaviors of Nanostructured Lead Dioxide and Commercial Positive Plates of Lead-Acid Batteries," Nano Research Laboratory, Abhar, July 2010.

[36] Electric cells or batteries. [Online]. http://knol.google.com/k/electric-cells-or-batteries#

Appendices

Appendix A Financial User Interface

The complete Matlab code of the Financial User Interface is presented in this Appendix.

```
function varargout = FinalEvaluate(varargin)
% FINALEVALUATE M-file for FinalEvaluate.fig
%      FINALEVALUATE, by itself, creates a new FINALEVALUATE or raises the
existing
%      singleton*.
%
%      H = FINALEVALUATE returns the handle to a new FINALEVALUATE or the
handle to
%      the existing singleton*.
%
%      FINALEVALUATE('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in FINALEVALUATE.M with the given input
arguments.
%
%      FINALEVALUATE('Property','Value',...) creates a new FINALEVALUATE or
raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before FinalEvaluate_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to FinalEvaluate_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help FinalEvaluate

% Last Modified by GUIDE v2.5 08-Jun-2011 14:56:03

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @FinalEvaluate_OpeningFcn, ...
                   'gui_OutputFcn',  @FinalEvaluate_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
```

```
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before FinalEvaluate is made visible.
function FinalEvaluate_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to FinalEvaluate (see VARARGIN)

% Choose default command line output for FinalEvaluate
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes FinalEvaluate wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = FinalEvaluate_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in NPV_pushbutton.
function NPV_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to NPV_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% get the table data and discount rate
tableData = get(handles.uitable1,'Data');
[num_Years,column] = size(tableData)
discountRate = str2double(get(handles.discountRate_editText,'String'));


% calculate NPV of individual case
case1NPV = num2str(calculateNPV(tableData(:,1),discountRate));
case2NPV = num2str(calculateNPV(tableData(:,2),discountRate));
case3NPV = num2str(calculateNPV(tableData(:,3),discountRate));

% display the result on the GUI
```

```
set(handles.case1NPV_staticText,'String',case1NPV);
set(handles.case2NPV_staticText,'String',case2NPV);
set(handles.case3NPV_staticText,'String',case3NPV);
guidata(hObject, handles);

% plot the cash flow graph
axes(handles.axes1);

x = 1:num_Years;
plot(x,tableData(:,1),'r',x,tableData(:,2),'g',x,tableData(:,3),'b');

%adds a title, x-axis description, and y-axis description
title('Yearly Cashflow');
xlabel('Year');
ylabel('Cashflow');

% add legends
legend('Case 1','Case 2','Case 3')
guidata(hObject, handles);

% plot NPV bar graph
bargroup = [str2double(case1NPV); str2double(case2NPV); str2double(case3NPV)]
axes(handles.axes2);
bar(bargroup);
guidata(hObject, handles);


% d = tableData(2,3)+tableData(4,2)
% --- Executes when entered data in editable cell(s) in uitable1.
function uitable1_CellEditCallback(hObject, eventdata, handles)
% hObject    handle to uitable1 (see GCBO)
% eventdata  structure with the following fields (see UITABLE)
%   Indices: row and column indices of the cell(s) edited
%   PreviousData: previous data for the cell(s) edited
%   EditData: string(s) entered by the user
%   NewData: EditData or its converted form set on the Data property. Empty
if Data was not changed
%   Error: error string when failed to convert EditData to appropriate value
for Data
% handles    structure with handles and user data (see GUIDATA)


% --- Executes when selected cell(s) is changed in uitable1.
function uitable1_CellSelectionCallback(hObject, eventdata, handles)
% hObject    handle to uitable1 (see GCBO)
% eventdata  structure with the following fields (see UITABLE)
%   Indices: row and column indices of the cell(s) currently selecteds
% handles    structure with handles and user data (see GUIDATA)



function Years_editText_Callback(hObject, eventdata, handles)
% hObject    handle to Years_editText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Years_editText as text
%        str2double(get(hObject,'String')) returns contents of Years_editText
as a double

%store the contents of input as a string. if the string
%is not a number then input will be empty
input = str2num(get(hObject,'String'));

if (isempty(input))
     set(hObject,'String','0')
end
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function Years_editText_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Years_editText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% % --- Executes on button press in populateTable.
% function populateYears_Callback(hObject, eventdata, handles)
% % hObject    handle to populateTable (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)
%
% % input number of years_edittext (rows) in the table
% num_Years = str2num(get(handles.Years_editText,'String'))
%
% myData = zeros(num_Years, 3)
% %now populate the table with the all zeros
% set(handles.uitable1,'data',myData );



function discountRate_editText_Callback(hObject, eventdata, handles)
% hObject    handle to discountRate_editText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of discountRate_editText as
text
%        str2double(get(hObject,'String')) returns contents of
discountRate_editText as a double
```

```
%store the contents of input as a string. if the string
%is not a number then input will be empty
input = str2num(get(hObject,'String'));

if (isempty(input))
     set(hObject,'String','0')
end
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function discountRate_editText_CreateFcn(hObject, eventdata, handles)
% hObject    handle to discountRate_editText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in loadfromFile_pushbutton.
function loadfromFile_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to loadfromFile_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% load data from a .txt file
fileData = load('cashFlow.txt')

% input the data into uitable1
set(handles.uitable1,'data',fileData);



function loanpaymentPeriod_editText_Callback(hObject, eventdata, handles)
% hObject    handle to loanpaymentPeriod_editText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of loanpaymentPeriod_editText
as text
%        str2double(get(hObject,'String')) returns contents of
loanpaymentPeriod_editText as a double
input = str2double(get(hObject,'String'));

if (isempty(input))
     set(hObject,'String','0')
end
guidata(hObject, handles);
```

```
% --- Executes during object creation, after setting all properties.
function loanpaymentPeriod_editText_CreateFcn(hObject, eventdata, handles)
% hObject    handle to loanpaymentPeriod_editText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in generateCashflow_pushbutton.
function generateCashflow_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to generateCashflow_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% obtain initial capital cost, loan payment period annual maintenance cost,
% annual operation cost, project life span and interest rate for payment
% period

% get initial capital cost
% initialCapital = str2double(get(handles.initialCapital_editText,'string'))
initialCapital = str2double(get_param('Case_4V2/Gas
Boilers','InitCap'))+str2double(get_param('Case_4V2/Electric
Boilers','InitCap'))+str2double(get_param('Case_4V2/Biomass
Boilers','InitCap'))

loanpaymentPeriod =
str2double(get(handles.loanpaymentPeriod_editText,'string'))

% get maintenance cost
% maintenanceCost = str2double(get(handles.maintenanceCost_editText,'string'))
maintenanceCost = str2double(get_param('Case_4V2/Gas
Boilers','MantCost'))+str2double(get_param('Case_4V2/Electric
Boilers','MantCost'))+str2double(get_param('Case_4V2/Biomass
Boilers','MantCost'))

% get operational cost
% operationCost = str2double(get(handles.operationCost_editText,'string'))
operationCost = str2double(get_param('Case_4V2/Gas
Boilers','OperCost'))+str2double(get_param('Case_4V2/Electric
Boilers','OperCost'))+str2double(get_param('Case_4V2/Biomass
Boilers','OperCost'))

num_Years = str2double(get(handles.Years_editText,'string'))
loanpaymentInterestRate =
str2double(get(handles.loanpaymentInterestRate_editText,'string'))
```

```
% calculate the annual ordinary general annuity for loan payment period
annuity = initialCapital/((1-(1+loanpaymentInterestRate)^(-
1*loanpaymentPeriod))/loanpaymentInterestRate)

% sum up all the annual costs without fuel cost
% annualCashFlow = annuity + maintenanceCost + operationCost

% populate the case years
  caseTable = zeros(num_Years,1);

% loading fuel cost and revenue
load('Eleccost.mat')
load('Biocost.mat')
load('Gascost.mat')
load('Revenue.mat')


switch get(handles.case_popupmenu,'Value')
    case 1
        for i = 1:num_Years
    if i <= loanpaymentPeriod
        caseTable(i,1) = -1*(annuity + maintenanceCost + operationCost)-
1000000*(cost_gas+cost_bio+cost_elec)+revenue
    else
        caseTable(i,1) = -1*(maintenanceCost + operationCost)-
1000000*(cost_gas+cost_bio+cost_elec)+revenue
    end
        end
        assignin('base','case1Temp',caseTable)


    case 2
        for i = 1:num_Years
    if i <= loanpaymentPeriod
        caseTable(i,1) = -1*(annuity + maintenanceCost + operationCost)-
1000000*(cost_gas+cost_bio+cost_elec)+revenue
    else
        caseTable(i,1) = -1*(maintenanceCost + operationCost)-
1000000*(cost_gas+cost_bio+cost_elec)+revenue
    end
        end
        assignin('base','case2Temp',caseTable)

    case 3
        for i = 1:num_Years
    if i <= loanpaymentPeriod
        caseTable(i,1) = -1*(annuity + maintenanceCost + operationCost)-
1000000*(cost_gas+cost_bio+cost_elec)+revenue
    else
        caseTable(i,1) = -1*(maintenanceCost + operationCost)-
1000000*(cost_gas+cost_bio+cost_elec)+revenue
    end
        end
        assignin('base','case3Temp',caseTable)
    otherwise
```

```
end

% load data from a .txt file
% load('manualCashFlow.mat')

% input the data into uitable1
casesTable = [evalin('base','case1Temp') evalin('base','case2Temp')
evalin('base','case3Temp')]

set(handles.uitable1,'data',casesTable);

function loanpaymentInterestRate_editText_Callback(hObject, eventdata,
handles)
% hObject    handle to loanpaymentInterestRate_editText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of
loanpaymentInterestRate_editText as text
%        str2double(get(hObject,'String')) returns contents of
loanpaymentInterestRate_editText as a double
input = str2double(get(hObject,'String'));

if (isempty(input))
     set(hObject,'String','0')
end
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function loanpaymentInterestRate_editText_CreateFcn(hObject, eventdata,
handles)
% hObject    handle to loanpaymentInterestRate_editText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in case_popupmenu.
function case_popupmenu_Callback(hObject, eventdata, handles)
% hObject    handle to case_popupmenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns case_popupmenu
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
case_popupmenu
```

```
% --- Executes during object creation, after setting all properties.
function case_popupmenu_CreateFcn(hObject, eventdata, handles)
% hObject    handle to case_popupmenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in populateTable.
function populateTable_Callback(hObject, eventdata, handles)
% hObject    handle to populateTable (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% setup three arrays to store case cashflow in the workspace

case1Temp = zeros(str2double(get(handles.Years_editText,'string')),1)
case2Temp = zeros(str2double(get(handles.Years_editText,'string')),1)
case3Temp = zeros(str2double(get(handles.Years_editText,'string')),1)
assignin('base','case1Temp',case1Temp)
assignin('base','case2Temp',case2Temp)
assignin('base','case3Temp',case3Temp)
```

Appendix B Living Lab with Battery System

In Figure 58, the detailed battery function schematic is illustrated.

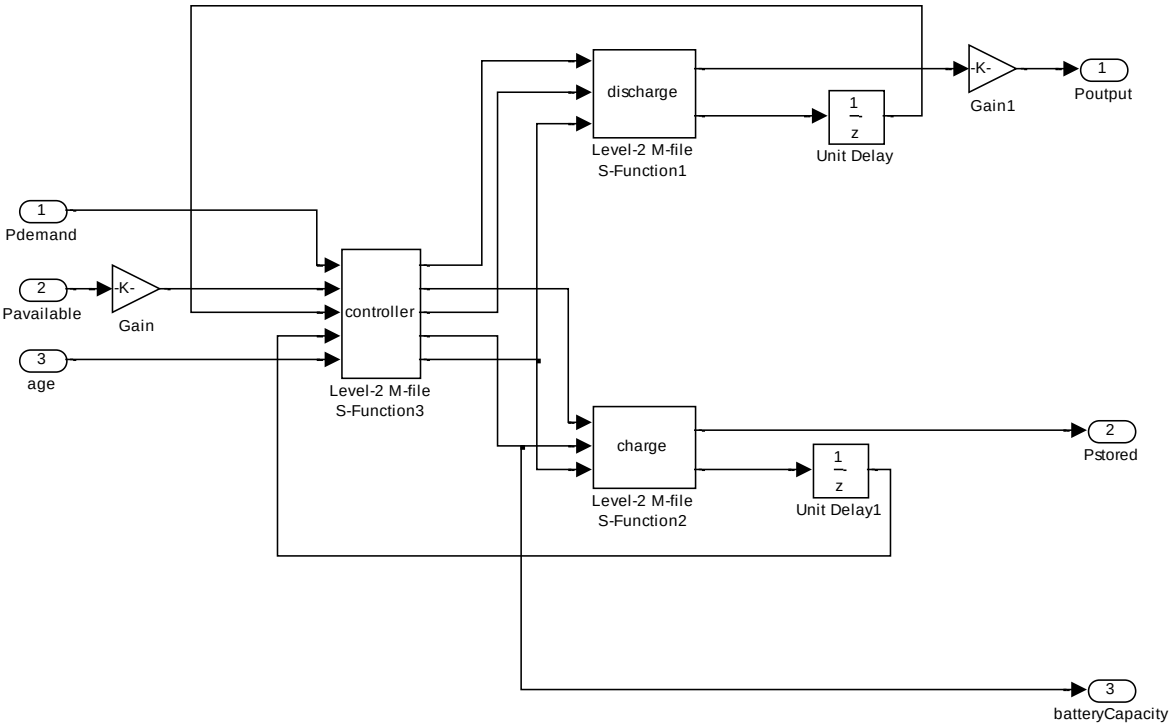


Figure 58: Battery function schematic

In Figure 59, the detailed battery system schematic is illustrated.

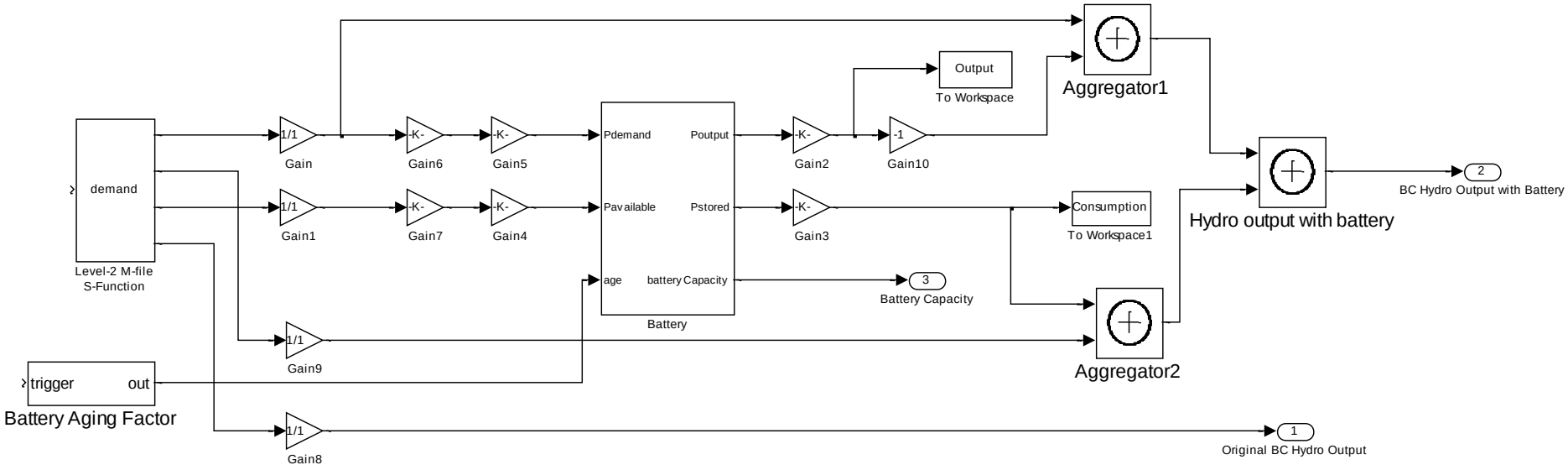


Figure 59: Battery system schematic

In Figure 60, the battery system financial model schematic is illustrated.

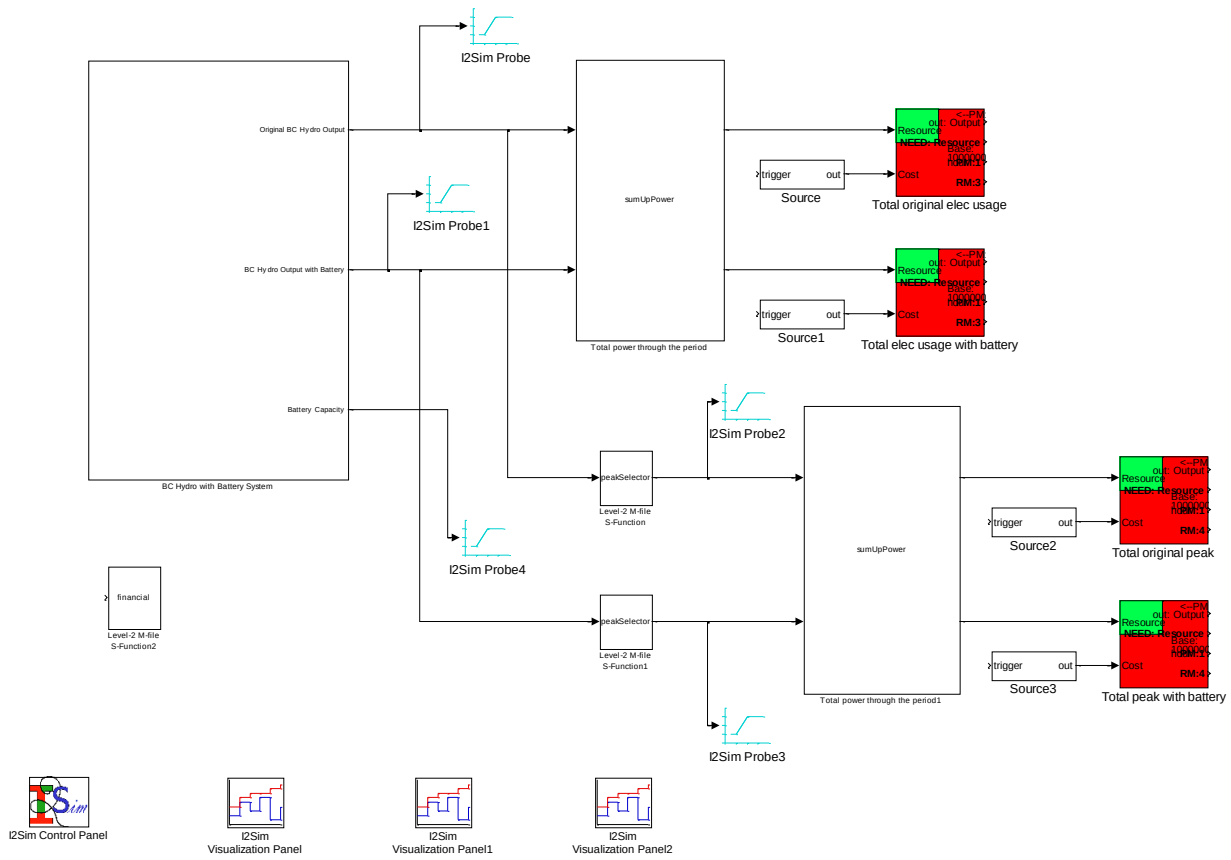


Figure 60: Battery system financial model schematic

B.1 Battery Discharge Function Matlab Code

This appendix contains the Matlab code of Battery Discharge function.

```
function discharge(block)
% Level-2 M file S-Function for times two demo.
%   Copyright 1990-2004 The MathWorks, Inc.
%   $Revision: 1.1.6.1 $
  setup(block);
%endfunction

function setup(block)

  %% Register number of input and output ports

  block.NumInputPorts  = 3;
  block.NumOutputPorts = 2;

  %% Setup functional port properties to dynamically
  %% inherited.
  block.SetPreCompInpPortInfoToDynamic;
  block.SetPreCompOutPortInfoToDynamic;

  % Allow multidimensional signals
  block.AllowSignalsWithMoreThan2D = true;

  block.InputPort(1).Dimensions        = 1; %Demand Power
  block.InputPort(1).DatatypeID  = 0;  % double
  block.InputPort(1).Complexity  = 'Real';
  block.InputPort(1).DirectFeedthrough = true;

  block.InputPort(2).Dimensions        = 1; %Current Capacity
  block.InputPort(2).DatatypeID  = 0;  % double
  block.InputPort(2).Complexity  = 'Real';
  block.InputPort(2).DirectFeedthrough = true;

  block.InputPort(3).Dimensions        = 1; %aging factor
  block.InputPort(3).DatatypeID  = 0;  % double
  block.InputPort(3).Complexity  = 'Real';
  block.InputPort(3).DirectFeedthrough = true;

  block.OutputPort(1).Dimensions       = 1; %Output Power
  block.OutputPort(1).DatatypeID  = 0; % double
  block.OutputPort(1).Complexity  = 'Real';

  block.OutputPort(2).Dimensions       = 1; %New Current Capacity
  block.OutputPort(2).DatatypeID  = 0; % double
  block.OutputPort(2).Complexity  = 'Real';


  % Register parameters
  block.NumDialogPrms     = 0;
```

```
  %% Set block sample time to inherited
  block.SampleTimes = [-1 0];

  %% Run accelerator on TLC
  block.SetAccelRunOnTLC(true);

  %% Register methods
  block.RegBlockMethod('PostPropagationSetup',    @DoPostPropSetup);
  block.RegBlockMethod('InitializeConditions',    @InitConditions);
  block.RegBlockMethod('SetInputPortSamplingMode',@SetInputPortSamplingMode);
  %block.RegBlockMethod('SetInputPortDimensions', @SetInpPortDims);
  %block.RegBlockMethod('Terminate', @Terminate);
  block.RegBlockMethod('Outputs', @Output);
%endfunction

function DoPostPropSetup(block)

  %% Setup Dwork
  block.NumDworks = 8;

  block.Dwork(1).Name = 'Capacity'; %Battery Capacity
  block.Dwork(1).Dimensions      = 1;
  block.Dwork(1).DatatypeID      = 0;
  block.Dwork(1).Complexity      = 'Real';
  block.Dwork(1).UsedAsDiscState = true;

  block.Dwork(2).Name = 'turnPoint'; %The Point where battery discharge curve
changes
  block.Dwork(2).Dimensions      = 1;
  block.Dwork(2).DatatypeID      = 0;
  block.Dwork(2).Complexity      = 'Real';
  block.Dwork(2).UsedAsDiscState = true;

  block.Dwork(3).Name = 'Demand';  %How much is power demand
  block.Dwork(3).Dimensions      = 1;
  block.Dwork(3).DatatypeID      = 0;
  block.Dwork(3).Complexity      = 'Real';
  block.Dwork(3).UsedAsDiscState = true;

  block.Dwork(4).Name = 'Voltage';  %Battery operating voltage
  block.Dwork(4).Dimensions      = 1;
  block.Dwork(4).DatatypeID      = 0;
  block.Dwork(4).Complexity      = 'Real';
  block.Dwork(4).UsedAsDiscState = true;

  block.Dwork(5).Name = 'eNominal'; %Energy stored in the Nominal region
  block.Dwork(5).Dimensions      = 1;
  block.Dwork(5).DatatypeID      = 0;
  block.Dwork(5).Complexity      = 'Real';
  block.Dwork(5).UsedAsDiscState = true;

  block.Dwork(6).Name = 'eExponential'; %Energy stored in the Exponetial
region
  block.Dwork(6).Dimensions      = 1;
```

```
  block.Dwork(6).DatatypeID      = 0;
  block.Dwork(6).Complexity      = 'Real';
  block.Dwork(6).UsedAsDiscState = true;

  block.Dwork(7).Name = 'vNominal'; %Nominal Voltage
  block.Dwork(7).Dimensions      = 1;
  block.Dwork(7).DatatypeID      = 0;
  block.Dwork(7).Complexity      = 'Real';
  block.Dwork(7).UsedAsDiscState = true;

  block.Dwork(8).Name = 'age'; %aging factor of the battery
  block.Dwork(8).Dimensions      = 1;
  block.Dwork(8).DatatypeID      = 0;
  block.Dwork(8).Complexity      = 'Real';
  block.Dwork(8).UsedAsDiscState = true;
%endfunction

function InitConditions(block)
%%
%   block.Dwork(2).Data = 25; %turnPoint
  block.Dwork(7).Data = 100; %constant voltage
  block.Dwork(1).Data = 0;

 %endfunction



%endfunction

function SetInputPortSamplingMode(block, idx, fd)
  block.InputPort(idx).SamplingMode = fd;
  for i=1:block.NumOutputPorts
    block.OutputPort(i).SamplingMode = fd;
  end
%endfunction

function Output(block)

block.Dwork(1).Data = block.InputPort(2).Data;
block.Dwork(3).Data = block.InputPort(1).Data;
block.Dwork(8).Data = block.InputPort(3).Data;

block.Dwork(2).Data = 25*block.Dwork(8).Data; %turnPoint


if 0<= block.Dwork(1).Data && block.Dwork(1).Data<= block.Dwork(2).Data

    block.Dwork(4).Data = block.Dwork(7).Data;
    block.Dwork(5).Data = block.Dwork(1).Data*block.Dwork(4).Data;
    block.Dwork(6).Data = 0;
else

    block.Dwork(4).Data = 5*(block.Dwork(1).Data)-25;
    block.Dwork(5).Data = block.Dwork(2).Data*block.Dwork(7).Data;
```

```
    block.Dwork(6).Data = (block.Dwork(1).Data-
block.Dwork(2).Data)*block.Dwork(4).Data;
end

if block.Dwork(1).Data>block.Dwork(2).Data
    if block.Dwork(3).Data*1<=block.Dwork(6).Data
        block.OutputPort(1).Data = block.Dwork(3).Data;
        block.OutputPort(2).Data = block.Dwork(1).Data-
(block.Dwork(3).Data*1/block.Dwork(4).Data);
    else
        if (block.Dwork(3).Data*1)<(block.Dwork(5).Data+block.Dwork(6).Data)
            block.OutputPort(1).Data = block.Dwork(3).Data;
            block.OutputPort(2).Data = block.Dwork(2).Data-
((block.Dwork(3).Data*1-block.Dwork(6).Data)/block.Dwork(7).Data);
        else
            block.OutputPort(1).Data =
(block.Dwork(5).Data+block.Dwork(6).Data)/1;
            block.OutputPort(2).Data = 0;
        end
    end
else
    if (block.Dwork(3).Data*1)<=(block.Dwork(5).Data+block.Dwork(6).Data)
         block.OutputPort(1).Data = block.Dwork(3).Data;
         block.OutputPort(2).Data = block.Dwork(1).Data-
(block.Dwork(3).Data*1/block.Dwork(7).Data);
    else
         block.OutputPort(1).Data =
(block.Dwork(5).Data+block.Dwork(6).Data)/1;
         block.OutputPort(2).Data = 0;
    end
end
%endfunction
```

B.2 Battery Charge Function Matlab Code

This appendix contains the Matlab code of Battery charge function.

```
function charge(block)
% Level-2 M file S-Function for times two demo.
%   Copyright 1990-2004 The MathWorks, Inc.
%   $Revision: 1.1.6.1 $
  setup(block);
%endfunction

function setup(block)

  %% Register number of input and output ports

  block.NumInputPorts  = 3;
  block.NumOutputPorts = 2;

  %% Setup functional port properties to dynamically
  %% inherited.
  block.SetPreCompInpPortInfoToDynamic;
  block.SetPreCompOutPortInfoToDynamic;

  % Allow multidimensional signals
  block.AllowSignalsWithMoreThan2D = true;

  block.InputPort(1).Dimensions        = 1; %Available Power
  block.InputPort(1).DatatypeID  = 0;  % double
  block.InputPort(1).Complexity  = 'Real';
  block.InputPort(1).DirectFeedthrough = true;

  block.InputPort(2).Dimensions        = 1; %Current Capacity
  block.InputPort(2).DatatypeID  = 0;  % double
  block.InputPort(2).Complexity  = 'Real';
  block.InputPort(2).DirectFeedthrough = true;

  block.InputPort(3).Dimensions        = 1; %aging factor
  block.InputPort(3).DatatypeID  = 0;  % double
  block.InputPort(3).Complexity  = 'Real';
  block.InputPort(3).DirectFeedthrough = true;

  block.OutputPort(1).Dimensions       = 1; %Stored Power
  block.OutputPort(1).DatatypeID  = 0; % double
  block.OutputPort(1).Complexity  = 'Real';

  block.OutputPort(2).Dimensions       = 1; %New Current Capacity
  block.OutputPort(2).DatatypeID  = 0; % double
  block.OutputPort(2).Complexity  = 'Real';


  % Register parameters
  block.NumDialogPrms     = 0;
```

```
  %% Set block sample time to inherited
  block.SampleTimes = [-1 0];

  %% Run accelerator on TLC
  block.SetAccelRunOnTLC(true);

  %% Register methods
  block.RegBlockMethod('PostPropagationSetup',    @DoPostPropSetup);
  block.RegBlockMethod('InitializeConditions',    @InitConditions);
  block.RegBlockMethod('SetInputPortSamplingMode',@SetInputPortSamplingMode);
  %block.RegBlockMethod('SetInputPortDimensions', @SetInpPortDims);
  %block.RegBlockMethod('Terminate', @Terminate);
  block.RegBlockMethod('Outputs', @Output);
%endfunction

function DoPostPropSetup(block)

  %% Setup Dwork
  block.NumDworks = 10;

  block.Dwork(1).Name = 'Capacity'; %Battery Capacity
  block.Dwork(1).Dimensions      = 1;
  block.Dwork(1).DatatypeID      = 0;
  block.Dwork(1).Complexity      = 'Real';
  block.Dwork(1).UsedAsDiscState = true;

  block.Dwork(2).Name = 'turnPoint'; %The Point where battery charge curve
changes
  block.Dwork(2).Dimensions      = 1;
  block.Dwork(2).DatatypeID      = 0;
  block.Dwork(2).Complexity      = 'Real';
  block.Dwork(2).UsedAsDiscState = true;

  block.Dwork(3).Name = 'available';  %How much power is available
  block.Dwork(3).Dimensions      = 1;
  block.Dwork(3).DatatypeID      = 0;
  block.Dwork(3).Complexity      = 'Real';
  block.Dwork(3).UsedAsDiscState = true;

  block.Dwork(4).Name = 'Voltage';  %Battery operating voltage
  block.Dwork(4).Dimensions      = 1;
  block.Dwork(4).DatatypeID      = 0;
  block.Dwork(4).Complexity      = 'Real';
  block.Dwork(4).UsedAsDiscState = true;

  block.Dwork(5).Name = 'eNominal'; %Energy stored in the Nominal region
  block.Dwork(5).Dimensions      = 1;
  block.Dwork(5).DatatypeID      = 0;
  block.Dwork(5).Complexity      = 'Real';
  block.Dwork(5).UsedAsDiscState = true;

  block.Dwork(6).Name = 'eExponential'; %Energy stored in the Exponetial
region
  block.Dwork(6).Dimensions      = 1;
```

```
  block.Dwork(6).DatatypeID      = 0;
  block.Dwork(6).Complexity      = 'Real';
  block.Dwork(6).UsedAsDiscState = true;

  block.Dwork(7).Name = 'vNominal'; %Nominal Voltage
  block.Dwork(7).Dimensions      = 1;
  block.Dwork(7).DatatypeID      = 0;
  block.Dwork(7).Complexity      = 'Real';
  block.Dwork(7).UsedAsDiscState = true;

  block.Dwork(8).Name = 'energyRoom'; %how much storage room is left
  block.Dwork(8).Dimensions      = 1;
  block.Dwork(8).DatatypeID      = 0;
  block.Dwork(8).Complexity      = 'Real';
  block.Dwork(8).UsedAsDiscState = true;

  block.Dwork(9).Name = 'capacityMax'; %Max capacity
  block.Dwork(9).Dimensions      = 1;
  block.Dwork(9).DatatypeID      = 0;
  block.Dwork(9).Complexity      = 'Real';
  block.Dwork(9).UsedAsDiscState = true;

  block.Dwork(10).Name = 'age'; %aging factor of the battery
  block.Dwork(10).Dimensions      = 1;
  block.Dwork(10).DatatypeID      = 0;
  block.Dwork(10).Complexity      = 'Real';
  block.Dwork(10).UsedAsDiscState = true;


%endfunction

function InitConditions(block)
%%
%   block.Dwork(2).Data = 25; %turnPoint
  block.Dwork(7).Data = 100; %constant voltage

%   block.Dwork(9).Data = 30; %battery capacity
  block.Dwork(1).Data = 0;
 %endfunction



%endfunction

function SetInputPortSamplingMode(block, idx, fd)
  block.InputPort(idx).SamplingMode = fd;
  for i=1:block.NumOutputPorts
    block.OutputPort(i).SamplingMode = fd;
  end
%endfunction

function Output(block)

block.Dwork(1).Data = block.InputPort(2).Data;
```

```
block.Dwork(3).Data = block.InputPort(1).Data;
block.Dwork(10).Data = block.InputPort(3).Data;

block.Dwork(2).Data = 25*block.Dwork(10).Data; %turnPoint
block.Dwork(9).Data = 30*block.Dwork(10).Data; %battery capacity

if 0<= block.Dwork(1).Data && block.Dwork(1).Data <= block.Dwork(2).Data

    block.Dwork(4).Data = block.Dwork(7).Data;
    block.Dwork(5).Data = block.Dwork(1).Data*block.Dwork(4).Data;
    block.Dwork(6).Data = 0;
else

    block.Dwork(4).Data = 5*(block.Dwork(1).Data)-25;
    block.Dwork(5).Data = block.Dwork(2).Data*block.Dwork(7).Data;
    block.Dwork(6).Data = (block.Dwork(1).Data-
block.Dwork(2).Data)*block.Dwork(4).Data;
end

block.Dwork(8).Data =
block.Dwork(7).Data*block.Dwork(2).Data+(5*(block.Dwork(9).Data)-
25)*(block.Dwork(9).Data-block.Dwork(2).Data)-block.Dwork(5).Data-
block.Dwork(6).Data;

if block.Dwork(1).Data > block.Dwork(2).Data

    if (block.Dwork(3).Data*1) < block.Dwork(8).Data
        block.OutputPort(2).Data = block.Dwork(1).Data+(-
1*block.Dwork(4).Data+sqrt(block.Dwork(4).Data*block.Dwork(4).Data+4*5*block.
Dwork(3).Data*1))/(2*5);
        block.OutputPort(1).Data = block.Dwork(3).Data;
    else
        block.OutputPort(2).Data = block.Dwork(9).Data;
        block.OutputPort(1).Data = block.Dwork(8).Data/1;
    end
else
    if block.Dwork(3).Data*1 <= (block.Dwork(7).Data*block.Dwork(2).Data-
block.Dwork(5).Data)
        block.OutputPort(2).Data =
block.Dwork(1).Data+(block.Dwork(3).Data/block.Dwork(7).Data);
        block.OutputPort(1).Data = block.Dwork(3).Data;
    else
        if (block.Dwork(3).Data*1) < block.Dwork(8).Data
            block.OutputPort(2).Data = block.Dwork(2).Data+(-
1*block.Dwork(4).Data+sqrt(block.Dwork(4).Data*block.Dwork(4).Data+4*5*(block
.Dwork(3).Data*1-block.Dwork(7).Data*(block.Dwork(2).Data-
block.Dwork(1).Data))))/(2*5);
            block.OutputPort(1).Data = block.Dwork(3).Data;
        else
            block.OutputPort(2).Data = block.Dwork(9).Data;
            block.OutputPort(1).Data = block.Dwork(8).Data/1;
        end
    end
end
%endfunction
```

B.3 Battery Controller Function Matlab Code

This appendix contains the Matlab code of Battery controller function.

```
function controller(block)
% Level-2 M file S-Function for times two demo.
%   Copyright 1990-2004 The MathWorks, Inc.
%   $Revision: 1.1.6.1 $
  setup(block);
%endfunction

function setup(block)

  %% Register number of input and output ports

  block.NumInputPorts  = 5;
  block.NumOutputPorts = 5;

  %% Setup functional port properties to dynamically
  %% inherited.
  block.SetPreCompInpPortInfoToDynamic;
  block.SetPreCompOutPortInfoToDynamic;

  % Allow multidimensional signals
  block.AllowSignalsWithMoreThan2D = true;

  block.InputPort(1).Dimensions        = 1; %Demand Power
  block.InputPort(1).DatatypeID  = 0;  % double
  block.InputPort(1).Complexity  = 'Real';
  block.InputPort(1).DirectFeedthrough = true;

  block.InputPort(2).Dimensions        = 1; %Current Capacity
  block.InputPort(2).DatatypeID  = 0;  % double
  block.InputPort(2).Complexity  = 'Real';
  block.InputPort(2).DirectFeedthrough = true;

  block.InputPort(3).Dimensions        = 1; %Demand Power
  block.InputPort(3).DatatypeID  = 0;  % double
  block.InputPort(3).Complexity  = 'Real';
  block.InputPort(3).DirectFeedthrough = true;

  block.InputPort(4).Dimensions        = 1; %Demand Power
  block.InputPort(4).DatatypeID  = 0;  % double
  block.InputPort(4).Complexity  = 'Real';
  block.InputPort(4).DirectFeedthrough = true;

  block.InputPort(5).Dimensions        = 1; %Aging factor
  block.InputPort(5).DatatypeID  = 0;  % double
  block.InputPort(5).Complexity  = 'Real';
  block.InputPort(5).DirectFeedthrough = true;

  block.OutputPort(1).Dimensions       = 1; %Output Power
  block.OutputPort(1).DatatypeID  = 0; % double
```

```
  block.OutputPort(1).Complexity  = 'Real';

  block.OutputPort(2).Dimensions       = 1; %New Current Capacity
  block.OutputPort(2).DatatypeID  = 0; % double
  block.OutputPort(2).Complexity  = 'Real';

  block.OutputPort(3).Dimensions       = 1; %Output Power
  block.OutputPort(3).DatatypeID  = 0; % double
  block.OutputPort(3).Complexity  = 'Real';

  block.OutputPort(4).Dimensions       = 1; %Output Power
  block.OutputPort(4).DatatypeID  = 0; % double
  block.OutputPort(4).Complexity  = 'Real';

  block.OutputPort(5).Dimensions       = 1; %Aging factor
  block.OutputPort(5).DatatypeID  = 0; % double
  block.OutputPort(5).Complexity  = 'Real';

  % Register parameters
  block.NumDialogPrms     = 0;

  %% Set block sample time to inherited
  block.SampleTimes = [-1 0];

  %% Run accelerator on TLC
  block.SetAccelRunOnTLC(true);

  %% Register methods
  block.RegBlockMethod('PostPropagationSetup',    @DoPostPropSetup);
  block.RegBlockMethod('InitializeConditions',    @InitConditions);
  block.RegBlockMethod('SetInputPortSamplingMode',@SetInputPortSamplingMode);
  %block.RegBlockMethod('SetInputPortDimensions', @SetInpPortDims);
  %block.RegBlockMethod('Terminate', @Terminate);
  block.RegBlockMethod('Outputs', @Output);
%endfunction

function DoPostPropSetup(block)

  %% Setup Dwork
  block.NumDworks = 1;


  block.Dwork(1).Name = 'age'; %Aging Factor
  block.Dwork(1).Dimensions      = 1;
  block.Dwork(1).DatatypeID      = 0;
  block.Dwork(1).Complexity      = 'Real';
  block.Dwork(1).UsedAsDiscState = true;



%endfunction

function InitConditions(block)
%%
```

```
  %block.Dwork(1).Data = 30; %turnPoint
 block.OutputPort(3).Data=5;
 block.OutputPort(4).Data=5;
%  block.Dwork(1).Data=30 %battery max capacity;



 %endfunction



%endfunction

function SetInputPortSamplingMode(block, idx, fd)
  block.InputPort(idx).SamplingMode = fd;
  for i=1:block.NumOutputPorts
    block.OutputPort(i).SamplingMode = fd;
  end
%endfunction

function Output(block)

block.OutputPort(1).Data=block.InputPort(1).Data;
block.OutputPort(2).Data=block.InputPort(2).Data;
block.OutputPort(5).Data=block.InputPort(5).Data;
block.Dwork(1).Data=block.InputPort(5).Data;



if block.InputPort(1).Data>0
    if (block.Dwork(1).Data*30)>block.InputPort(3).Data
 block.OutputPort(3).Data=block.InputPort(3).Data;
 block.OutputPort(4).Data=block.InputPort(3).Data;
    else
        block.OutputPort(3).Data=block.Dwork(1).Data*30;
        block.OutputPort(4).Data=block.Dwork(1).Data*30;
    end
else
    if (block.Dwork(1).Data*30)>block.InputPort(4).Data
 block.OutputPort(3).Data=block.InputPort(4).Data;
 block.OutputPort(4).Data=block.InputPort(4).Data;
    else
        block.OutputPort(3).Data=block.Dwork(1).Data*30;
        block.OutputPort(4).Data=block.Dwork(1).Data*30;
    end
end


%

%endfunction
```