Year Three Final Report

Smart Energy Management and Control for Fuel Cell Based Micro-Grid Connected Neighborhoods

Project Period: November 30, 2004 to March 15, 2006

Date of Report: March 16, 2006

Recipient: University of South Alabama

Award Number: DE-FC36-04GO14243

Subcontractor: Radiance Technologies

Cost-Sharing Partner: Radiance Technologies

Primary Contact: Dr. Mohammad S. Alam (PI/PD)
University of South Alabama
College of Engineering
ECE Department, EEB 75
Mobile, AL 36688-0002

Phone: 251-460-6117

Fax: 251-460-6028

Email: malam@usouthal.edu

DOE Technology Development Manager:

DOE Project Officer: Paul Bakke

Objectives

- To demonstrate that smart energy management of a fuel cell based micro-grid connected neighborhood can be efficient and cost-effective
- To define the most economical micro-grid configuration
- To determine how residential micro-grid connected fuel cell(s) can contribute to America's hydrogen energy future

Background

Fuel cell power generation promises to be an efficient, pollution-free, reliable power source in both large scale and small scale, remote applications. DOE formed the Solid State Energy Conversion Alliance with the intention of breaking one of the last barriers remaining for cost effective fuel cell power generation. The Alliance's goal is to produce a core solid-state fuel cell module at a cost of no more than \$400 per kilowatt and ready for commercial application by 2010. With

their inherently high, 60-70% conversion efficiencies, significantly reduced carbon dioxide emissions, and negligible emissions of other pollutants, fuel cells will be the obvious choice for a broad variety of commercial and residential applications when their cost effectiveness is improved.

In a research program funded by the Department of Energy, the research team has been investigating smart fuel cell-operated residential micro-grid communities. This research has focused on using smart control systems in conjunction with fuel cell power plants, with the goal to reduce energy consumption, reduce demand peaks and still meet the energy requirements of any household in a micro-grid community environment. In Phases I and II, a SEMaC was developed and extended to a micro-grid community. In addition, an optimal configuration was determined for a single fuel cell power plant supplying power to a ten-home micro-grid community. In Phase III, the plan is to expand this work to fuel cell based micro-grid connected neighborhoods (mini-grid). The economic implications of hydrogen cogeneration will be investigated. These efforts are consistent with DOE's mission to decentralize domestic electric power generation and to accelerate the onset of the hydrogen economy.

A major challenge facing the routine implementation and use of a fuel cell based mini-grid is the varying electrical demand of the individual micro-grids, and, therefore, analyzing these issues is vital. Efforts are needed to determine the most appropriate means of implementing micro-grids and the costs and processes involved with their extended operation.

With the development and availability of fuel cell based stand-alone power plants, an electrical mini-grid, encompassing several connected residential neighborhoods, has become a viable concept. A primary objective of this project is to define the parameters of an economically efficient fuel cell based mini-grid. Since pure hydrogen is not economically available in sufficient quantities at the present time, the use of reforming technology to produce and store excess hydrogen will also be investigated.

From a broader perspective, the factors that bear upon the feasibility of fuel cell based micro-grid connected neighborhoods are similar to those pertaining to the electrification of a small town with a localized power generating station containing several conventional generating units. In the conventional case, the town or locality would also be connected to the larger grid system of the utility company. Therefore, in the case of the fuel cell based micro-grid connected neighborhoods, this option should also be available.

Task Schedule

Task Number	Project Milestones	Percent Complete	Progress Notes
1	System Economic Analysis <ul style="list-style-type: none">• Analysis of mini-grid connected neighborhoods• Analysis of thermal energy recovery strategy• Analysis of hydrogen co-generation• Cost-optimized operation of a grid-parallel PEM fuel cell power plant• Analysis of load smoothing	100%	
2	Energy Management <ul style="list-style-type: none">• Algorithms for load smoothing• Historical load profiles simulation• Testing and data collection• Results	100%	
3	Hydrogen Co-generation <ul style="list-style-type: none">• Excess hydrogen generation• Separating the PEM fuel cell stack and the reformer• Hydrogen storage methods and delivery systems• Hydrogen purification	100%	
4	Smart Energy Management at the Appliance Level <ul style="list-style-type: none">• Appliance selection• Power control selection• Hardware development• Protocol development• Algorithm development	100%	
5	Final Technical Report	100%	

UNIVERSITY OF SOUTH ALABAMA
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Final Report

**Smart Energy Management and Control for Fuel Cell
Based Micro-Grid Connected Neighborhoods**

Submitted by:

Dr. Mohammad S. Alam (PI/PD)
University of South Alabama
College of Engineering
ECE Department, EEB 75
Mobile, AL 36688-0002

Phone: 251-460-6117
Fax: 251-460-6028

Email: malam@usouthal.edu

March 16, 2006

CONTENTS

1. System Economic Analysis

- 1.1 Analysis of mini-grid connected neighborhoods
- 1.2 Analysis of thermal energy recovery strategy
- 1.3 Analysis of hydrogen co-generation
- 1.4 Cost-optimized operation of a grid-parallel PEM fuel cell power plant
- 1.5 Analysis of load smoothing

2. Energy Management

- 2.1 Algorithms for load smoothing
- 2.2 Historical load profiles simulation
- 2.3 Testing and data collection
- 2.4 Results

3. Hydrogen Co-generation

- 3.1 Excess hydrogen generation
- 3.2 Separating the PEM fuel cell stack and the reformer
- 3.3 Hydrogen storage methods and delivery systems
- 3.4 Hydrogen purification

4. Smart Energy Management at the Appliance Level

- 4.1 Appliance selection
- 4.2 Power control selection
- 4.3 Hardware development
- 4.4 Protocol development
- 4.5 Algorithm development

Publications

References

Appendices

1. System Economic Analysis

1.1 Analysis of Mini-Grid Connected Neighborhoods

The mini-grid consists of 5 neighborhoods and each neighborhood contains a cluster of 10 identical single storied homes. Neighborhoods are fed from proton exchange membrane (PEM) fuel cell power plants (FCPP) with a total capacity of 250 KW. The layout of the neighborhoods is the key issues for the design of the mini-grid. In this project neighborhoods are supplied by a single 250 kW PEM FCPP, which is located centrally with the feeders aligned in a radial fashion. The distribution design can be divided into primary and secondary distribution systems. The feeder system between the FCPP and the distribution transformers located in the neighborhoods constitutes the primary distribution system. The secondary distribution system relates to the feeder system between the distribution transformer and the homes in the neighborhood. The layout of the neighborhoods remains unchanged as in Phase-II. The essential factors governing the mini-grid design are:

1. The length of feeders from the FCPP to the neighborhoods.
2. Feeder ampacities.
3. Voltage drops in the feeders & transformers.
4. Power loss in the feeder system.

The assumptions made in Phase-II for the design and layout of neighborhoods stays unchanged. Further assumptions are listed below:

- 1) The designer has complete freedom to decide on the location of neighborhoods for the mini-grid community.
- 2) All five neighborhoods are identical and each and every neighborhood has an IEEE-RTS load profile with a peak of 50 kW [1] to simulate the hourly electrical load profile of the community.
- 3) Hot water usage and space heating rates for the winter season in the region of Atlanta, Georgia [2] is considered to represent the thermal load profile. The summer and spring/fall thermal load data are estimated from the available winter data.
- 4) The thermal load is used along with the electrical load profile to simulate total hourly load profile of the mini-grid community.
- 5) The electric power to each neighborhood is regulated and controlled electronically through the 'Local Energy Management System' (LEMSYS), the 'Micro-grid Energy Management System' (MEMSYS) and 'Neighborhood Energy Management System' (NEMSYS). These controls will, in most instances, maintain a 50 kW power ceiling for each neighborhood. NEMSYS is **not** part of the mini-grid design.
- 6) The design will be based on the National Electric Code (2002) [3], and reference will be made to the Articles and Sections of the code throughout the design. Other texts and publications are referenced in the design as well.

1.1.1 Layout of the mini-grid

The layout design is based principally on locating the PEM FCPP in a centralized manner with respect to the neighborhoods. The centralized placement of the FCPP is critical because it determines the length of the feeder from the FCPP and/or distribution transformer to the farthest end-user. The length of feeder and/or tie-line, and the power carried by them as well as voltage drop in distribution transformers determine the total voltage drop and power loss in the system. The sizing or ampacity of the feeder however, is determined in accordance with the NEC code. Fig. 1.1.1 shows the layout of the mini-grid.

Fig.1.1.1 Mini-grid community layout.

Each neighborhood is located on its own 'neighborhood island' surrounded by roads on all four sides as shown in Fig.1.1.1. The mini-grid community is composed of 5 neighborhood islands each of width 220 ft. and 400 ft long. Each neighborhood receives power from its own single-phase distribution transformer with a nominal rating of 2400-120/240 V and 75 kVA.

1.1.2 Primary distribution design of the mini-grid

Fig.1.1.2 represents the simplified single-line diagram of the feeder system, while Fig. 1.1.3 shows the details of the feeder layout.

Fig. 1.1.2 Single line diagram of the mini-grid feeder system

Fig. 1.1.3 Single-phase drawing of the mini-grid feeder system

The design of the feeder is based on the load profiles and assumptions described above. The management schemes will individually maintain a ceiling of 50 kW load demand for each neighborhood using energy management algorithms LEMSYS, MEMSYS and NEMSYS developed in Phases I, II and III.

Considering the mini-grid as shown in Fig. 1.1.3, one might consider the extreme case of two neighborhoods (1 & 2 or 4 & 5); each has a distribution transformer of 75 kVA, drawing the total allowable 150 kW of power. In such a case the feeder supplying the two neighborhoods should be able to carry 150 kW and have a voltage drop within the limits specified by ANSI, and a loss that is acceptable from the FCPP operational strategy

point of view (LEMSYS, MEMSYS and NEMSYS). ANSI Standard C84 specifies that for low voltage systems, the actual service voltage must lie within $\pm 5\%$ of nominal voltage for continuous operation. The dc bus voltage of the 250 kW FCPP can vary from 600 V DC to 900 V DC [4]. Because of that, no step-up transformers are used just after the FCPP, instead a DC/AC inverter is used to raise the voltage from 900 V dc to 2400 V ac. This implies that power from the FCPP will directly be supplied through an inverter at 2400 V level to 75 kVA distribution transformer with a rated voltage of 2400-240/120V. The step down distribution transformer with a 240V (nominal) secondary winding must have a center tap to allow for the grounded neutral and the two 'outer' lines for the 120V (nominal) supplies [5]. Taps will be provided on the secondary side as shown in Fig. 1.1.4, to allow for the adjustment of the supply voltage to the end-users. The feeder design inside the neighborhood remains same as in Phase II.

Fig. 1.1.4 75 kVA, transformer with secondary taps ($\%r = 0.8$, $\%x = 1.2$) for single-unit configuration.

1) Feeder Length

The longest length of the feeder plus lateral is taken as 465 ft. This length includes the path from the FCPP to the remotest distribution transformer.

2) Sizing of Feeders

Since the maximum power to be carried is 150 kW in the extreme case, the feeders must be sized accordingly. The next issue to be addressed is the type of cable appropriate for this application. Since the design voltage level is taken to be 2.4 kV, Article 310.60 of [3] suggests that MV type cable should be used.

To determine the size of the cables the current loading must be known. As indicated earlier, the worst case would be when a feeder, which is 465 ft long, is allowed to draw the full 150 kW at 2400V. Assuming a power factor of 0.95 lagging, the current is, $150000 / (2400 \times 0.95) = 65.79\text{A}$. This current will be

considered *continuous*. Hence according to Article 310.15 of [3], the design load current will be $1.25 \times 65.79 = 82.24$ A. The minimum over current device (service entrance breaker) rating must be 90A. This is in accordance with Article 240.6 of [3]. Article 310.60 of [3] gives the selection of conductors rated at medium voltage, ranging from 2001 to 35000 volts. According to this article, ampacities for conductors rated at medium voltage level should be selected from Tables 310.67 through 310.86. Since our focus is on single-insulated copper conductors directly buried in earth, the cable must be taken from Table 310.81 of [3] from column marked 90°C. The minimum conductor size of Type MV-90, from the table is #8 AWG having a maximum allowable current rating of 110A. In this design, full load current is 65.79 A, which is expected to produce a conductor temperature much less than 60 °C. This is because of the fact that power loss is proportional to square of the current, $P \propto I^2$ and conductor temperature is proportional to power loss, $T \propto P$. Thus conductor temperature is proportional to square of the current, $T \propto I^2$. This design also satisfies the requirement of Article 110.14(C)(1)(a) of [3], which indicates that if the design load current is *less* than 100A, the ampacity of conductors with higher than 60 °C temperature ratings should not go beyond the ampacity level determined by such conductors. So the outcome of the design so far determines the type and size of the feeder to be two single-insulated conductors of Type MV-90, #8 AWG copper cables, directly buried.

3) Voltage drop in the feeders

When calculating the voltage drop, the impedance of the 75 kVA transformer as well as the feeder must be taken into account. Reference [2] indicates that a 75 kVA distribution transformer has a percentage resistance and reactance of %r = 0.8, %x = 1.2, which yields %1.134 of voltage drop.

Actual impedance and reactance values in ohms are:

$$R_{TR} = \frac{1}{100} \left[\frac{(0.8\%) \times (2400 \text{ volts})^2}{75000 \text{ VA}} \right] = 0.6144 \Omega$$

$$X_{TR} = \frac{1}{100} \left[\frac{(1.2\%) \times (2400 \text{ volts})^2}{75000 \text{ VA}} \right] = 0.9216 \Omega,$$

Current flow on the primary side is obtained as follows:

$$S = V \times I \Rightarrow I = \frac{75000}{2400} = 31.25 \text{ A}$$

The total voltage drop across the transformer is:

$$v_D = 31.25 \times [0.6144 \times \cos(-18.2) - 0.9216 \times \sin(-18.2)] = 27.23 \text{ V}$$

and percentage voltage drop across the transformer is:

$$V_{DTr} \% = \frac{27.23}{2400} \times 100 = 1.134$$

The cable resistance and reactance per 1000 ft. is obtained from Table 9, Chapter 9 of [3]. The resistance is obtained from the column entitled ‘Alternating-Current Resistance for Uncoated Copper Wires’ and with the subheading ‘PVC Conduit’. The reactance is obtained from the column entitled ‘ X_L (Reactance) for All Wires.’ The values for #8 AWG are, Resistance = 0.78 ohm/1000 ft. and Reactance = 0.052 ohm/1000 ft. Thus, for a total length of 465 ft as shown in Fig. 1.1.5 we have,

$$R_{Feeder} = 2 \times 0.78 \times \frac{465}{1000} = 0.7254 \text{ ohm and } X_{Feeder} = 2 \times 0.052 \times \frac{465}{1000} = 0.04836 \text{ ohm}$$

The voltage drop in the 465 ft. feeder line can be calculated:

$$V_D = 65.79 \times [0.7254 \times \cos(-18.2) - 0.04836 \times \sin(-18.2)] = 46.32V,$$

and the percent voltage drop is:

$$V_{DLine} \% = \frac{46.32}{2400} \times 100 = 1.93 \text{ (Percent voltage drop in the line)}$$

Fig. 1.1.5 Single line diagram showing the FCPP and the feeder.

Now the total voltage drop is the summation of line and transformer voltage drop, which is $V_{Drop} = \%1.134V + \%1.93V = \%3.064V$. The magnitude of the drop is 73.55V in a 2400V system. This drop is well within the allowable $\pm 5\%$ drop as specified by ANSI Standard C84.

4) Power loss in the feeder system and transformer

The total power loss in the feeder system on both side of the FCPP is, collectively, $P_{Feeder} = 2 \times 0.7254 \times 65.79^2 = 6280$ watts. Assuming a transformer

efficiency of 95% for a loading of 50 kW, the total loss in the transformers is determined as,

$$5 \times 50000 \cdot \left[\frac{1 - 0.95}{0.95} \right] = 13160 \text{ watts}$$

1.2 Analysis of Thermal Energy Recovery Strategy

Two scenarios were evaluated to analyze the commercial implications of thermal energy recovery. In **Case 1**, a community consisting of five clusters of ten family homes was considered. Each cluster of homes is served by a 50kW proton exchange membrane (PEM) type fuel cell. The excess heat produced by fuel processing is recovered by heating water for domestic use for an overall system efficiency of approximately 60%. In **Case 2**, the same size community is served by a hybrid system consisting of a 175kW Solid-oxide fuel cell (SOFC) and a 75kW PEM. Heat energy is recovered from the SOFC and is used to process fuel for the PEM system, resulting in an overall system efficiency of approximately 61%.

1.2.1 Case 1 - a community consisting of five clusters of ten family homes. Each cluster of ten homes is served by a 50kW PEM fuel cell located in the center of the cluster.

The University of South Alabama has an operational 5kW Proton Exchange Membrane (PEM) type fuel cell manufactured by Plug Power Corporation of Latham, NY. The GenSys 5C fuel cell system incorporates a natural gas reformer that operates as methane to hydrogen converter. The flow rate of natural gas to the unit is approximately 1.5 ft³/min for the production of 5 kW of electrical power. The flow of hydrogen to the fuel cell stack is approximately 3.0 ft³/min. An 88-cell membrane electrode assembly, also called a cell stack, provides power at 48 VDC to an AC inverter at approximately 25% electrical efficiency. The output to the grid and to connected loads is 120 VAC. The unit is grid parallel, but is capable of stand-alone operation for short periods of time.

One of the advantages of a PEM fuel cell is the relatively low operating temperature of the cell stack. The hydrogen stream from the fuel reformer is humidified at the stack anode at approximately 67°C. The cell exhaust temperature is approximately 69°C, so the heat directly available from the fuel cell stack is minimal. The partial oxidation reactor in the primary stage of the steam reformer, however, operates at about 700°C. The final preferential oxidation reactor to remove carbon monoxide is operated at approximately 90°C. The GenSys 5C utilizes an ethylene glycol liquid cooling system to manage heat transfer in the system. There is a liquid-to-liquid heat exchanger that is available for customer use for heat reclamation. If the customer chooses not to utilize this system, the waste heat is rejected to the atmosphere by means of a large capacity automotive type radiator.

At full rated power of 5 kW, approximately 8kW of thermal energy is produced by the GenSys 5CS. The combined electrical and thermal energy available for use is approximately 13kW. The total amount of thermal energy available from the methane

feed is approximately 20kW, indicating an overall combined heat and power system efficiency of approximately 65%. The balance of the thermal energy is required to keep the reformer components at an elevated temperature for the steam reforming reactions to proceed.

Reported values for energy generation for larger fuel cell systems vary widely, but as a first approximation, the assumption can be made that a 50kW fuel cell would be capable of operating at a minimum combined electrical and thermal efficiency of 60%. This would indicate that a minimum of 20kW of thermal energy would be available when the fuel cell is operating at maximum output. The maximum temperature of the cooling system is approximately 80°C, which rules out the possibility of reclaiming heat energy with steam turbines or thermodynamic engines. The temperature is simply not high enough. In the Southeastern US, home heating requirements are modest and home heating is necessary for a relatively small fraction of the year. Therefore, the focus is on reclaiming and utilizing excess energy to heat water for domestic use.

Various sources for average hot water consumption in the US were consulted [6]. There were regional differences, but in the Southeastern US, a consumption of 150 gal/day for a family of four is a reasonable estimate. This would allow for showers for four people (25 gal each), a load of laundry (25 gallons) and a load of dishes in the dishwasher (25 gallons). Domestic hot water is typically supplied at a tank temperature of 60°C. In the Southeast, a typical water supply temperature is 15°C. Assuming 24-hour operation of the fuel cell at full rated power, 480 kWh, or 1.64×10^6 BTU would be available for water heating. This would be sufficient to raise the temperature of 1738 gallons of water from 15°C to 60°C. For a 10 family micro-grid community, 1500 gallons per day would be required. It is feasible to utilize the excess thermal energy from the fuel cell to heat domestic water, provided an economical means of delivery can be devised.

For a micro-grid community of ten homes arranged as shown in Figure 1.2.1, it is proposed that two circulation loops be used to continuously circulate hot water from the fuel cell reformer cooling system to individual houses.

The circulation piping and pumps must be sized such that the delivery pressure to individual houses is greater than that required by local building codes at the maximum total household consumption rate. It is recommended that two loops be employed to reduce the pressure and temperature drop incurred by the mechanics of continuous water circulation.

Hot water is pumped to the houses at supply pressure and returns through a pressure reducer, providing continuous circulation. The pressure differential allows hot water to circulate through the resident loop without the need for a pump. As shown in Figure 1.2.1, the length of the supply line and the return line are estimated to be 135 ft. The distance from the supply and return line to each of the houses is estimated to be 60 ft. The distance between the pressure and return line will be neglected since they will be assumed to be buried in the same trench upon installation.

Figure 1.2.2 shows the layout of the primary storage and distribution system.

Figure 1.2.1 - Proposed community hot water distribution schematic.

Hot water from the fuel cell heat exchanger exit is directed into two storage tanks, each providing storage for five residences. A pump capable of supplying the required flow rate and pressure is attached to each tank. The return line from each half of the grid enters a mixing valve where make-up water is added to replace the volume of water removed from the system by the residents. The water is then pumped through a radiator, which allows the water to be cooled, if necessary, to maintain the proper supply temperature at the inlet of the fuel cell heat exchanger. A connection from the central storage tank to the low-pressure line provides a closed loop system, allowing the system to be full at all times. A bypass line connecting the two systems is added for maintenance purposes. If the supply pump on one side of the system should need maintenance, the other pump can be routed to supply both sides with hot water.

Figure 1.2.3 shows a proposed residential delivery system. For each residence, there is a local circulation tank that is maintained at a water temperature of approximately 80°C by a local circulating pump. The circulation tank is used to supply hot water to the residence at approximately 60°C. The water from the primary loop is mixed with cold water from the residence to reduce the temperature to that required for household use. If the circulation tank temperature cannot be maintained at the required delivery temperature, electrical heating elements are used for auxiliary heating.

1.2.1.1 System design considerations

- **Peak hot water demand**

The system must be adequately sized to meet peak demand for all residences. The peak demand of the system is assumed to take place at approximately 7:00 AM and 7:00 PM. The peak demand flow rate is approximately 50 GPM, assuming that two showers in each house are in use for 30 minutes, using a 2.5 GPM showerhead. Each of the two subsystems supplying five homes must be capable of supporting this peak demand.

- **Low hot water demand**

A sufficient circulation must be maintained through the fuel cell to provide adequate cooling. The fuel cell heat exchanger exit temperature is assumed to be a constant 80°C. Using a water temperature of 15°C entering the fuel cell heat exchanger, the mass flow rate through the heat exchanger can be determined [11]:

Figure 1.2.2 – Primary storage and distribution system plumbing detail.

$$\begin{aligned}
 Q &= \dot{m} C_p \Delta T \\
 \dot{m} &= \frac{Q}{C_p \Delta T} \\
 \dot{m} &= \frac{40kW}{(4180J / kgK)(80-15)^{\circ}C} \\
 \dot{m} &= 0.147kg / s = 2.3 GPM
 \end{aligned}
 \tag{1.2.1}$$

Water entering the exchanger at a temperature of 15°C would require a minimum flow rate of 2.3 GPM to cool the fuel cell.

- **Hot water storage**

The central storage tanks should be large enough to hold the amount of hot water needed for peak demand. From the calculations, 750 gallons in each subsystem will be needed for times of high demand. Since each house will contain a small storage tank, estimated at 40 gallons, the central storage tank can be reduced in size. For five houses, a total of

200 gallons can be subtracted from the size of the central storage tank, giving total of 550 gallons for each central tank.

- **System pressure drop**

It is desirable to keep the system head loss due to valves, fittings, and lines less than 5 psi. The estimated pressure loss through the system was determined using equations for pressure drop through a pipe. The total head loss through a pipe is the head loss due to elevation plus head loss due to friction. Head loss due to friction through a pipe is given by:

$$\Delta h = f \left(\frac{L}{D} \right) \left(\frac{V^2}{2g} \right) \quad (1.2.2)$$

Where: f = friction coefficient
 L = length (ft)
 D = diameter (ft)
 V = velocity (ft/s)
 g = acceleration of gravity (ft/s²)

Head loss due to friction through valves and fitting is given by:

$$\Delta h = k \left(\frac{V^2}{2g} \right) \quad (1.2.3)$$

Where: k = Loss Coefficient for fittings and valves

Combining both equations gives:

$$\Delta h = \left(\frac{V^2}{2g} \right) \left(f \left[\frac{L}{D} \right] + k \right) \quad (1.2.4)$$

After iterative head loss calculations, a pipe diameter of 2 inches was determined to be suitable for use in the supply and return line [6]. A pipe diameter of 0.75 inches was calculated for connection between the house and the supply and return pipes.

Head loss through the entire supply line, 135 ft of 2-inch pipe: $\Delta h = 1.94$ ft

Head loss through the longest resident line, 60 ft of 3/4" pipe: $\Delta h = 4.73$ ft

Total maximum head loss from pump to resident: $\Delta h = 6.67$ ft, or 2.9 psi.

- **System thermal losses**

The temperature drop through the supply loop is given by:

$$\frac{q}{L} = \frac{T_h - T_c}{R1 + R2 + R3 + R4} \quad (1.2.5)$$

Where: q = heat transfer
 L = length
 T_h = temp. of fluid in pipe
 T_c = temp. of surroundings
 R = resistance (conduction or convection)

Figure 1.2.4 illustrates the heat loss mechanism through the wall of a pipe.

Where: $R1$ = inner pipe radius
 $R2$ = outer pipe radius
 $R3 = R2 + \text{insulation thickness (1.5 inch)}$
 $R4 = R3 + \text{soil insulation}$

Figure 1.2.4 – Heat loss through the wall of a pipe.

The heat loss per unit length was calculated and combined with the mass flow at to determine the maximum temperature drop with the relationship:

$$q = \dot{m}C_p\Delta T \quad (1.2.6)$$

Where: $\dot{m}$ = mass flow rate of the water
 C_p = specific heat of the water
 ΔT = change in temperature though the pipe

The temperature loss was calculated at a flow of 2.3 GPM, since the heat loss is assumed to be greatest at the lowest flow rate. The results were as follows:

Thermal loss through the entire supply line, 135 ft of 2-inch PVC with 1.5-inch insulation

$$\Delta T = 1^\circ\text{C} \quad (1.2.7)$$

Thermal loss through the entire supply line, 135 ft of 2-inch PVC, no insulation

$$\Delta T = 2.4\text{ }^{\circ}\text{C} \quad (1.2.8)$$

Head loss through the longest resident line 60 ft of 3/4-inch PVC, 1.5-inch insulation

$$\Delta T = 0.28\text{ }^{\circ}\text{C} \quad (1.2.9)$$

Head loss through the longest resident line 60 ft of 3/4-inch PVC, no insulation

$$\Delta T = 0.85\text{ }^{\circ}\text{C} \quad (1.2.10)$$

- **Summary of design criteria**

- Peak hot water demand 50gpm
- Low hot water demand 2.3 gpm
- Hot water storage capacity 550 gal
- System pressure drop 2.9 psi
- System thermal losses <2 °C, for 1.5-inch insulation

1.2.1.2 System components

Based on the design criteria, suitable off-the-shelf components were selected. The components are shown in Figure 1.2.2 and Figure 1.2.3.

- **Supply pump**

1.5hp 4-stage, providing approximately 75 psi at low flow demand and approximately 45 psi at high flow demand. The pump is controlled by a variable speed controller to give a constant pressure over the required flow rate range.

- **Make-up water pump**

1/6 hp single stage. Adds makeup water to system to replace resident use.

- **Circulation pump**

1/6 hp single stage. Circulates water through the radiator and fuel cell heat exchanger and into the central storage tanks. Pump is controlled by a thermocouple at the exit of the fuel cell heat exchanger to regulate temperature.

- **Radiator**

Sized for an inlet temperature of 75°C and an outlet temperature from 15-30°C. An average cooling capacity of approximately 25kW will be required.

1.2.1.3 Economic analysis of system

A valid economic assessment of the proposed cost of a thermal recovery system is difficult. It is relatively straightforward to determine the capital costs associated with

hardware such as pumps, piping, and fittings, but a large fraction of the cost is installation specific. A cost analysis of a representative system in the Southeastern US is detailed in Table 1.2.1. The estimated cost is for a hot water circulation system serving 10 homes in a micro-grid neighborhood supplied by a 50kW PEM fuel cell. The cost analysis does not include the capital, installation, and maintenance costs of the fuel cell system.

Table 1.2.1 – Cost analysis of a proposed thermal energy recovery system.

Hardware components

Primary storage and distribution system	
Component	Cost
550 Gallon Tank x 2	\$8,000
Pressure Pump x 2	\$2,000
Pump Controller x 2	\$5,000
Make-up Water Pump	\$400
Radiator Pump	\$400
Radiator	\$3,000
Piping	\$1,500
Pressure Reducer x 2	\$1,100
Valves	\$1,000
Instrumentation	\$5,000
Total System Cost	\$27,400

Residential delivery system, each home	
Component	Cost
40 Gallon Storage Tank w/ heater	\$150
Check Valve	\$100
Mixing Valve	\$100
Piping (3/4 in. PVC)	\$250
Total System Cost	\$600

Capital cost for 10-home installation	\$33,400
Installation cost	\$50,000
Maintenance cost, based on 10% of capital cost per year	\$3,340
Total cost for 10-home installation	\$86,740

The total installed cost for ten homes includes the first year of maintenance in advance. The total cost per home is approximately \$8700.

1.2.2 Case 2 - a community consisting of five clusters of ten family homes. The entire community is served by a centrally-located 250kW fuel cell system.

In the current stand-alone power generation market, PEM fuel cells larger than about 75kW in capacity are uncommon. There are several types of fuel cells in use for high power and intermediate power generation. One of the most popular is the solid-oxide fuel cell (SOFC), typically operating at temperatures in the range of 900-1000°C. A SOFC is arguable the simplest fuel cell system, since it only uses gas and solid phases. Ion transport is done by migration of oxygen ions through a solid ceramic electrolyte rather than migration of hydrogen ions through a permeable membrane. As a result, a SOFC can utilize either hydrogen or carbon monoxide as fuel, and is immune to carbon monoxide poisoning, unlike PEM systems.

There are a number of advantages to operating fuel cells at higher temperatures [9]:

- The reactions proceed more rapidly without the need for precious metal catalysts.
- The exit gases are at high temperature and can be used to drive turbines and compressors.
- There is no need for a separate fuel reformer due to the high operating temperature.

One main disadvantage is that fuel utilization is low compared to PEM fuel cells. This means that a significant fraction of the hydrogen supplied to the fuel cell is not converted to electricity. One way to alleviate the problem is to operate the cell under pressure, which raises the partial pressure of the hydrogen fuel, and thus the current density and fuel utilization.

Recovery of heat energy from a SOFC can be done by using a gas turbine, particularly if the system is operated under pressure. The gas turbine is then used to generate electricity in tandem with the fuel cell, resulting in a higher overall system efficiency [7]. A novel approach that combines an SOFC and a PEM system is proposed by [10]. A variation of this hybrid system is shown in Figure 1.2.5, and is the proposed system for use in the Case 2 mini-grid community.

A 175kW SOFC is operated at an elevated inlet pressure and a low fuel utilization. This allows the selection of a smaller cell stack for the rated power, since the system will be operated fuel rich.

Figure 1.2.5 – Hybrid SOFC/PEM fuel cell system

As previously mentioned, the SOFC acts as a fuel processor in addition to being a fuel cell. Methane and air are the feed gases, which undergo the oxygenolysis reaction.

This reaction is endothermic, requiring that a hot ($>500^\circ\text{C}$) nickel catalyst be present for rapid equilibrium. A disadvantage is that both steam and energy must be supplied. The energy typically comes from the combustion of a portion of the feed methane. An alternative process for hydrogen formation is exothermic and is known as partial oxidation.

The tradeoff is that less hydrogen is produced per mole of methane. In practice, either reaction produces roughly the same amount of hydrogen for the same methane feed. In the proposed tandem system, the oxygenolysis reaction is actually preferred, since there is some steam in the reformat exiting the SOFC.

The hot exhaust gases from the SOFC are used to drive a gas turbine, which drives compressors for the fuel and air supplied to the SOFC. The exhaust gases are cooled in the process. The exhaust gases from the SOFC are composed of a mixture of CH_4 , H_2 , CO_2 , CO , and H_2O as steam. This mixture goes to a shift reactor where most of the CO is converted to CO_2 and H_2 by the water-gas shift reaction at a temperature of $200\text{--}400^\circ\text{C}$

A heat exchanger downstream of the shift reactor cools the gas stream to approximately 120°C , where two reactions, selective oxidation and methanization occur to further reduce the CO content of the stream.

The temperature is controlled to keep the consumption of hydrogen to a minimum; however the concentration of CO at the inlet of the selective oxidation reactor is relatively low. The composition of the gas exiting the selective oxidation reactor is rich in hydrogen, CO_2 , and steam and is low in CO , which is an excellent fuel for the 75kW PEM fuel cell.

The exhaust from the PEM fuel cell typically still contains a small amount of hydrogen and unburned methane. The exhaust is routed into an afterburner, where the energy is reclaimed for use in heating the SOFC.

Good thermal management is the key to efficient operation of the design. A representative energy balance for the system is shown in Table 1.2.2.

Table 1.2.2 – Summary of output powers for proposed hybrid system [10].

SOFC stack power	184.65 kW
PEM stack power	73.35 kW
Turbine power	50.15 kW
Compressor power	-50.4 kW
Net power output	257.75 kW
Electrical output	244.85 kW
Overall efficiency (%) LHV	61

The overall system efficiency is based on the low heating value (LHV) of the hydrogen produced, -241.83 kJ/mol, which is the energy available from the hydrogen produced by the methane reforming process, assuming that the end product is water vapor.

This efficiency is comparable to the theoretical efficiency achieved by the combined heat and power system proposed in Case 1. There is a distinct advantage in that all of the heat energy produced by the fuel cell system is consumed by the fuel cell system itself. There is no need for conversion to water heating or problems with hot water distribution, both of which would be logistically more difficult with a large centrally-located fuel cell.

1.3 Analysis of Hydrogen Co-generation

It has been determined that the average daily power consumption of a typical residence is actually less than 5 kW [8]. There are periods during which loads reach or exceed 5kW, but they are typically confined to relatively brief periods of the day. In addition, it has been determined that some of the highest electrical loads are due to heating, ventilation, and air conditioning (HVAC). In the Southeast, there are seasonal heating and cooling variations during which the HVAC loads on the residences are relatively low.

Operational experience with the 5kW fuel cell indicates that a methane reformer operates most efficiently when in a steady state. The response of a reformer to step changes in electrical load is sluggish, due to the reaction kinetics of the multi-stage steam reforming reaction and the thermal load of the vessels in the reforming system. On the other hand, the fuel cell stack responds rapidly to step changes in current demand, with time constants in the millisecond range. If the PEM cell stack and reformer were separate systems, the issues of maintenance and thermal recovery would be less complicated. In times when the fuel cell is not operating at full load, the reformer could continue to

produce the maximum amount of hydrogen. The cell stack would use hydrogen on electrical demand, while the reformer would continue to independently produce hydrogen at a constant rate.

1.3.1 Distributed generation - Five independent 50kW PEM systems

Each 50kW fuel cell system described in Section 1.2.1 can be used to produce reformat during times when the electrical power consumption of any 10-home community is below the maximum of 50kW. Assuming that the reformer can be operated independently of the cell stack, reformat could be compressed and stored in a central location for later use by any of the five 50kW fuel cells.

The hydrogen consumption of a 50kW PEM fuel cell is approximately 20ft³/min at full capacity. If a 50kW fuel cell operates at an average capacity of 60%, then 8ft³/min of hydrogen could be stored for use. An economic analysis of this approach is detailed in Section 1.4.1.2.

Another possibility for using the excess hydrogen is to fuel hydrogen powered automobiles. A 50kW fuel cell at 60% utilization would provide approximately 15kg of hydrogen per day for automotive use. There are some problems associated with this scenario, however.

- One vehicle would consume approximately 10kg/day, based on figures for typical consumption from GM and BMW [9].
- The hydrogen would have to be purified.
- The hydrogen would have to be compressed or liquefied for delivery.
- Some infrastructure would have to be built for fueling automobiles with high pressure or liquid hydrogen.
- Fuel cell-based or hydrogen combustion-based automobiles would have to achieve sufficient market penetration to be relatively common.

Based on the information presented above and in Section 1.4.1.2, the most economical use of excess hydrogen would be storage under low pressure for use in the fuel cell.

1.3.2 Central system - One 250kW hybrid fuel cell system

The system described in Section 1.2.2 consists of a 175kW solid oxide fuel cell (SOFC) used as a fuel processor for a 75kW PEM fuel cell. The SOFC fuel cell system directly reforms methane as part of the power generation process. As a result, it is impossible to separate the reformer from the fuel cell system.

The effluent from the SOFC would be suitable for storage, however, since the SOFC is operated inefficiently by design as a fuel processor. In fact, the 75kW PEM fuel cell system could be started up or shut down very rapidly because it is operated directly from reformat from the SOFC. One advantage is that during times when the electrical demand is low, the PEM cell could be held in reserve and hydrogen-rich reformat could

be compressed and stored. The PEM cell could then be brought on line for peak demand or as primary load support for the SOFC during scheduled maintenance.

Use of excess hydrogen to fuel automobiles would face the same challenges as the distributed fuel cell system case. The most economical scenario for excess hydrogen utilization would be storage at low pressure.

1.4 Cost-Optimized Operation of a Grid-Parallel PEM Fuel Cell Power Plant

Fuel cells as a renewable energy source, is considered as one of the most promising sources of electrical energy that can meet the increasing demand in electrical energy and environmental constraints. Fuel cells are not only distinguished by higher efficiency than conventional power plants, but are also environmentally clean, and have extremely low emission of oxides of nitrogen and sulfur. Due to low working temperature (80-100°C) and fast start up, PEM fuel cells power plants (FCPP) are the best candidates for residential and vehicular applications.

Fuel cell economics and economical aspects have been presented in the literature [12]-[16]. In [12] and [13] a cost-based approach has been introduced to estimate the optimal output power from the FCPP while satisfying system operational constraints. The approach only considers the possibility of selling and buying energy from the local grid, and the optimal use of thermal power output from the FCPP.

In this research the cost-based approach in [12] and [13] has been extended to include the hydrogen production/storage while introducing management strategies for optimal utilization of both hydrogen and recovered thermal energy. Introduction of hydrogen generation and thermal recovery managements in the approach brings to completion all the operational aspects of the FCPP that have an impact on the reduction of operational cost and increase in efficiency.

The cost-based approach takes the form of a cost optimization problem subject to system technical and operational constraints. In this research, the resulting objective function is modified according to different operational strategies and the results are compared with a base case (without hydrogen and thermal energy management strategies). For example, the surplus hydrogen/thermal energy can be sold to other customers or can be stored and reused based on system economics. The unmet thermal energy demand on the other hand, can be supplied either by using gas or electricity. Furthermore, since the FCPP is assumed to be operated in grid-parallel mode, the local grid service as a buffer during periods of shortfall or surplus generation. To arrive at the daily optimal operational strategy a hybrid technique based on evolutionary programming (EP) and Hill-Climbing (HC) methods [12, 17] is used. The evolutionary programming is employed to search for the near optimal solution while the HC method is used to ensure feasibility during the solution process.

1.4.1 Development of the cost-based approach

At full load, the FCPP produces thermal energy as a by product, approximately equal to the electrical energy [13]. Besides, the difference between the maximum capacity of the FCPP and the generated electric power can be utilized to produce hydrogen. In order to handle excess electrical/thermal energy and hydrogen production, it is essential to introduce thermal energy recovery and hydrogen production strategies.

1.4.1.1 Recovered thermal energy strategy

The FCPP operates within the efficiency range approximately 30-40 percent. The efficiency is slightly higher at low load compared to full load operation. In Ref. [13], experimentally determined efficiency and thermal to electrical energy ratio curves have been developed. The efficiency and the thermal to electrical energy ratio are functions of the part load ratio, PLR . Mathematical expressions to approximate the curves have been developed in Ref. [13] as follows:

For $PLR_j < 0.05$

$$\eta_j = 0.2716, \quad r_{TE,j} = 0.6801 \quad (1.4.1)$$

For $PLR_j \geq 0.05$

$$\eta_j = 0.9033PLR_j^5 - 2.9996PLR_j^4 + 3.6503PLR_j^3 - 2.0704PLR_j^2 + 0.4623PLR_j + 0.3747 \quad (1.4.2)$$

$$r_{TE,j} = 1.0785PLR_j^4 - 1.9739PLR_j^3 + 1.5005PLR_j^2 - 0.2817PLR_j + 0.6838 \quad (1.4.3)$$

where η_j is the fuel cell electrical efficiency at interval j , PLR is the part load ratio (P_j/P^{max}), and r_{TE} is the thermal to electrical energy ratio.

The thermal power recovered from the fuel cell based on the electric and hydrogen power output can be calculated as follows:

$$P_{th,j} = r_{TE}(P_j + P_a + P_{H_j}) \quad (1.4.4)$$

where P_a is the power for auxiliary devices (kW), P_{H_j} is the equivalent electric power for hydrogen production (kW), P_j is the electrical power produced at interval j (kW) less the power for auxiliary devices, and $P_{th,j}$ is the thermal load at interval j (kW).

In PEM FCPP thermal energy is recovered mainly from the reformer where the temperature rises to about 365°C. The recovery from the stack is neglected due to the lower operating temperature (70-80°C). This research considers thermal load (space heating and hot water) as part of the loading of the PEM FCPP along with electric loads. The thermal load is satisfied by utilizing the recovered thermal energy from the FCPP, and if necessary through the direct use of natural gas.

Based on the difference between the generated thermal energy and the thermal load, four different strategies can be considered:

If the recovered thermal energy is less than the thermal load, the difference between the recovered thermal energy and the thermal load can be supplied using, **Strategy 1**: natural gas, and **Strategy 2**: electricity from the FCPP or from the local grid. In Strategy 2 the difference between the recovered thermal energy and the thermal load is added to the electrical load of the FCPP.

If the recovered thermal energy is more than the thermal load the following strategies can be established: **Strategy 3**: the surplus recovered thermal energy can be sold to other neighborhoods, and **Strategy 4**: the surplus thermal energy can be stored and reused based on system economics. In this case, the stored thermal energy can be calculated as follows:

$$P_{Thr,st,j} = P_{Thr,st,j-1} + \max(P_{th,j} - L_{th,j}, 0) \eta_{stth} - P_{th_usage,j} \quad (1.4.5)$$

where $\eta_{st,th}$ is the thermal energy storage efficiency, $P_{Thr,st,j}$ is the stored thermal power at interval j (kW), $P_{th,j}$ is the thermal load at interval j (kW), $L_{th,j}$ is the thermal load demand at interval j (kW), and $P_{th_usage,j}$ is the amount of thermal power used from thermal reservoir at interval j (kW)

1.4.1.2 Hydrogen production strategy

To include the hydrogen in the FCPP model, the equivalent electric power (P_{Hj}) for generated hydrogen is considered at the fuel cell stack output as shown in Fig. 1.4.1. Figure 1.4.1 (upper figure) reflects the electric power output and hydrogen output locations in the FCPP stages. Fig. 1.4.1 (lower figure) shows the location of P_{Hj} in the FCPP.

Considering P_{Hj} at the stack terminals makes it possible to quantify the production of hydrogen (kg/s) in terms of electrical energy. The hydrogen production in kg/s can be calculated using P_{Hj} as follows [9]:

$$(H_2) \text{ Amount} = 1.05 \times 10^{-8} \frac{P_{Hj}}{v_{cell}} \quad (1.4.6)$$

where v_{cell} is the cell operating voltage, $v_{cell} = 0.6$ (V).

Based on the difference between the maximum capacity of the FCPP and the generated electric power at each interval, the strategy of hydrogen production from the FCPP can yet again be considered in two different strategies. **Strategy 5**: the surplus hydrogen energy can be sold to other customers, and **Strategy 6**: the surplus hydrogen energy can be stored and reused to generate electrical energy based on system economics. In this

case, the hydrogen generated at high thermal demand intervals is stored in a hydrogen tank. During low thermal demand periods the stored hydrogen along with the hydrogen produced from the reformer are used to produce electrical energy.

Two hydrogen streams are considered in this strategy as shown in Fig. 1.4.2. The secondary hydrogen stream produces electric power only. On the other hand any hydrogen produced by the reformer results in the generation of thermal energy. The recovered thermal energy from the stack is neglected.

In the hydrogen storage strategy the stored hydrogen energy ($P_{Hst,j}$) can be calculated as follows:

$$P_{Hst,j} = P_{Hst,j-1} + P_{H,j} \eta_{stH} - P_{H_usage,j} \quad (1.4.7)$$

where $\eta_{st,H}$ is the hydrogen storage efficiency, $P_{H_usage,j}$ is the secondary hydrogen stream amount in kW at interval j, and $P_{Hst,j}$ is the stored hydrogen power at interval j (kW).

Fig. 1.4.1 Hydrogen insertion in the FCPP stages.

Fig. 1.4.2 Hydrogen flow for Strategies 5 and 6.

1.4.1.3 Mathematical formulation

To make the approach comprehensive, this research includes all economic aspects of using recovered thermal energy as well as hydrogen production. The approach considers the electrical power output, the thermal power recovery, hydrogen production, and the power trade with the local grid. The mathematical representation of the cost optimization problem subject to system and operational constraints can be stated as follows:

$$\text{Objective Function} = \min \left(\sum_i \text{Cost}_i - \sum_i \text{Income}_i \right) \quad (1.4.8)$$

First term of the objective function is the overall operational cost. This cost includes the daily fuel cost, the daily cost of purchased electrical energy if the demand exceeds the electrical energy produced, and the daily cost of purchased gas for thermal loads if there is insufficient production of thermal energy. Besides, energy consumption of auxiliary components, start up cost and the operation and maintenance cost of the FCPP are also included in the first term. Second term of the objective function is the total system income pertaining to daily income from the sale of surplus electrical energy, and daily income from the sale of excess thermal and hydrogen energy.

1.4.1.3.1 System cost components

In this section, description and formulation of the objective function cost components of the FCPP are introduced.

a) Fuel cost

The daily fuel cost for electrical and hydrogen energy generation of the FCPP can be written as:

$$C_{\text{FUEL}} = c_{n1} T \sum_j \frac{P_j + P_{a_j} + P_{H_j}}{\eta_j} \quad (1.4.9)$$

where η_j is the fuel cell electrical efficiency at interval j , C_{FUEL} is the daily fuel cost (\$), c_{n1} is the price of natural gas for FCPP (\$/kWh), and T is the length of time interval (h).

Since thermal energy is a byproduct of electrical and hydrogen power generation, it has no effect on the fuel cost.

b) Purchased energy cost

The system can buy energy to satisfy both the electric and thermal load in case of generation shortfall. It is to be noted that if the demand exceeds the electrical energy produced, the required amount of electrical energy is purchased from the local grid and/or drawn from the hydrogen reservoir. As stated in the introduction, various strategies for FCPP management are considered to meet the required electrical and thermal load demand. The cost of purchased electrical and thermal energy may differ based on the applied strategy.

To meet the shortfall from the local grid only, the following equation can be used to define the energy cost:

$$C_{\text{EL,P}}^{(1)} = c_{el,p} T \sum_j \max(L_{el,j} - P_j, 0) \quad (1.4.10)$$

where $C_{\text{EL,P}}$ is the daily cost of purchasing electric power from the local grid (\$), $c_{el,p}$ is the tariff for purchasing electricity (\$/kWh), and $L_{el,j}$ is the electrical load demand at interval j (kW).

On the other hand, if the local grid and the hydrogen reservoir are used together, then following equation can be used:

$$C_{\text{EL,P}}^{(2)} = c_{el,p} T \sum_j \max(L_{el,j} - P_j - P_{H_usage,j}, 0) \quad (1.4.11)$$

As for the thermal generation shortfall, this can be met either by gas or by electricity. The following cost equation can be used if in thermal energy differential is compensated by natural gas:

$$C_{\text{GAS}}^{(1)} = c_{n2} T \sum_j \max(L_{th,j} - P_{th,j}, 0) \quad (1.4.12)$$

where C_{GAS} is the daily cost of natural gas (\$) and c_{n2} is the fuel price for residential loads (\$/kWh).

If the surplus recovered thermal energy is stored and reused by the system, the cost of purchased natural gas is:

$$C_{GAS}^{(2)} = c_{n2} T \sum_j \max(L_{th,j} - P_{th,j} - P_{th_usage,j}, 0) \quad (1.4.13)$$

Assuming the household appliances are all electrical, the difference between demand (electrical plus thermal) and generated energy (electrical plus recovered thermal energy) must be compensated by the use of electricity. In this case, cost of purchased energy (electrical and thermal) can be defined as below:

$$C_{EL,P}^{(3)} = c_{el,p} T \sum_j \max(L_{el,j} + \max(L_{th,j} - P_{th,j}, 0) - P_j, 0) \quad (1.4.14)$$

c) Excess energy storage cost

When the thermal and hydrogen output power from the FCPP exceeds the load demand the surplus energy is sold or stored for future use. For that reason there is a daily cost incurred in the selling or storing process because of the consumption of electric energy by the pumping apparatus. By taking into account storage efficiency, η_{st} at interval j , the pumping cost for thermal and hydrogen energy can be defined as:

$$C_{ST,TH} = c_{p1} T \sum_j P_{th,j} \eta_{st,th} \quad (1.4.15)$$

$$C_{ST,H} = c_{p2} T \sum_j P_{H,j} \eta_{st,H} \quad (1.4.16)$$

where c_{p1} is the thermal energy pumping cost \$/kWh, c_{p2} is the hydrogen pumping cost \$/kWh, $C_{ST,H}$ is the daily cost of storing hydrogen (\$), and $C_{ST,TH}$ is the daily cost of storing thermal energy (\$).

d) Maintenance and start-up cost

The operation and maintenance cost (COM) are designed as a function of output power. The startup cost is dependant upon the temperature and the off time of the FCPP and therefore can be defined as a function of time (t_{off}) and FCPP cooling time constant τ as follows:

$$C_{SUP} = \alpha + \beta(1 - e^{-\frac{t_{off}}{\tau}}) \quad (1.4.17)$$

where α, β are the hot and cold start up cost respectively, τ is the fuel cell cooling time constant (h), C_{SUP} is the daily start up cost (\$), and t_{off} is the time the FCPP has been off (h).

1.4.1.3.2 System incomes

This section presents description and formulation of the income due to excess energy management strategies of the FCPP system.

a) Surplus electrical energy income

It is assumed that when the electrical output power from the FCPP exceeds the electrical demand, the surplus electrical energy is sold to the local grid. In this case, daily income from selling excess electrical energy (\$) is,

$$P_{EL,S}^{(1)} = c_{el,s} T \sum_j \max(P_j - L_{el,j}, 0) \quad (1.4.18)$$

where $c_{el,s}$ is the tariff for selling electricity (\$/kWh) and $P_{EL,S}$ is the daily income from selling electric power to local grid (\$).

If available stored hydrogen energy in the reservoir is used to produce electrical power, then daily income from selling excess electrical energy (\$) is,

$$P_{EL,S}^{(2)} = c_{el,s} T \sum_j \max(P_j + P_{H_usage,j} - L_{el,j}, 0) \quad (1.4.19)$$

Assuming unsupplied thermal load is met by the use of electricity, then the total daily income from the sale of excess electrical energy (\$) is,

$$P_{EL,S}^{(3)} = c_{el,s} T \sum_j \max(P_j - L_{el,j} - \max(L_{th,j} - P_{th,j}, 0), 0) \quad (1.4.20)$$

b) Surplus thermal energy income

When the system produces surplus thermal energy, then the daily income from the sale of this surplus energy (\$) is given as,

$$P_{TH,S} = c_{n3} T \sum_j \max(P_{th,j} - L_{th,j}, 0) \cdot \eta_{st,th} \quad (1.4.21)$$

where c_{n3} is the thermal energy selling price (\$/kg) and $P_{TH,S}$ is the daily income from sale of thermal energy (\$).

In case of storage and reuse of thermal energy, the unused and stored thermal energy can be sold to other neighborhoods at the end of the day. In this case, the daily income from the sale of unused thermal energy is,

$$P_{TH_ST,S} = c_{n3} P_{th,end} \quad (1.4.22)$$

where $P_{ST-TH,S}$ is the daily income from selling thermal energy at the end of day (\$) and $P_{th,end}$ is the available stored thermal energy at the end of the day (kWh).

c) Surplus hydrogen energy income

The income from the sale of surplus hydrogen energy (\$) is given as,

$$P_{H,S} = c_{Hs} T \sum_j \eta_{st,H} (P_{H,j}) F \quad (1.4.23)$$

where c_{Hs} is the hydrogen selling price (\$/kg), F is a conversion factor (kg of hydrogen/kW of electric power), where $F=1.05*10^{-8}/v_{cell}$, and $P_{H,S}$ is the daily income from selling hydrogen (\$).

In case of storage and reuse of hydrogen, the unused stored hydrogen energy can be sold at the end of the day. In this case, daily income from the sale of unused hydrogen is,

$$P_{H-ST,S} = c_{Hs} P_{H,end} F \quad (1.4.24)$$

where $P_{H,end}$ is the available stored hydrogen at the end of the day (kWh) and $P_{H-ST,S}$ is the daily income for selling hydrogen at the end of the day (\$).

1.4.1.3.3 System constraints

It is assumed in this research that the objective function (1.4.8) is constrained by FCPP rated capacity, ramp rate constraint, minimum up/down time limits, and a maximum number of FCPP start-stop cycles. The constraints are given below:

$$P^{min} \leq P_{T_j} \leq P^{max} \quad (1.4.25)$$

$$P_{T_j} - P_{T_{j-1}} \leq \Delta P_u \quad (1.4.26)$$

$$P_{T_{j-1}} - P_{T_j} \leq \Delta P_d \quad (1.4.27)$$

$$(T_{j-1}^{on} - MUT)(U_{j-1} - U_j) \geq 0.0 \quad (1.4.28)$$

$$(T_{j-1}^{off} - MDT)(U_j - U_{j-1}) \geq 0.0 \quad (1.4.29)$$

$$n_{start-stop} \leq N^{max} \quad (1.4.30)$$

where ΔP_D is the lower limit of the ramp rate, ΔP_u is the upper limit of the ramp rate, MDT is the minimum down-time (number of intervals), MUT is the minimum up-time (number of intervals), N^{max} is the maximum number of start-stop events, $N_{start-stop}$ is the number of start-stop events, P^{max} is the maximum limit of generating power (kW), P^{min} is the minimum limit of generating power (kW), P_{Tj} is the total power produced at interval j , where $P_{Tj} = P_j + P_a + P_{Hj}$, T^{off} is the FCPP off-time (number of intervals), T^{on} is the FCPP on-time (number of intervals), and U is the FCPP on-off status, $U = 1$ for running, $U = 0$ for stopping.

1.4.1.3.4 System efficiency

The efficiency of the FCPP given in [18] is based on the electrical output power versus the input gas power. This efficiency neglects thermal and hydrogen energies, which results in efficiency range of 30-40%. However, using the hydrogen and recovered thermal power from the FCPP enhances the overall efficiency considerably. In this case the overall efficiency can be calculated as follows:

$$\eta_{overall_j} = \frac{P_j + P_a + \eta_{st,H} P_{H,j} + \eta_{st,th} P_{th,j}}{\left[\frac{P_j + P_a + P_{H,j}}{\eta_j} \right]} \quad (1.4.31)$$

where $\eta_{overall,j}$ is the overall fuel cell power plant efficiency.

1.4.2 Evolutionary Programming (Ep)-Based Solution Methodology

Evolutionary programming (EP) can be traced back to early 1950's when Turing discovered a relationship between machine learning and evolution. During the 1980's, advances of computer technology permitted the use of evolutionary programming to solve difficult real-world optimization problems [19]-[21]. The general scheme for solving optimization problems using evolutionary programming can be summarized as follows:

Initialization: An initial population of uniform randomly distributed solution is selected.

Mutation: A Gaussian random variable is added to all the current generation individuals using Eq. (1.4.32) with uniform probability.

$$S_{i+m,j} = S_{i,j} + N(0, \beta_i v(S_i) + z_j) \quad j = 1, \dots, k \quad (1.4.32)$$

where m is the number of individuals in the current generation, $v(S_i)$ is the fitness score, S_i is the j^{th} element of the i^{th} individual, $N(\mu, \sigma^2)$ is the Gaussian random variable with mean μ and variance σ^2 , β_i is a constant of probability to scale $v(S_i)$, z_j is an offset to

guarantee a minimum amount of variance, i is individual number, and k is number of variables in each individual.

Competition: A probabilistic selection scheme is used to assign a weight to each offspring individual according to a comparison between current individual and a randomly chosen one. The weights are calculated as follows:

$$W_i = \sum_{j=1}^N W_{i,j} \quad (1.4.33)$$

where N is the competition number generated randomly, and $W_{i,j}$ is either 0 or 1 depending on the competition of the individual with another individual selected randomly from the population. The value of $W_{i,j}$ can be calculated as follows:

$$W_{i,j} = \begin{cases} 1 & \text{if } v(S_i) \leq v(S_p) \\ 0 & \text{otherwise} \end{cases} \quad (1.4.34)$$

where $p = \lfloor 2mu_l + 1 \rfloor$, $p \neq i$, $u_l \sim U(0,1)$

The above mentioned EP procedure is used to search for the optimal operational strategy for the FCPP. The Hill-Climbing technique [22] is used to watch for the infeasibility of the solutions during the search process.

1.4.3 Tests and results

The proposed model has been applied to a 250 kW grid-parallel FCPP that supplies a residential neighborhood. The IEEE-RTS load profile with a peak of 250 kW [23] is used to simulate the hourly electrical load profile of the system. In this test system, the weekly, daily and hourly peak load values are given in percent of annual, weekly and daily peak loads, respectively. The winter hot water usage and space heating load for Atlanta, Georgia [13] is used to estimate the thermal load profile. The thermal load is used along with the electrical load profile to simulate total hourly operation of the FCPP. Table 1.4.1 gives gas prices and FCPP/EP parameters for the following case studies. The thermal energy selling price is set to a value lower than the price for buying residential gas to give incentive to other neighborhoods to buy thermal energy from the FCPP.

Case 1: The recovered thermal energy management strategies (Strategies 1-4) are individually tested using the electrical and estimated thermal load profiles. A summary of test results for each strategy is given in Table 1.4.2. It is clear from the table that using gas to compensate for the thermal generation shortage yields lower overall cost than using electricity. Considering gas compensation as a base case, the electricity compensation makes the system lose \$2,237.45 yearly. Strategies 3 (selling thermal energy) and 4 (storing thermal energy) are tested using gas and electricity to supply the shortfall in thermal energy supply. It is shown from testing Strategies 3 and 4 that using

gas to compensate the thermal energy shortfall gives a lower cost and higher saving per year compared to the cost of using electricity. Therefore, it can be concluded that using gas with any of the recovered thermal energy management strategies results in a lower overall cost than when the electricity is used.

Table 1.4.1 FCPP and Evolutionary Programming Parameters.

Maximum limit of generating power, P^{max} (kW)	250
Minimum limit of generating power, P^{min} (kW)	0.0
Length of time interval, T (h)	0.25
Upper limit of the ramp rate, ΔP_u (kW)	20
Lower limit of the ramp rate, ΔP_D (kW)	25
Price of natural gas for FCPP, c_{n1} (\$/kWh)	0.04
Tariff for purchasing electricity, $c_{el,p}$ (\$/kWh)	0.13
Tariff for selling electricity, $c_{el,s}$ (\$/kWh)	0.08
Fuel price for residential loads, c_{n2} (\$/kWh)	0.06
Thermal Energy Selling Price, c_{n3} (\$/kWh)	0.04
Hydrogen selling price, c_{H_s} (\$/kg)	1.80
Thermal/Hydrogen pumping cost, c_{p1} , c_{p2} (\$/kWh)	0.01
Thermal storage efficiency, $\eta_{st,th}$ (%)	90
Hydrogen storage efficiency, $\eta_{st,h}$ (%)	95
Hot start up cost, α (\$)	0.05
Cold start up cost, β (\$)	0.15
The fuel cell cooling time constant, τ (h)	0.75
Minimum up-time, MUT (number of intervals)	2
Minimum down-time, MDT (number of intervals)	2
Maximum number of starts-stops, N^{max}	5
Maximum number of evolutionary generation	20000
Number of individuals	150

Comparing Strategies 3 and 4, shows that storing and reusing thermal energy saves the system \$8,606.70 yearly compared to \$5,161.10 yearly in the case of selling the available thermal energy. Therefore, storing thermal energy along with gas compensation is the best strategy in terms of overall cost and savings per year.

Case 2: In this case hydrogen Strategies 5 (selling hydrogen) and 6 (storing hydrogen) are tested individually using the available electrical and thermal load profiles. The test results are shown in Table 1.4.3. Comparing Strategies 5 and 6 with the Base Case (no hydrogen management strategy) shows that the strategy of selling the available hydrogen saves the system \$11,004.75 yearly and storing the hydrogen saves the system \$12,424.60 yearly. In conclusion, the storing of hydrogen is better than the selling of hydrogen in terms of the overall cost and total savings.

In the following test cases a mix between different hydrogen and thermal energy management strategies is tested using the available thermal and electrical load profiles.

Case 3: In this case, a mix of Strategy 3 (selling thermal energy) and Strategy 5 (selling hydrogen) is tested. In this case the objective function (OF) is given as below:

$$OF = \min [(C_{FUEL} + C_{EL,P}^{(1)} + C_{GAS}^{(1)} + C_{ST,TH} + C_{ST,H} + C_{OM} + C_{SUP}) - (P_{EL,S}^{(1)} + P_{TH,S} + P_{H,S})] \quad (1.4.35)$$

The obtained cost components from minimizing Eq. (1.4.35) are given in Table 1.4.4. Comparison of the total cost of Case 3 (Table 1.4.4) and the cost of applying individual strategies shows that the selling hydrogen and thermal energy together saves the system \$34.12 daily as compared to selling thermal energy with gas compensation. On the other hand, the saving is \$18.11 when compared to selling hydrogen only. These savings amount to \$12,453.80 and \$6,610.15 per year respectively. Comparing Case 3 with the Base Case overall cost (Table 1.4.2) shows that selling hydrogen and thermal energy saves the system \$48.26 daily or \$17,614.90 yearly.

Table 1.4.2 Case Studies for Energy Management Strategies of Thermal Energy.

Daily cost components (\$)	Gas Compensation (Base Case)	Electrical Compensation	Thermal Selling		Thermal Storing	
			Gas	Elect.	Gas	Elect.
Fuel cost	539.72	591.46	540.67	603.13	536.85	572.56
Purchased electricity cost	2.31	10.40	0.0	0.01	0.0	0.0
Sold electricity income	99.48	108.96	98.83	112.32	96.95	112.60
Residential natural gas cost	45.55	0.0	46.24	0.00	19.95	0.00
O&M cost	23.79	25.11	23.84	25.54	23.72	24.70
Income from sale of thermal energy	N/A	N/A	18.92	18.92	0.0	0.0
Thermal storing cost	N/A	N/A	4.73	4.73	4.73	5.51
Total cost	511.88	518.01	497.74	502.17	488.30	490.17
Total savings per year	0.0	-2,237.45	5,161.10	3,544.15	8,606.70	7,924.15

The electric power trade with the local grid, and the electrical load/generation for Case 3 are shown in Fig. 1.4.3. Figure 1.4.4 shows the hydrogen production and thermal load/generation for the same case. The effect of the thermal load on the optimal operation of the FCPP can be noticed from Fig. 1.4.4.b. During the low thermal load period (8:00 to 18:00) the system satisfies the electrical load while reducing hydrogen production. During this interval it can be noticed that there is no trade of power with the local grid. The excess of thermal energy during this interval is sold to neighboring loads to reduce the system cost. Due to the fact that the FCPP system sells the excess thermal energy for less than the natural gas cost, the FCPP does not produce more excess thermal power and hydrogen during low thermal load intervals. In addition, the storage efficiency of hydrogen and thermal energy has a significant effect on the excess amount of hydrogen and thermal power.

Case 4: In this case, a mix of Strategy 4 (storing thermal energy) and Strategy 5 (selling hydrogen) is tested. The applicable objective function is given as below:

$$OF = \min [(C_{FUEL} + C_{EL,P}^{(1)} + C_{GAS}^{(2)} + C_{ST,TH} + C_{ST,H} + C_{OM} + C_{SUP}) - (P_{EL,S}^{(1)} + P_{H,S} + P_{TH,ST,S})] \quad (1.4.36)$$

The cost components resulting from applying Strategies 4 and 5 are given in Table 1.4.5. Comparing Tables 1.4.5, 1.4.2 and 1.4.3 shows that applying strategies 4 and 5 together

saves the system \$30.55 daily compared to the thermal energy storage option with gas compensation. The saving is \$23.98 daily compared to the cost incurred when only the hydrogen sale strategy is chosen. These savings amount to \$11,150.75 and \$8,752.70 per year respectively. Comparing the cost of Case 4 with the Base Case (Table 1.4.2) shows that applying strategies 4 and 5 together saves the system \$54.13 daily or \$19,757.45 yearly.

Table 1.4.3 Case Studies for Energy Management Strategies of Hydrogen Power.

Daily cost components (\$)	Base Case	Strategy 1 H_Selling	Strategy 2 H_Storing
Fuel cost	539.72	597.36	550.78
Purchased electricity cost	2.31	16.65	1.28
Sold electricity income	99.48	10.89	7.79
Residential natural gas Cost	45.55	10.50	10.48
O&M cost	23.79	17.70	14.72
Income from sale of hydrogen	N/A	164.07	107.88
Hydrogen storing cost	N/A	14.47	16.27
Total cost	511.88	481.73	477.84
Total savings per year	0.0	11,004.75	12,424.60

Table 1.4.4 Case Studies for Selling Hydrogen and Storing Thermal Energy.

Daily cost components (\$)	Case 3
Fuel cost	608.68
Purchased electricity cost	0.02
Sold electricity income	5.31
Residential natural gas Cost	10.41
O&M cost	17.99
Income from sale of hydrogen	167.51
Income from sale of thermal energy	20.57
Hydrogen storing cost	14.77
Thermal storing cost	5.14
Total cost	463.62

Fig. 1.4.3 Case 3, (a) electrical power trade with the grid, (b) electrical load and power generation.

Fig. 1.4.4 Case 3, (a) hydrogen production, (b) thermal load and generation.

Table 1.4.5 Case Studies for Selling Hydrogen and Storing Thermal Energy.

Daily cost components (\$)	Case 4
Fuel cost	565.43
Purchased electricity cost	0.02
Sold electricity income	2.33
Residential natural gas Cost	4.46
O&M cost	17.80
Income from sale of hydrogen	145.86
Income from sale of thermal energy	0.00
Hydrogen storing cost	12.86
Thermal storing cost	5.37
Total cost	457.75

The power trade with the local grid and the electric load/generation for Case 4 are given in Fig. 1.4.5. Fig. 1.4.6 shows the hydrogen production and thermal load/generation for the same case.

Fig. 1.4.5 Case 4, (a) electrical power trade with the grid, (b) electrical load and power generation.

Fig. 1.4.6 Case 4, (a) hydrogen production, (b) thermal load and generation.

From Fig. 1.4.6.b it is clear that the system stored thermal energy during the low thermal demand period (8:00 to 18:00). During the high thermal load period (after 18:00) the system uses all the stored thermal energy, reduces the amount of generated hydrogen, and produces electrical energy enough to satisfy the electric load as shown in Figs. 1.4.5 and 1.4.6.

Case 5: In this case a mix of Strategy 3 (sale of thermal energy) and Strategy 6 (hydrogen storage) is tested using the available electrical and thermal load profiles. In this case the objective function is given by equation (1.4.37).

$$OF = \min [(C_{FUEL} + C_{EL,P}^{(2)} + C_{GAS}^{(1)} + C_{ST,TH} + C_{ST,H} + C_{OM} + C_{SUP}) - (P_{EL,S}^{(2)} + P_{TH,S} + P_{H,ST,S})] \quad (1.4.37)$$

The cost components resulting from Strategies 3 and 6 are shown in Table 1.4.6. Power trade with the local grid and the electric load/generation for Case 5 are given in Fig. 1.4.7. Figure 1.4.8 shows hydrogen production and thermal load/generation for the same case.

Table 1.4.6 Case Studies for Storing Hydrogen and Selling Thermal Energy.

Daily cost components (\$)	Case 5
Fuel cost	613.16
Purchased electricity cost	0.36
Sold electricity income	0.65
Residential natural gas Cost	10.46
O&M cost	17.24
Income from sale of hydrogen	177.28
Income from sale of thermal energy	21.95
Hydrogen storing cost	16.52
Thermal storing cost	5.49
Total cost	463.36

Fig. 1.4.7 Case 5, (a) electrical power trade with the grid, (b) electrical load and power generation.

Fig. 1.4.8 Case 5, (a) hydrogen production, (b) thermal load and generation.

Comparing the cost of selling thermal energy and storing hydrogen with the cost incurred in Case 3, it is clear that the cost for both cases is almost the same. The reason for this is shown in Fig. 1.4.7.b where the system uses a small amount of hydrogen from the hydrogen tank to satisfy the electric load. At the end of the day the remaining hydrogen is sold. For a different thermal and electrical load profile the sale of thermal energy and storage of hydrogen is expected to yield lower cost than the sale of these two.

Case 6: In this case a mix of Strategy 4 (Storage of thermal energy) and Strategy 6 (Storage of hydrogen) is tested using the available electrical and thermal load profiles. The objective function is given in (1.4.38). The cost components are given in Table 1.4.7. In this case the FCPP produces just enough electric power to satisfy the electric load resulting in no trade with the local grid. The electric load/generation for Case 6 is given in Fig. 1.4.9. Figure 1.4.10 shows the hydrogen production and thermal load/generation for the same case.

$$OF = \min [(C_{FUEL} + C_{EL,P}^{(2)} + C_{GAS}^{(2)} + C_{ST,TH} + C_{ST,H} + C_{OM} + C_{SUP}) - (P_{EL,S}^{(2)} + P_{TH,ST,S} + P_{H,ST,S})] \quad (1.4.38)$$

Comparing the cost of storing hydrogen and thermal energy with cost of Case 4 shows

that the costs for the two cases are almost the same. The reason for this is clear from Fig. 1.4.9 because the FCPP used a small amount of hydrogen from the tank to satisfy the electric load. The rest of the hydrogen is sold at the end of the day. Although the total cost for storing hydrogen and thermal energy is the same as for the sale of hydrogen and storage thermal energy, the production levels of thermal, electrical, and hydrogen powers are different. For different thermal and electrical load profiles storing hydrogen and thermal energy is expected to yield the lowest operating cost.

Table 1.4.7 Case Studies for Storing Hydrogen and Thermal Energy.

Daily cost components (\$)	Case 6
Fuel cost	565.30
Purchased electricity cost	0.00
Sold electricity income	0.01
Residential natural gas Cost	4.59
O&M cost	17.39
Income from sale of hydrogen	148.51
Income from sale of thermal energy	0.09
Hydrogen storing cost	13.63
Thermal storing cost	5.29
Total cost	457.60

Fig. 1.4.9 electrical load and power generation for Case 5.

Fig. 1.4.10 Case 5, (a) hydrogen production, (b) thermal load and generation.

The summary of the daily total cost and yearly savings compared to the Base Case for all

the strategies discussed is given in Table 1.4.8. It is clear from Table 1.4.8 that storing hydrogen and thermal energy gives the lowest cost and highest savings.

Using equation 1.4.31, the efficiencies for Cases 3-6 are calculated and given in Fig. 1.4.11. It is evident from Fig. 1.4.11 that including the hydrogen and thermal energy in efficiency calculations increases the efficiency range to 64%-67%. Fig. 1.4.11 shows that Case 3 (selling hydrogen and thermal energy) and Case 5 (selling thermal energy and storing hydrogen) have almost the same efficiency variations. While Case 4 (storing thermal energy and selling hydrogen) and Case 6 (storing thermal energy and hydrogen) reflect the same efficiency level. These efficiency levels are seen to be higher than those of Cases 3 and 5 during high thermal load intervals. In conclusion, storing thermal energy and hydrogen results in the lowest operational cost and the highest efficiency.

Table 1.4.8 COSTS and Savings Summary for all Strategies.

	Daily total cost (\$)	Yearly savings(\$)
Gas compensation (Base Case)	511.88	0.0
Electricity compensation	518.01	-2,237.45
Thermal selling (gas compensation)	497.74	5,161.10
Thermal selling (electricity compensation)	502.17	3,544.15
Thermal storing (gas compensation)	488.30	8,606.70
Thermal storing (electricity compensation)	490.17	7,924.15
Hydrogen selling	481.73	11,004.75
Hydrogen storing	477.84	12,424.60
Thermal-selling, hydrogen-selling	463.62	17,614.90
Thermal-storing, hydrogen-selling	457.75	19,757.45
Thermal-selling, hydrogen-storing	463.36	17,709.80
Thermal-storing, hydrogen-storing	457.60	19,812.20

Fig. 1.4.11 Efficiency curves for Cases 3-6.

1.5 Analysis of Load Smoothing

1.5.1 Objectives

- The first objective is to find the amount of inherent load smoothing that occurs in a micro-grid environment
- The second is to analyze home load profiles to determine the loads/appliances that should be available for load management

1.5.2 Hypothesis

It is hypothesized that the load curve of homes in a micro-grid community is smoother or stays closer to a constant value than the load curve of an individual home with no computer management in either situation. The reason a smoother curve is expected is that occupants in different households will have different daily routines and will use high powered loads at different times. If scale is taken into account, then it follows that the micro-grid community will have a smoother load curve than a single home. The team has chosen to define a smooth curve with two metrics: standard deviation and load factor. A smoother curve has a lower standard deviation because its data points, whether peaks or valleys, are closer to the average of the data points. A perfectly smooth curve has a standard deviation of zero. Next, load factor is defined as the average load divided by the peak load. The higher the load factor, the smoother the load curve is, up to a maximum of one. A perfectly smooth load curve has a load factor of one.

1.5.3 Micro-grid versus a single house

The comparison uses artificial load profiles which are partly based upon data gathered from real homes. There are ten house profiles in all consisting of a combination of one or more of the following: a man, woman, girl, boy, teenage boy, teenage girl, retired man and/or retired woman. A small amount of randomness is present because any of the profiles can be designated as a “Traveler.” A traveler has a random chance of being out of the house for a few days of the week and was introduced so that profiles generated would not all appear exactly the same.

Again, the metrics used for comparison are the standard deviation and load factor of power usage per day in a single home versus that of a hypothetical micro-grid. The closer the standard deviation is to zero and the closer the load factor is to one, the smoother the load curve. According to the hypothesis, the standard deviation of a micro-grid should be lower than the average standard deviation of the individual houses and the load factor should be higher. Results to date show that the micro-grid’s standard deviation is 14.8 kW, while the average standard deviation of the individual houses is 1.53 kW. If you multiply by the scaling factor of 10, then the standard deviation of the individual houses is 3.3% higher than the micro-grid. The micro-grid load factor calculates to 23.1% while the average load factor of the individual homes is 20.4%. The load factor increases by 2.7 percentage points when in the micro-grid configuration. Both calculations show that the

load curve is smoother if the houses operate within a micro-grid. More randomized occupant profiles would result in an even smoother curve. Fig. 1.5.1 shows typical results for unmanaged/managed loads in a ten house micro-grid. The average powers consumed are 14.4 kW and 14.6 kW for the unmanaged and managed cases, respectively. It is interesting to note that the average power in the managed scenario is slightly higher than the unmanaged one.

Fig. 1.5.1 Unmanaged and Managed Loads (kW) v. Time (Hours)

1.5.4 Load profiles for manageable appliances

In the process of gathering data for analysis, the project team ran scenarios in the laboratory house located on the USA campus. The team first ran unmanaged scenarios in order to create a baseline. The team then ran scenarios where the Micro-grid Energy Management System or MEMSYS was enabled, so management occurred whenever energy usage exceeded 5kW. The unmanaged scenarios ran four hours and typically consumed around 13-15 kWh of total energy. The managed scenarios ran longer because appliances that were turned off had to wait to run when enough power was available. The managed scenarios ran typically an hour longer and used slightly more energy overall. If the managed scenarios were limited to the same four hours as the unmanaged scenarios, then energy used would be lower because average power is lower in the managed scenarios. Also, certain appliances such as the stove were not turned back on after management. Future runs will include overriding the MEMSYS and keeping the stove and/or other appliances on. Analysis of the collected data shows that average power in the managed house dropped by 400W. The standard deviation dropped by 60% and the

load factor dropped by close to 3%. The load factors for the lab house are 40.5% and 37.5% for the unmanaged house and the managed house, respectively. The measured load factors are significantly higher than the simulated load factors which hovered in the lower twenties, and the primary reason for this difference is that the laboratory run scenarios only lasted four hours and high powered appliances were in use the entire time. The simulated load profiles however run all day with significant periods where the house is empty of occupants and few loads are in use.

During the runs, the team noted the management decisions that seemed to unnecessarily inconvenience occupants. Inconveniences occurred whenever the power hovered around the 5kW limit. Because the limit is a hard limit, even a small load over 5kW causes appliances to shut off. The MEMSYS would often manage large loads first instead of small ones that could easily bring the total under 5kW. Also, the MEMSYS would sometimes cycle large loads like the dryer on and off within the space of a few minutes repeatedly because the water heater or refrigerator would run intermittently.

Analyzing the data revealed measurement error that would cause the power measured to fluctuate enough to place the management software in a cycle of switching loads on and off every few seconds. Sometimes a load that was turned off would register 6W of power usage. For larger loads using a different current sensor, the error was around 36W. The measurement errors increase as loads increase so it is thought that crosstalk in the wires contributes to the error. Other possible sources of error include the multiplexer which is built with components that are already 5% error tolerant. The smaller current sensors, which output 1V at the full scale of 20A, have a 4% error. Many house loads typically run at 1A or less. The reading is then sent to an A/D card which has a range of $\pm 5V$ and a sensitivity of 12 bits. Since the reading is always positive and 1V max, the slightest interference can lead to a visible error.

To keep an error of 6W within the 5% tolerance range, a load must be at least 72W. Luckily, the lights and many other appliances draw more than 72W or draw a small enough wattage as to fade into the noise. The larger 50A sensors output .45V and are constructed with a 3% error. Voltage fluctuations in this sensor would result in errors that are potentially greater than the 20A sensors since it outputs on a smaller voltage range. Again, if an output error of less than 5% is required, then the maximum error of 36W would dictate the loads they measure be 720W or greater. Loads outfitted with the larger sensors draw power in the range of 600-3000W, so some loads fall below the 720W threshold needed for 5% accuracy. We will look at ways of raising the accuracy for the loads that fall below the 600W threshold, but correction of the problem may require a redesign of the multiplexer.

In the third year, the team is investigating the source of these errors and attempt to ensure they remain within the 5% error tolerance. One potential source of the interference is cross talk between the wires, and the solution to this may be to place the current sensors at the wall outlet and send the readings wirelessly or by power line carrier. Another potential source is the wires between the multiplexer and the A/D card which is located in the computer several feet away. The error could be reduced by eliminating the line loss in

the next version of the MEMSYS hardware. Finally, the hard limit of 5kW will be removed with the inclusion of either fuzzy logic or error compensation algorithms or both. With fuzzy logic, the MEMSYS can operate in a given power band acceptable to the fuel cell and so reduce management cycling. Error compensation may be investigated by creating an error curve and programming it into the MEMSYS to achieve a similar result to the fuzzy logic. Since the third year consists of algorithm improvement, all these proposed solutions fit with the overall direction of the project.

Some nuisance load cycling can also be improved by installing a larger water heater and letting the MEMSYS manage it more often. The fuel cell's co-generation features will help with this aspect also. The team also proposes that in a commercial product, the number of loads that are actually controlled directly by the MEMSYS be reduced. The installation of the current system is very costly and the utility of some of its functions is questionable. If the primary purpose of MEMSYS is power management, the commercial product could simply manage the larger loads in the house like the washer, dryer, water heater and stove. If electric companies can manage the usage of these specific items during their peak usage periods, they will be managing the majority of a home's load. The system could pay for itself in a smaller time period and will be less intrusive.

2. Energy Management

GIFCO Year I and II load management algorithms used rules based logic. We proposed in Year III to enhance these algorithms by using “smart” algorithms whose purpose is to make better management decisions around the energy management thresholds. Our research indicated that a fuzzy control algorithm is well suited for this type of management algorithm. We also studied and modeled a neural network based algorithm for managing appliances based on historical usage patterns, to indicate probability of future demand. What follows is a description of the fuzzy logic control algorithm, the simulation home profiles, the neural network algorithm, and the results of the Load Profile Simulations.

2.1 Algorithms for Load Smoothing

2.1.1 Fuzzy logic control system

The GIFCO Fuzzy Logic Control system consists of two main modules, the Fuzzy Controller module (GIFCO Controller) and the fuzzy engine module (GIFCO Engine) as shown in Figure 2.1.

Figure 2.1 - GIFCO Fuzzy Logic Control System.

The GIFCO Controller handles communications with the external systems, data management, and oversees the execution of the GIFCO Engine. The GIFCO Engine

determines which room needs energy management information and then executes the appropriate room rule engine. These two modules are described in more detail in the following sections.

2.1.1.1 GIFCO controller module

- **Module Inputs**
 - The GIFCO Controller Module takes appliance level load data as input. The load data includes appliance location, management status and priority, and current power consumption. It takes input requests from the user to turn managed loads back on. For the purposes of the GIFCO Year 3 project the module read only simulated load profile data, but it is capable of reading the information from the real world data acquisition hardware.
- **Module Outputs**
 - The module outputs management information. If management is required then it will tell the external system what appliance to manage and whether to turn that appliance on or off.
- **Module Algorithm**
 - The Controller Module algorithm is represented by the flow diagram in Figure 2.2

Figure 2.2 – Controller Module Algorithm

1. Initialization: Upon startup, the Controller Module clears the management list and performs other housekeeping chores related to initialization.
2. Data Input: The Controller reads in all the appliance level load data as described above. The module adds all the raw loads that the user requests to be on to determine the aggregate raw load. It also scans in a predetermined user defined appliance priority list. At this stage, the user can indicate to the module any appliances that he wishes to remain unmanaged and dynamically change the priority list. If the aggregate raw load exceeds the target load, the system goes to Step 3. On the other hand if the aggregate load falls below a pre-specified target load then the system goes to Step 4.
3. Power Save: The module scans the list of unmanaged loads along with any user requests to avoid managing particular appliances. If rooms are unoccupied the module goes to Step 3a. If all rooms are occupied, it goes to Step 3b.
 - a. The module determines appliance priority within its respective room and sends each unoccupied room to the fuzzy engine for evaluation. The

- module determines appliance priority by assigning each appliance a priority relative to the other appliances in its room. As described in the GIFCO Engine section, the fuzzy engine returns the best appliance to manage in a room and the appliance's "score." The fuzzy controller then picks the room that has the best score and adds that room's best appliance to the managed list.
- b. The module assigns each appliance a priority relative to all the appliances in the house (called the ALLOCC list) and sends the list to the GIFCO Engine. For appliances that are common across rooms (HVAC, water heater, outside flood lights, etc.), occupancy is not considered.
 - c. The module adds the appliance with the highest priority returned from the GIFCO Engine to the managed list of appliances and repeats Step 2.
4. Power Restoration: In Step 2, if the module finds that the aggregate load falls below a pre-specified target load, the system then looks for appliances to turn back on.
- a. The module scans the managed list of appliances (ones the fuzzy system turned off previously) and finds the loads in this list which are rescheduleable. The module then sends the new list to the GIFCO Engine.
 - b. After receiving data from the GIFCO Engine, the module restores the load recommended by the Engine. The Engine will return a load whose previous power consumption was less than the current amount of excess power. There are appliances, such as stoves, that are not rescheduleable for safety reasons. The external system must indicate that an appliance has been turned off for an unrescheduleable appliance to be removed from the managed list. When this occurs, the module removes the appliance from the managed list of appliances and repeats Step 2.

2.1.1.2 GIFCO engine module

The GIFCO Engine contains one fuzzy sub-engine for each room in the house. Each of these sub-rooms contains a full set of fuzzy logic rules that evaluate the current conditions and decides the best appliance to manage in the case of excess energy consumption. Each room's fuzzy logic is detailed in Appendix A, showing the information and logic rules that make up that room's sub-engine. Each room receives the same information for each appliance: the appliance priority and its load differential. The load differential is the distance from the amount of power that must be saved to reach the target load and the amount of power that the individual appliance consumes. This variable gives a higher weight to an appliance that is as close to the needed power saving as possible. For example, let's assume the needed power saving is 250 watts and the room has the following three appliances turned on: LRLight, 300 watts; LRFan, 100 watts; powertool1, 1100 watts. The fuzzy system will give a higher weight to the LRLight because it is closer to the needed power than either the LRFan or the powertool1. The fuzzy engine will then execute and evaluate every appliance in the room using the user priority variable, the load differential and the appliance status (which makes sure that only appliances that are turned on are evaluated). The fuzzy engine then scores each appliance. The "score" for an appliance can be NO_CHANGE, NOT_PROMISING,

LESS_PROMISING, NEUTRAL, SLIGHTLY_PROMISING, PROMISING, and VERY_PROMISING. The fuzzy engine then returns the best appliance and its “score” to the fuzzy controller.

2.1.2 Neural network algorithm

A neural network was developed to help predict usage demands of large appliances in the Smart Home. It was observed that appliances that run unnoticed or without regular human interaction tend to be an area of unnecessary power consumption by the home. By predicting usage demands of these appliances, smarter management decisions can be made, and a more efficient Smart Home can be maintained. The hot water heater was modeled, based on water usage collected from a research team member’s home for a 24 hour period. The hot water heater is a prime candidate for predictive management, because in a typical U.S. home, the hot water heater elements are on many times a day, regardless of whether there is a demand for hot water or not. The heater elements are merely maintaining a preset temperature, based on potential demand. The model for the hot water heater in the Smart Home consists of a flow switch installed at the output of the hot water heater, sampling water flow at one minute intervals. A constant flow rate is assumed, and hot water flow is recorded as on or off.

2.1.2.1 GIFCO neural network description

The GIFCO Neural Network (GNN) is a four level, feed-forward, back-propagation neural network. Currently there are 5 input variables to model for the water flow:

1. Time of day (cyclic - hours and minutes combined) – This variable captures changes that are governed by time of day, for example, less use in early morning hours.
2. Day of week (cyclic 0-6) – This variable is to detect changes that occur with a weekly frequency. It should be good at encoding behavior such as more laundry done on weekends, holidays, etc.
3. Day of month (cyclic 0 - days in month) – This variable is to capture changes that occur on a monthly frequency, perhaps tied to household paydays.
4. Month (cyclic 0-11) – This variable is to detect changes in water use based on season of year.
5. Water Flow (time lag data) – The system is allowed to remember what its water flow was 1 minute prior to the current time of day.

These 5 input variables correspond to 9 input neurons. There is not a one-to-one correlation of input variables to input neurons because 4 of the input variables are cyclic or circular in nature. Cyclic variables such as month must be encoded as two variables, sine and cosine of the variable. Each of these variables changes smoothly, and together they uniquely define the circular variable.

There are two hidden layers, the first hidden layer contains 5 neurons and the second hidden layer contains 3 neurons.

There is one output layer with a single neuron. This output layer corresponds with predicted water flow (probability of water flow). It ranges from 0.0 to 1.0.

2.1.2.2 Neural network test results

A three day period of water usage was generated at one minute intervals, and was fed to the neural network for analysis. One disadvantage of the neural network is the amount of time required to train the network. Twenty four hours was required to train three days of hot water flow at one minute intervals. This prohibited larger sample sets from being tested, but very promising results were obtained from a very small sample set of data. A summary of the neural network test is in table 2.1.

Number of Correct Flow Predictions	2100
Number of Missed Flow Predictions	33
Number of False Alarms (predicted flow > 0.5 but no real flow)	13

Table 2.1 Neural Network Summary Results

2.2 Historical Load Profiles Simulation

Based on data collected from GIFCO team member's homes in year one of the GIFCO project, we designed and developed a home profile simulation generator that takes as input the number and characteristics of the occupants of a home, and outputs 7 days worth of simulated loads in the home based on occupancy. These load profiles are at 5 minute intervals, and all use a common set of appliances, which are described in section 2.4.1. A set of 10 basic simulations were created using the generator, and these were used to benchmark and test the Fuzzy Logic energy management algorithms. A total of 50 load profiles were generated and simulated, to study the effects of load smoothing across the micro-grid. The SEMAC software was modified to allow simulation using the load profiles, vs. real-time data collected from the GIFCO home (constructed in year one of the GIFCO project).

2.3 Testing and Data Collection

The data collected during the testing phase is analyzed in Section 1.4 of this report. The results of the data collection are reported in Section 2.4.

2.4 Results

The results of the 10 home simulation using the improved Fuzzy Logic algorithms are shown in Figure 2.3. Note that for ease of viewing, only a single 24 hour time period is graphed for each home simulation.

Man, Travelling Retired Man, Teen boy 15 Minute Power Average

Man, Travelling Woman 15 Minute Power Average

Figure 2.3 – Home Profile Graphs showing unmanaged and managed loads.

2.4.1 GIFCO home profile

The set of appliances used, along with their maximum loads are listed in Table 2.1. These appliance profiles model those used in the GIFCO smart home, as well as from data collected by GIFCO team members in Year I of the GIFCO project.

Appliance Name	Max Load	Appliance Name	Max Load
Dishwasher	600	Refrigerator	200
Washer	1000	Oven	3400
Dryer	5000	Television	350
Living Room Light	300	Power Tool	1100
Living Room Fan	100	Stereo	230
Kitchen Light	300	Computer	300
Kitchen Fan	100	Blow Drier	1000
Range Small Burner 1	1200	HVAC	3000
Range Small Burner 2	1200	Water Heater	3000
Range Large Burner 1	1800	Range Hood	100
Range Large Burner 2	1800	Outdoor Lights	600
Toaster Oven	1300	Disposal	445
Microwave	900	Always On*	75

Table 2.1 - Common Appliance Load List

*Always On is a static load that represents various small power appliances in the homes. Clock Radios, Smoke Alarms, Night Lights are examples of loads represented by Always On.

3. Hydrogen Co-generation

In **Section 1.3**, hydrogen cogeneration was investigated as a possibility for fuel cell systems used in a 50-home mini-grid community. Two scenarios were evaluated. In **Case 1**, a community consisting of five clusters of ten family homes was considered. Each cluster of homes is served by a 50kW proton exchange membrane (PEM) type fuel cell. In **Case 2**, the same size community is served by a hybrid system consisting of a 175kW Solid-oxide fuel cell (SOFC) and a 75kW PEM.

3.1 Excess Hydrogen Generation

Case 1 – 250kW at 60% utilization as five independent 50kW fuel cell systems. The total excess hydrogen available would be enough to generate 100kW, corresponding to a flow rate of approximately 40ft³/min of hydrogen. The total daily production of hydrogen would be 57,600 ft³, or a total of approximately 145kg/day. As described in Section 1.3.1, this production is insufficient to use as a fuel source for automobiles, especially since the hydrogen would have to be purified and compressed to high pressure. The most economically attractive use is storage for later use.

Case 2 – 175kW SOFC and 75kW PEM system. The maximum available excess hydrogen would be the source feed for the 75kW PEM system when the PEM system is on standby. The hydrogen flow rate would be approximately 30ft³/min, or 43,200ft³/day. Again, the most economically attractive use is as a fuel source for the PEM system to meet peak demands.

3.2 Separating the PEM Fuel Cell Stack and the Reformer

For the purposes of this project, the only reformer technology considered is natural gas reforming. The major component of natural gas is generally methane, so the terms “methane reforming” and “natural gas reforming” are synonymous.

The University of South Alabama currently has a 5kW PEM fuel cell that is being tested as a source of electrical power for a typical residence. The unit operates on natural gas and has an integrated steam reforming system. Operational experience with the 5kW fuel cell indicates that the reformer operates most efficiently when in a steady state. The response of a reformer to step changes in electrical load is sluggish, due to the reaction kinetics of the multi-stage steam reforming reaction and the thermal load of the vessels in the reforming system. If the PEM cell stack and reformer were separate systems, the reformer could continue to produce the maximum amount of hydrogen during times when the fuel cell is not operating at full load. The cell stack would use hydrogen on electrical demand, while the reformer would continue to independently produce hydrogen at a constant rate for storage.

Currently, two major approaches to reforming are in use. Steam methane reforming is endothermic and requires energy input. Partial oxidation of methane is exothermic, but has a somewhat lower efficiency. A combination of the two methods is utilized by the 5kW PEM system, a process known as autothermal reforming. The three processes are described below [9].

3.2.1 Steam methane reforming

Widely used in industry to produce hydrogen, steam reforming of methane has been around for many years. The ammonia production industry and the petroleum refining industry are the two largest annual producers of hydrogen by steam reforming of methane. The key reaction, sometimes called oxygenolysis, is:

This reaction is endothermic, requiring that a hot (>500°C) nickel catalyst be present for rapid equilibrium. High temperatures and low pressures favor hydrogen production over the reverse reaction. The energy typically comes from the combustion of a portion of the feed methane.

One of the key problems with steam reforming is that large quantities of carbon monoxide (CO) are produced. As a result, in most industrial steam reforming processes,

the effluent from the steam reformer goes to a shift reactor, where most of the CO is converted to CO₂ and H₂ by the water-gas shift reaction at a temperature of 200-400°C

Although the exothermic water-gas shift reaction occurs in tandem with the steam reforming oxygenolysis reaction, the water-gas shift reaction is typically carried out in a separate vessel.

The overall product gas is a mixture of carbon monoxide, carbon dioxide, and hydrogen, with traces of water vapor and unconverted methane. Since carbon monoxide acts as a poison to PEM catalysts, the CO concentration must be further reduced before the effluent (known as reformat) can be used as fuel for the PEM fuel cell.

3.2.2 Partial oxidation reforming

An alternative process for hydrogen formation is exothermic and is known as partial oxidation reforming.

This reaction is carried out at high temperatures (1200 – 1500C) and requires neither a catalyst nor a source of water. The tradeoff is that less hydrogen is produced per mole of methane. In practice, either reaction, steam reforming or partial oxidation, produces roughly the same amount of hydrogen for the same methane feed.

3.2.3 Auto thermal reforming

In most practical methane reformers for use with PEM fuel cells, a combination of steam reforming and partial oxidation is used. Figure 3.1 illustrates a typical reformer.

Figure 3.1 – Typical autothermal reformer

The first step in the process is to remove sulfur-containing compounds, which act as poisons to the PEM membranes in a fuel cell stack. Sulfur in methane occurs naturally and is also deliberately added as an odorant.

Methane is then mixed with air in stoichiometric proportion of one mole of oxygen per two moles of methane. The gas stream then goes through a humidifier where it is humidified to an 84°C wet-bulb temperature. The resulting humidified gas is then heated to a temperature of 500°C. The steam to carbon ratio is in the range of 3.0 to 4.0 on a mole percent basis.

The hot mixture of steam, methane, and oxygen then enters the partial oxidation reactor, where the partial oxidation and the steam reforming reactions take place in tandem. The process is mildly exothermic, taking place over a supported nickel catalyst. The support is usually some type of alumino-silicate compound such as a zeolite.

The outlet stream of the partial oxidation reactor is cooled from approximately 700°C to approximately 400°C before entering a high-temperature shift reactor. The water-gas shift reaction reduced the CO content and enriches the hydrogen content. Another heat exchanger reduces the outlet stream temperature to approximately 220°C, where another stage of water-gas shift reaction is done. The purpose of using two reactors is to allow for better heat recovery and management.

A heat exchanger downstream of the shift reactor cools the gas stream to approximately 120°C, where two reactions, selective oxidation and methanization occur to further reduce the CO content of the stream.

The outlet of the final stage is called reformat, and is composed of a mixture of CH₄, H₂, N₂, CO₂, CO, and H₂O as steam. The hydrogen composition is approximately 40-45% and the CO concentration is <50ppm. The other components are inert as far as the PEM fuel cell membranes are concerned.

As can be seen from the above discussion, the technical challenges presented by separating the fuel processing system from the PEM fuel cell are not insurmountable. As of this writing, the fuel cell market for stand-alone power generation has resulted in a trend toward separate reforming/hydrogen generation systems and PEM fuel cell systems that can accept hydrogen from a variety of sources. Plug Power, Ballard Generation Systems, Sanyo Electric, Osaka Gas, and Chevron Texaco are all participating in efforts to design stand-alone reforming systems. It is initially more expensive to separate the two systems, but in USA's operational experience with a 5kW combined heat and power

generation system, the reformer is the source of the majority of maintenance problems. The advantages to separating the two systems include:

- Easier maintenance
- The ability to use multiple hydrogen sources for the PEM
- The ability to operate multiple PEM systems independently of their associated reformers

3.3 Hydrogen Storage Methods and Delivery Systems

The hydrogen produced by a typical reformer is at or near atmospheric pressure. As discussed in Section 1.3.1, the amount of hydrogen produced as surplus by five 50kW PEM fuel cell reformers would be insufficient to economically fuel automobiles. As a result, some means of on-site storage is desirable. Storage methods include compression, storage in metal hydrides, and liquefaction.

3.3.1 Mechanical compression

Hydrogen is a light fuel and must be stored at very high pressures (3000 – 5000 psig) to provide a useful capacity for power generation. Hydrogen at 4600 psig in a modern composite tank has about a 3% mass fraction [25]. For example, a tank weighing 100 kg would contain 3 kg of hydrogen at 4600 psig. The volume of 3 kg of high-pressure hydrogen at atmospheric pressure would be approximately 1200 ft³, or a sufficient quantity of hydrogen to operate a 50 kW fuel cell at full capacity for about 1 hour. The volume of 1200 ft³ of hydrogen at 4600 psig would be approximately 4 ft³.

For the multiple 50kW PEM system of **Case 1**, the amount of excess hydrogen produced in one day is 57,600ft³. At a pressure of 250psi, a tank with a diameter of 10ft and a length of 40ft would be required for storage. At a pressure of 4600psi, a tank with a diameter of 5ft and a length of 10ft would be sufficient. Although tank size and weight will not be the problem in fixed installations that it would be in automotive applications, the economics of storage and delivery must be considered.

Mechanical compression has an electrical efficiency of approximately 20% [9]. To achieve a pressure increase from atmospheric pressure at the reformer outlet to 5000 psi in a hydrogen storage vessel would require an appreciable fraction of the energy available from the hydrogen, either as heat or as electrical power. In addition, the reformat is low purity, so an additional energy penalty must be paid to either remove the inerts, or compress the inerts together with the hydrogen.

Other than the energy cost associated with compression, there are safety concerns. If the storage vessel is punctured, either deliberately or accidentally, the resulting release of hydrogen could be catastrophic. Hydrogen and air mixtures are explosive over a wide range of concentrations, and the resulting flames are colorless.

3.3.2 Metal hydride storage

A promising technology for hydrogen storage and delivery is the use of reversible metal hydrides. Certain metals, for example titanium and iron, reversibly react to form metal hydrides:

Hydrogen is supplied to the metal bed at a little above atmospheric pressure. The hydrating reaction proceeds rapidly, and is mildly exothermic, so some air cooling may be required. To release the hydrogen for use, the hydride bed is heated and the reaction proceeds to the left.

In terms of mass efficiency, metal hydrides are not very attractive, but in terms of volumetric efficiency, metal hydrides can achieve higher storage densities than liquid hydrogen [9]. For a fixed installation, metal hydride storage has the following advantages.

- Hydrogen is stored at low pressure
- If the integrity of the storage vessel is compromised, hydrogen release is slow and controlled
- Charging and discharging can be performed at low pressures
- The materials are relatively cheap
- The metal only adsorbs hydrogen, so the inert materials are removed from the reformat without complex purification

3.3.3 Liquefaction

The storage of hydrogen as a cryogenic liquid is currently the only widely-used method for storing large quantities of hydrogen. Petroleum refining, ammonia production, and NASA are the largest consumers.

Liquefaction of hydrogen is technically challenging and requires about 79 kJ/mol. This amounts to about 1/3 of the LHV (242 kJ/mol) of hydrogen, which means that 33% of the available energy is consumed by the liquefaction process [27]. In addition, there are two isomeric forms of hydrogen – ortho-hydrogen, where the individual atomic spins are the same, and para-hydrogen, where the spins are opposite. Long-term storage requires that all of the hydrogen be para, which takes additional energy.

The purification, compression, liquefaction, conversion and storage in cryogenic vessels are technical problems that can only be economically solved for production facilities that produce in excess of 1000 kg/hr [9]. Liquefaction is not feasible for the small quantities of hydrogen produced by a 250kW reforming process.

3.3.4 Exotic methods

There is currently some research in hydrogen storage with carbon nanotube materials that are capable of very dense concentrations at relatively low pressures [24]. This technology is very new and has not reached commercial viability. There are also chemical methods for the storage and production of hydrogen, such as sodium borohydride [26]. The technical problems and costs associated with these methods and the difficulty of adapting the methods to the storage of low-quality and low pressure hydrogen produced by a methane reformer are prohibitive.

3.4 Hydrogen Purification

It is well known that PEM fuel cells operate for longer periods of time with fewer maintenance problems when supplied with pure hydrogen. The hydrogen stream from a reformer typically consists of about 40 mol% hydrogen and about 50 PPM of carbon monoxide, with nitrogen, carbon dioxide, and methane making up the balance. The carbon monoxide acts as a poison to the fuel cell catalytic membrane. The nitrogen and carbon dioxide are inert diluents that must be transported through the system at considerable energy cost. There are a number of methods for hydrogen purification including permeable membranes, liquid stripping, and selective adsorption. Each method has advantages and disadvantages in terms of energy cost

The most common commercial purification method is pressure swing adsorption (PSA), where hydrogen is selectively adsorbed on a substrate from a low concentration stream and is released into a high concentration stream [25]. PSA systems are widely used to produce large quantities of hydrogen, but the same economies of scale apply to PSA as apply to liquefaction. For the small quantities of hydrogen from a 250kW reformer, PSA is not economically attractive.

The most promising purification method for the application in this project is selective adsorption and storage using metal hydrides. As stated above, many metals preferentially adsorb hydrogen, so the inert materials in the reformat stream do not present a problem. Water vapor can be removed by utilizing a zeolite scrubber that can be regenerated periodically using waste heat from the reformer. The block diagram of **Figure 3.2** illustrates a simplified suggested design for a metal hydride based purification and storage system.

The reformat from the reformer is dried in one of two zeolite beds. When one bed becomes saturated, the stream is routed to a dry bed and waste heat from the reformer is used to drive the water off of the saturated bed.

The dry reformat then enters a low-pressure compressor or blower where the pressure is increased to approximately two atmospheres. The stream then enters a metal hydride storage vessel, where the hydrogen is preferentially adsorbed.

Figure 3.2 – Block diagram of a hydrogen storage/purification system.

Heat from the reformer can be used to desorb hydrogen from the metal hydride bed on demand. Details missing from the diagram include the necessary heat exchangers to cycle the water removal beds and to desorb the hydrogen from the hydride bed. Also, some provision must be made for venting the inert materials, such as nitrogen, to the atmosphere. All of these are problems that can be solved by relatively straightforward application of existing technology.

4. Smart Energy Management at the Appliance Level

4.1 Appliance Selection

The range is chosen as an appliance example to demonstrate the feasibility of power control at the appliance level. The range is assumed to consist of four cooking elements located at the top, and two heating elements located inside the oven.

4.2 Power Control Selection

Power delivered to each element will be controlled individually. In addition, level of power delivered to each element will be controlled individually between 0% and 100% in a continuous manner using phase control method as explained next.

Each element will be connected in series with a high-power phase-control solid-state relay with zero-crossing turn off. The DC low-voltage control side of the solid-state relay will be controlled by a variable pulse-width signal to determine the turn-on time. Each pulse will be timed relative to a zero-crossing signal to turn on that solid-state relay. The delay time depends on the amount of power to be delivered during each half-cycle of the AC power source. The zero-crossing signal is generated from a stepped down AC signal that is derived from the AC power source.

Accordingly, the variable pulse-width signal turns on the solid-state relay after the occurrence of zero-crossing allowing current to flow into the corresponding element, and the solid-state relay turns itself off at the next zero-crossing. Thus, by controlling the instant at which the solid-state turns on, the amount of power delivered to the element can be controlled in continuous manner between 0% and 100%.

4.3 Hardware Development

Fig. 4.1 shows the block diagram of the main blocks in the appliance control system. A high-power phase-control solid-state relay with zero-crossing turn off will be used to regulate the amount of power delivered to each element. All six solid-state relays will be controlled on their DC low-voltage sides by the microprocessor based control system.

In addition, the microprocessor based control system will be connected to six knobs, where each knob provides an analog signal representing the requested amount of power for the corresponding element. The microprocessor based control system will receive these input requests from the user to adjust power delivered to any element, and will instruct the corresponding phase-control device to deliver the requested power level.

The microprocessor based controller of the range provides an Ethernet interface, which is used to communicate to a PC running LEMSYS (Local Energy Management SYStem). The controller will receive commands from LEMSYS, execute them, and provide corresponding status back to LEMSYS.

Refer to Appendix B for more details about the hardware specifications, schematics and layout of the microprocessor based controller of the range.

Fig. 4.1 Block diagram of the range power control system.

4.4 Protocol Development

The microprocessor based controller of the range communicates with a PC running LEMSYS (Local Energy Management SYStem) through its Ethernet interface. The controller receives commands from LEMSYS and provides the requested status and actions. Each manageable sub-element of an appliance is given a unique load id, and is treated just like any other appliance in the GIFCO home. The following table provides a list of commands and their corresponding responses.

Command	Meaning	Response
@Am~	Request number of loads in appliance m	Number of elements (6 for range) in appliance
@Bm~	Request total maximum kW of appliance m	Total maximum kW rating of appliance
@Cmn~	Request maximum kW of load n in appliance m	Maximum kW rating of load n in appliance
@Dm~	Request actual total power level % of appliance m	Actual total power level % of all loads of appliance
@Emn~	Request actual power level % of load n in appliance m	Actual power level % of load n in appliance

@Fmnp~	Set power level of load n to p% in appliance m	Change actual power level of load n to p%, and acknowledge by sending actual power level % of load n in appliance
--------	--	---

4.5 Algorithm Development

Following power-up, LEMSYS will look up the characteristics of each sub-element in each appliance, just like it does for every other appliance in the GIFCO home. During normal operation, LEMSYS will periodically request actual total power delivered to the range as a percentage of its maximum power rating, as well as actual power delivered to each element as a percentage of their corresponding maximum power ratings. LEMSYS will treat each element as a separate load, and will run its standard management algorithm developed during Years I and II of the project. LEMSYS may decide to keep the current power consumption, or may decide to reduce it. If it decides to reduce power delivered to a particular element, it will communicate the proper command to the microprocessor based controller to do so. The microprocessor based controller will execute that command, and report adjusted power level information back to LEMSYS.

Publications Resulting from This Project

- [1] M. Y. El-Sharkh, A. Rahman, M. Alam, P. C. Byrne, A. A. Sakla, T. Thomas, "A dynamic model for a stand-alone PEM fuel cell power plant for residential applications," *Journal of Power Sources*, 138(2004) 199-204.
- [2] M. Tanrioven and **M. S. Alam**, "Modeling, Control and Power Quality Evaluation of a PEM Fuel Cell Based Power Supply System for Residential Use," accepted for publication, *IEEE Industry Applications Magazine*, Vol. 41, 2005.
- [3] M. Y. El-Sharkh, A. Rahman, M. S. Alam, A. A. Sakla, P. C. Byrne and T. Thomas, "Analysis of Active and Reactive Power Control of a Stand-Alone PEM Fuel Cell Power Plant," *IEEE Transactions on Power Systems*, vol. 19, no. 4, pp. 2022-2028, November 2004.
- [4] M. Tanrioven and **M. S. Alam**, "Reliability Modeling and Analysis of Stand Alone PEM Fuel Cell Power Plants," *International Journal of Renewable Energy*, p. 264-278, 2005.
- [5] M. Y. El-Sharkh, M. Tanrioven, A. Rahman, M. S. Alam, "Cost Related Sensitivity Analysis for Optimal Operation of a Grid-Parallel PEM Fuel Cell Power Plant," Submitted for review, *Electric Power Components & Systems*.
- [6] M. Y. El-Sharkh, M. Tanrioven, A. Rahman, M. S. Alam, "A Study of Cost-Optimized Operation of a Grid-Parallel PEM Fuel Cell Power Plant," Accepted for publication, *IEEE Transactions on Power Systems*.
- [7] M. Tanrioven and **M. S. Alam**, "Impact of Load Management on Reliability Assessment of Grid Independent PEM Fuel Cell Power Plants," accepted for publication, *Journal of Power Sources*, 2005.
- [8] M. Y. El-Sharkh, M. Tanrioven, A. Rahman, M. S. Alam, "Impact of Combined Wind Energy and Hydrogen Production on the Optimal Operation of a Fuel Cell Power Plant Based CHP System," *IASME Transactions*, Issue 4, Volume 2, pp. 493-490, June 2005.
- [9] M. Y. El-Sharkh, M. Tanrioven, A. Rahman, M. S. Alam, "Impact of Hydrogen Production on Optimal Economic Operation of a Grid-Parallel PEM Fuel Cell Power Plant," *Journal of Power Sources*, 153, pp. 136-144, 2006.
- [10] M. Y. El-Sharkh, M. Tanrioven, A. Rahman, M. S. Alam, "Optimal Operation of a Wind and Fuel Cell Power Plant Based CHP System for Grid-Parallel Residential Micro-Grid," *ASME / WSEAS International Conference on Energy, Environment, Ecosystems and Sustainable Development*, Athens, Greece, July 12-14, 2005.

- [11] M. Y. El-Sharkh, M. Tanrioven, A. Rahman, M. S. Alam, "Impact of Recovered Thermal Energy Management strategies on Economical Operation of a Grid-Parallel PEM Fuel Cell Power Plant," Submitted for review, *Electric Power Systems Research*.
- [12] M. Y. El-Sharkh, M. Tanrioven, A. Rahman, M. S. Alam, "Economics of Hydrogen Management Strategies for the Optimal Operation of a Grid-Parallel PEM Fuel Cell Power Plant," Submitted for review, *Renewable Energy*.
- [13] M. Y. El-Sharkh, A. Rahman, M. S. Alam, "Evolutionary Programming-Based Methodology for Economical Output Power from PEM Fuel Cell for Micro-Grid Application," *Journal of Power Sources*, Vol. 139, Issues 1-2, p.p. 165-169, January 2005.
- [14] M. Y. El-Sharkh, A. Rahman, M. S. Alam, "Neural Networks Based Control of Active and Reactive Power of a Stand-Alone PEM Fuel Cell Power Plant," *Journal of Power Sources*, Volume 135, Pages 88-94, 2004.
- [15] M. Y. El-Sharkh, M. Tanrioven, A. Rahman, M. S. Alam, "Neural Network-Based Economic Model for Optimal Operation of a Grid-Parallel PEM Fuel Cell Power Plant" Under prepration.
- [16] M. Tanrioven and M. S. Alam, "Modeling and Power Quality Evaluation of a PEM Fuel Cell Based Power System," *Proceedings of the IEEE Conference on Industry Applications*," Vol. 4, p. 2808-2814, Seattle, Washington, October 2004. This paper received the *IEEE Industry Applications Society 2004 Committee Prize Paper Award*.

References

- [1] Reliability Test System Task Force of the Application of Probability Methods Subcommittee, "IEEE Reliability Test System," *IEEE Trans. Power Apparatus and Systems*, vol. PAS-98, no. 6, pp. 2047- 2054, 1979.
- [2] M. B. Gunes, "Investigation of a Fuel Cell Based Total Energy System for Residential Applications," Master of Science Thesis, Dept. Mechanical Engineering, Virginia Polytechnic Institute and State University, 2001.
- [3] NFPA 70 *The National Electric Code* © 2002, National Fire Protection Association, Quincy , MA 02269.
- [4] Technical Info., www.fuelcells.org , Transportation Fuel Cells.
- [5] Turan Gonen, "Electric Power Distribution System Engineering" *McGraw-Hill Education*, 1986.
- [6] M. Gilly, J. Farbu, C. Holley, "Fuel cell waste heat hot water system", Technical report, Thermal Systems Design, presented December 07, 2004, University of South Alabama Department of Mechanical Engineering.
- [7] K. Hassman, "SOFC power plants, Westinghouse approach", *Fuel Cells*, **1**(1), 78-79, 2001.
- [8] T. Kreutz, J. Ogden, "Assessment of hydrogen-fueled PEM fuel cells for distributed generation and cogeneration", Final Report to the United States Department of Energy Hydrogen R&D Program, Contract No. DE-FG01-99EE35100, Center for Energy and Environmental Studies, Princeton University, Princeton, NJ 2000.
- [9] J. Larminie, A. Dicks, "Fuel cell systems explained, 2nd edition", John Wiley and Sons, West Sussex, UK, 2003. ISBN 0-470-84857-X.
- [10] C. Mescal, "*SOFC-PEM hybrid fuel cell systems*", ETSU Report F/03/00177/REP, AEA Technology, Harwell, UK, 1999.
- [11] W. Reyonlds, H. Perkins, "Engineering thermodynamics, 2nd edition", McGraw Hill, New York, 1977. ISBN 0-07-052046-1.
- [12] M. Y. El-Sharkh, A. Rahman, M. S. Alam, "Evolutionary Programming-Based Methodology for Economical Output Power from PEM Fuel Cell for Micro-Grid Application," *Journal of Power Sources*, vol. 139, no. 1-2, pp. 165-169, 2005.
- [13] S. Ahmed M. Azmy, and István Erlich, "Online Optimal Management of PEM Fuel Cells Using Neural Networks," *IEEE Transactions on Power Delivery*, vol. 20, no. 2, pp. 1051-1058, April 2005.
- [14] F. Barbir and T. Gomez, "Efficiency and Economics of Proton Exchange Membrane (PEM) Fuel Cell," *International Journal of Hydrogen Energy*, vol. 21, no. 10, pp. 891-901, 1996.
- [15] Y. Zoka, H. Sasaki, J. Kubokawa, R. Yokoyama, and H. Tanaka, "An optimal deployment of fuel cells in distribution systems by using genetic algorithms," *IEEE International Conference on Evolutionary Computation*, vol. 1, pp. 479-484, 1996.
- [16] George Erdmann, "Future economics of the fuel cell housing market," *International Journal of Hydrogen Energy*, no. 28, pp. 685– 694, 2003.
- [17] M. Y. El-Sharkh, A. A. El-Keib, "Maintenance Scheduling of Power System Generation and Transmission Using Fuzzy Evolutionary Programming," *IEEE Transactions on Power Systems*, vol. 18, no. 2, May 2003

- [18] Mehmet Burak Gunes, "Investigation of a Fuel Cell Based Total Energy System for Residential Applications," Master of Science Thesis, Dept. Mechanical Engineering, Virginia Polytechnic Institute and State University, 2001.
- [19] D. B. Fogel "Evolutionary Computation toward a New Philosophy of Machine Intelligence" 2nd ed, Wiley-IEEE Press, 1999.
- [20] T. Back, U. Hammel, H. P. Schwefel, "Evolutionary Computation: Comments on the History and Current State" *IEEE Transactions on Evolutionary computation*, vol. 1, no. 1, pp. 3-17, April 1997.
- [21] V. Miranda, D. Srinivasan, L. M. Proenca, "Evolutionary Computation in Power System," *Electrical Power and Energy Systems*, vol. 20, no. 2, pp. 89-98, 1998.
- [22] P. H. Winston, "Artificial Intelligence" 3rd ed, Addison-Wesley Publishing Company. 1993.
- [23] Reliability Test System Task Force of the Application of Probability Methods Subcommittee, "IEEE Reliability Test System," *IEEE Trans. Power Apparatus and Systems*, vol. PAS-98, no. 6, pp. 2047- 2054, 1979.
- [24] A. Chambers, C. Park, R. Baker, N. Rodriguez, "Hydrogen storage in graphite nanofibers", *Journal of Physical Chemistry B*, **102**, 4253-4256, 1998.
- [25] G. Hoogers, "Fuel cell technology handbook", CRC Press, Boca Raton, FL, 2003. ISBN 0-8493-0877-1.
- [26] P. Koschany, "Hydrogen sources integrated in fuel cells", *The Fuel Cell Home EFCF Conference Proceedings*, 2001.
- [27] D. Reister, W. Strobl, "Current development and outlook for the hydrogen fueled car", *Hydrogen Energy Progress IX*, 1992.

Appendix A

Information and Logic Rules for the House, the Living Room (LR), and the Kitchen

Information and Logic Rules for the House

```
operatorset opset {  
and xfl.prod();  
or xfl.max();  
defuz xfl.CenterOfArea();  
}
```

```
type TAppliance [0,2] {  
Off xfl.singleton(0);  
On xfl.singleton(1);  
Override xfl.singleton(2);  
}
```

```
type TAppliance_pri [0,19] {  
very_low xfl.trapezoid(0,0.1,1,2);  
low xfl.triangle(1,2,3);  
average xfl.triangle(2,3,4);  
above_average xfl.triangle(3,5,7);  
high xfl.triangle(5,6.5,8.5);  
very_high xfl.trapezoid(8,8.5,9,11);  
}
```

```
type TAppliance_load_diff [-1000,1000] {  
very_low xfl.trapezoid(-10001,-1000,-400,-200);  
low xfl.triangle(-700,-250,0);  
perfect xfl.triangle(-100,0,100);  
high xfl.triangle(0,250,700);  
very_high xfl.trapezoid(200,600,1000,10001);  
}
```

```
type TRoomOCCUPIED [0,1] {  
NO xfl.singleton(0);  
YES xfl.singleton(1);  
}
```

```
type OUT_TAPPLIANCE [0,10] {  
NO_CHANGE xfl.triangle(0,1,2);  
NOT_PROMISING xfl.triangle(3,4,5);  
LESS_PROMISING xfl.triangle(4,5,6);  
NEUTRAL xfl.triangle(5,6,7);  
SLIGHTLY_PROMISING xfl.triangle(6,7,8);  
PROMISING xfl.triangle(7,8,9);  
VERY_PROMISING xfl.triangle(8,9,10);
```

}

```
rulebase global(TRoomOCCUPIED RoomOCCUPIED, TAppliance HVAC,
TAppliance_pri HVAC_pri, TAppliance_load_diff HVAC_load_diff,
TAppliance waterheater, TAppliance_pri waterheater_pri, TAppliance_load_diff
waterheater_load_diff,
TAppliance frontFloods, TAppliance_pri frontFloods_pri, TAppliance_load_diff
frontFloods_load_diff
:OUT_TAPPLIANCE HVAC_OUT ,
OUT_TAPPLIANCE waterheater_OUT ,
OUT_TAPPLIANCE frontFloods_OUT
) using opset {
if (HVAC == Off | HVAC == Override | RoomOCCUPIED == YES) -> HVAC_OUT
= NO_CHANGE ;
if (waterheater == Off | waterheater == Override | RoomOCCUPIED == YES) ->
waterheater_OUT = NO_CHANGE ;
if (frontFloods == Off | frontFloods == Override | RoomOCCUPIED == YES) ->
frontFloods_OUT = NO_CHANGE ;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == very_low &
HVAC_load_diff == very_low) -> HVAC_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == very_low &
HVAC_load_diff == low) -> HVAC_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == very_low &
HVAC_load_diff == perfect) -> HVAC_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == very_low &
HVAC_load_diff == high) -> HVAC_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == very_low &
HVAC_load_diff == very_high) -> HVAC_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == low &
HVAC_load_diff == very_low) -> HVAC_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == low &
HVAC_load_diff == low) -> HVAC_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == low &
HVAC_load_diff == perfect) -> HVAC_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == low &
HVAC_load_diff == high) -> HVAC_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == low &
HVAC_load_diff == very_high) -> HVAC_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == average &
HVAC_load_diff == very_low) -> HVAC_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == average &
HVAC_load_diff == low) -> HVAC_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == average &
HVAC_load_diff == perfect) -> HVAC_OUT = PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == average &
HVAC_load_diff == high) -> HVAC_OUT = NEUTRAL;
```

if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == average & HVAC_load_diff == very_high) -> HVAC_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == above_average & HVAC_load_diff == very_low) -> HVAC_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == above_average & HVAC_load_diff == low) -> HVAC_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == above_average & HVAC_load_diff == perfect) -> HVAC_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == above_average & HVAC_load_diff == high) -> HVAC_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == above_average & HVAC_load_diff == very_high) -> HVAC_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == high & HVAC_load_diff == very_low) -> HVAC_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == high & HVAC_load_diff == low) -> HVAC_OUT = PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == high & HVAC_load_diff == perfect) -> HVAC_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == high & HVAC_load_diff == high) -> HVAC_OUT = PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == high & HVAC_load_diff == very_high) -> HVAC_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == very_high & HVAC_load_diff == very_low) -> HVAC_OUT = PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == very_high & HVAC_load_diff == low) -> HVAC_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == very_high & HVAC_load_diff == perfect) -> HVAC_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == very_high & HVAC_load_diff == high) -> HVAC_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & HVAC == On & HVAC_pri == very_high & HVAC_load_diff == very_high) -> HVAC_OUT = PROMISING;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == very_low & waterheater_load_diff == very_low) -> waterheater_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == very_low & waterheater_load_diff == low) -> waterheater_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == very_low & waterheater_load_diff == perfect) -> waterheater_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == very_low & waterheater_load_diff == high) -> waterheater_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == very_low & waterheater_load_diff == very_high) -> waterheater_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == low & waterheater_load_diff == very_low) -> waterheater_OUT = NOT_PROMISING;

if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == low & waterheater_load_diff == low) -> waterheater_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == low & waterheater_load_diff == perfect) -> waterheater_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == low & waterheater_load_diff == high) -> waterheater_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == low & waterheater_load_diff == very_high) -> waterheater_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == average & waterheater_load_diff == very_low) -> waterheater_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == average & waterheater_load_diff == low) -> waterheater_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == average & waterheater_load_diff == perfect) -> waterheater_OUT = PROMISING;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == average & waterheater_load_diff == high) -> waterheater_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == average & waterheater_load_diff == very_high) -> waterheater_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == above_average & waterheater_load_diff == very_low) -> waterheater_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == above_average & waterheater_load_diff == low) -> waterheater_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == above_average & waterheater_load_diff == perfect) -> waterheater_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == above_average & waterheater_load_diff == high) -> waterheater_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == above_average & waterheater_load_diff == very_high) -> waterheater_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == high & waterheater_load_diff == very_low) -> waterheater_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == high & waterheater_load_diff == low) -> waterheater_OUT = PROMISING;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == high & waterheater_load_diff == perfect) -> waterheater_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == high & waterheater_load_diff == high) -> waterheater_OUT = PROMISING;

if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == high & waterheater_load_diff == very_high) -> waterheater_OUT = SLIGHTLY_PROMISING;
 if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == very_high & waterheater_load_diff == very_low) -> waterheater_OUT = PROMISING;
 if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == very_high & waterheater_load_diff == low) -> waterheater_OUT = VERY_PROMISING;
 if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == very_high & waterheater_load_diff == perfect) -> waterheater_OUT = VERY_PROMISING;
 if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == very_high & waterheater_load_diff == high) -> waterheater_OUT = VERY_PROMISING;
 if (RoomOCCUPIED == NO & waterheater == On & waterheater_pri == very_high & waterheater_load_diff == very_high) -> waterheater_OUT = PROMISING;
 if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == very_low & frontFloods_load_diff == very_low) -> frontFloods_OUT = NOT_PROMISING;
 if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == very_low & frontFloods_load_diff == low) -> frontFloods_OUT = NOT_PROMISING;
 if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == very_low & frontFloods_load_diff == perfect) -> frontFloods_OUT = NEUTRAL;
 if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == very_low & frontFloods_load_diff == high) -> frontFloods_OUT = NOT_PROMISING;
 if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == very_low & frontFloods_load_diff == very_high) -> frontFloods_OUT = NOT_PROMISING;
 if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == low & frontFloods_load_diff == very_low) -> frontFloods_OUT = NOT_PROMISING;
 if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == low & frontFloods_load_diff == low) -> frontFloods_OUT = LESS_PROMISING;
 if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == low & frontFloods_load_diff == perfect) -> frontFloods_OUT = SLIGHTLY_PROMISING;
 if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == low & frontFloods_load_diff == high) -> frontFloods_OUT = LESS_PROMISING;
 if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == low & frontFloods_load_diff == very_high) -> frontFloods_OUT = NOT_PROMISING;
 if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == average & frontFloods_load_diff == very_low) -> frontFloods_OUT = LESS_PROMISING;
 if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == average & frontFloods_load_diff == low) -> frontFloods_OUT = NEUTRAL;
 if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == average & frontFloods_load_diff == perfect) -> frontFloods_OUT = PROMISING;

```

if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == average &
frontFloods_load_diff == high) -> frontFloods_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == average &
frontFloods_load_diff == very_high) -> frontFloods_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri ==
above_average & frontFloods_load_diff == very_low) -> frontFloods_OUT =
NEUTRAL;
if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri ==
above_average & frontFloods_load_diff == low) -> frontFloods_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri ==
above_average & frontFloods_load_diff == perfect) -> frontFloods_OUT =
VERY_PROMISING;
if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri ==
above_average & frontFloods_load_diff == high) -> frontFloods_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri ==
above_average & frontFloods_load_diff == very_high) -> frontFloods_OUT =
NEUTRAL;
if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == high &
frontFloods_load_diff == very_low) -> frontFloods_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == high &
frontFloods_load_diff == low) -> frontFloods_OUT = PROMISING;
if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == high &
frontFloods_load_diff == perfect) -> frontFloods_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == high &
frontFloods_load_diff == high) -> frontFloods_OUT = PROMISING;
if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == high &
frontFloods_load_diff == very_high) -> frontFloods_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == very_high
& frontFloods_load_diff == very_low) -> frontFloods_OUT = PROMISING;
if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == very_high
& frontFloods_load_diff == low) -> frontFloods_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == very_high
& frontFloods_load_diff == perfect) -> frontFloods_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == very_high
& frontFloods_load_diff == high) -> frontFloods_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & frontFloods == On & frontFloods_pri == very_high
& frontFloods_load_diff == very_high) -> frontFloods_OUT = PROMISING;
}

```

```

system(TRoomOCCUPIED RoomOCCUPIED, TAppliance HVAC,
TAppliance_pri HVAC_pri, TAppliance_load_diff HVAC_load_diff,

```

```

TAppliance waterheater, TAppliance_pri waterheater_pri, TAppliance_load_diff
waterheater_load_diff,
TAppliance frontFloods, TAppliance_pri frontFloods_pri, TAppliance_load_diff
frontFloods_load_diff
:OUT_TAPPLIANCE HVAC_OUT ,
OUT_TAPPLIANCE waterheater_OUT ,
OUT_TAPPLIANCE frontFloods_OUT
) {
global(RoomOCCUPIED, HVAC, HVAC_pri, HVAC_load_diff,
waterheater, waterheater_pri, waterheater_load_diff,
frontFloods, frontFloods_pri, frontFloods_load_diff
 : HVAC_OUT ,
waterheater_OUT ,
frontFloods_OUT
) ;}

```

Information and Logic Rules for the Living Room (LR)

```
operatorset opset {  
and xfl.prod();  
or xfl.max();  
defuz xfl.CenterOfArea();  
}
```

```
type TAppliance [0,2] {  
Off xfl.singleton(0);  
On xfl.singleton(1);  
Override xfl.singleton(2);  
}
```

```
type TAppliance_pri [0,19] {  
very_low xfl.trapezoid(0,0.1,1,2);  
low xfl.triangle(1,2,3);  
average xfl.triangle(2,3,4);  
above_average xfl.triangle(3,5,7);  
high xfl.triangle(5,6.5,8.5);  
very_high xfl.trapezoid(8,8.5,9,11);  
}
```

```
type TAppliance_load_diff [-1000,1000] {  
very_low xfl.trapezoid(-1000,-1000,-400,-200);  
low xfl.triangle(-700,-250,0);  
perfect xfl.triangle(-100,0,100);  
high xfl.triangle(0,250,700);  
very_high xfl.trapezoid(200,600,1000,1000);  
}
```

```
type TRoomOCCUPIED [0,1] {  
NO xfl.singleton(0);  
YES xfl.singleton(1);  
}
```

```
type OUT_TAPPLIANCE [0,10] {  
NO_CHANGE xfl.triangle(0,1,2);  
NOT_PROMISING xfl.triangle(3,4,5);  
LESS_PROMISING xfl.triangle(4,5,6);  
NEUTRAL xfl.triangle(5,6,7);  
SLIGHTLY_PROMISING xfl.triangle(6,7,8);  
PROMISING xfl.triangle(7,8,9);  
VERY_PROMISING xfl.triangle(8,9,10);
```

}

```
rulebase global(TRoomOCCUPIED RoomOCCUPIED, TAppliance LRLight,
TAppliance_pri LRLight_pri, TAppliance_load_diff LRLight_load_diff,
TAppliance LRFan, TAppliance_pri LRFan_pri, TAppliance_load_diff
LRFan_load_diff,
TAppliance tv1Big, TAppliance_pri tv1Big_pri, TAppliance_load_diff
tv1Big_load_diff,
TAppliance powertool1, TAppliance_pri powertool1_pri, TAppliance_load_diff
powertool1_load_diff,
TAppliance stereo, TAppliance_pri stereo_pri, TAppliance_load_diff
stereo_load_diff,
TAppliance computer, TAppliance_pri computer_pri, TAppliance_load_diff
computer_load_diff,
TAppliance alwayson, TAppliance_pri alwayson_pri, TAppliance_load_diff
alwayson_load_diff
:OUT_TAPPLIANCE LRLight_OUT ,
OUT_TAPPLIANCE LRFan_OUT ,
OUT_TAPPLIANCE tv1Big_OUT ,
OUT_TAPPLIANCE powertool1_OUT ,
OUT_TAPPLIANCE stereo_OUT ,
OUT_TAPPLIANCE computer_OUT ,
OUT_TAPPLIANCE alwayson_OUT
) using opset {
if (LRLight == Off | LRLight == Override | RoomOCCUPIED == YES) ->
LRLight_OUT = NO_CHANGE ;
if (LRFan == Off | LRFan == Override | RoomOCCUPIED == YES) ->
LRFan_OUT = NO_CHANGE ;
if (tv1Big == Off | tv1Big == Override | RoomOCCUPIED == YES) -> tv1Big_OUT
= NO_CHANGE ;
if (powertool1 == Off | powertool1 == Override | RoomOCCUPIED == YES) ->
powertool1_OUT = NO_CHANGE ;
if (stereo == Off | stereo == Override | RoomOCCUPIED == YES) -> stereo_OUT =
NO_CHANGE ;
if (computer == Off | computer == Override | RoomOCCUPIED == YES) ->
computer_OUT = NO_CHANGE ;
if (alwayson == Off | alwayson == Override | RoomOCCUPIED == YES) ->
alwayson_OUT = NO_CHANGE ;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == very_low &
LRLight_load_diff == very_low) -> LRLight_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == very_low &
LRLight_load_diff == low) -> LRLight_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == very_low &
LRLight_load_diff == perfect) -> LRLight_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == very_low &
LRLight_load_diff == high) -> LRLight_OUT = NOT_PROMISING;
```

if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == very_low & LRLight_load_diff == very_high) -> LRLight_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == low & LRLight_load_diff == very_low) -> LRLight_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == low & LRLight_load_diff == low) -> LRLight_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == low & LRLight_load_diff == perfect) -> LRLight_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == low & LRLight_load_diff == high) -> LRLight_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == low & LRLight_load_diff == very_high) -> LRLight_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == average & LRLight_load_diff == very_low) -> LRLight_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == average & LRLight_load_diff == low) -> LRLight_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == average & LRLight_load_diff == perfect) -> LRLight_OUT = PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == average & LRLight_load_diff == high) -> LRLight_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == average & LRLight_load_diff == very_high) -> LRLight_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == above_average & LRLight_load_diff == very_low) -> LRLight_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == above_average & LRLight_load_diff == low) -> LRLight_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == above_average & LRLight_load_diff == perfect) -> LRLight_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == above_average & LRLight_load_diff == high) -> LRLight_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == above_average & LRLight_load_diff == very_high) -> LRLight_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == high & LRLight_load_diff == very_low) -> LRLight_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == high & LRLight_load_diff == low) -> LRLight_OUT = PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == high & LRLight_load_diff == perfect) -> LRLight_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == high & LRLight_load_diff == high) -> LRLight_OUT = PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == high & LRLight_load_diff == very_high) -> LRLight_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == very_high & LRLight_load_diff == very_low) -> LRLight_OUT = PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == very_high & LRLight_load_diff == low) -> LRLight_OUT = VERY_PROMISING;

if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == very_high & LRLight_load_diff == perfect) -> LRLight_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == very_high & LRLight_load_diff == high) -> LRLight_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & LRLight == On & LRLight_pri == very_high & LRLight_load_diff == very_high) -> LRLight_OUT = PROMISING;
if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == very_low & LRFan_load_diff == very_low) -> LRFan_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == very_low & LRFan_load_diff == low) -> LRFan_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == very_low & LRFan_load_diff == perfect) -> LRFan_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == very_low & LRFan_load_diff == high) -> LRFan_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == very_low & LRFan_load_diff == very_high) -> LRFan_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == low & LRFan_load_diff == very_low) -> LRFan_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == low & LRFan_load_diff == low) -> LRFan_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == low & LRFan_load_diff == perfect) -> LRFan_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == low & LRFan_load_diff == high) -> LRFan_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == low & LRFan_load_diff == very_high) -> LRFan_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == average & LRFan_load_diff == very_low) -> LRFan_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == average & LRFan_load_diff == low) -> LRFan_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == average & LRFan_load_diff == perfect) -> LRFan_OUT = PROMISING;
if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == average & LRFan_load_diff == high) -> LRFan_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == average & LRFan_load_diff == very_high) -> LRFan_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == above_average & LRFan_load_diff == very_low) -> LRFan_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == above_average & LRFan_load_diff == low) -> LRFan_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == above_average & LRFan_load_diff == perfect) -> LRFan_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == above_average & LRFan_load_diff == high) -> LRFan_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == above_average & LRFan_load_diff == very_high) -> LRFan_OUT = NEUTRAL;

if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == high &
 LRFan_load_diff == very_low) -> LRFan_OUT = SLIGHTLY_PROMISING;
 if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == high &
 LRFan_load_diff == low) -> LRFan_OUT = PROMISING;
 if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == high &
 LRFan_load_diff == perfect) -> LRFan_OUT = VERY_PROMISING;
 if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == high &
 LRFan_load_diff == high) -> LRFan_OUT = PROMISING;
 if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == high &
 LRFan_load_diff == very_high) -> LRFan_OUT = SLIGHTLY_PROMISING;
 if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == very_high &
 LRFan_load_diff == very_low) -> LRFan_OUT = PROMISING;
 if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == very_high &
 LRFan_load_diff == low) -> LRFan_OUT = VERY_PROMISING;
 if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == very_high &
 LRFan_load_diff == perfect) -> LRFan_OUT = VERY_PROMISING;
 if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == very_high &
 LRFan_load_diff == high) -> LRFan_OUT = VERY_PROMISING;
 if (RoomOCCUPIED == NO & LRFan == On & LRFan_pri == very_high &
 LRFan_load_diff == very_high) -> LRFan_OUT = PROMISING;
 if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == very_low &
 tv1Big_load_diff == very_low) -> tv1Big_OUT = NOT_PROMISING;
 if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == very_low &
 tv1Big_load_diff == low) -> tv1Big_OUT = NOT_PROMISING;
 if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == very_low &
 tv1Big_load_diff == perfect) -> tv1Big_OUT = NEUTRAL;
 if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == very_low &
 tv1Big_load_diff == high) -> tv1Big_OUT = NOT_PROMISING;
 if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == very_low &
 tv1Big_load_diff == very_high) -> tv1Big_OUT = NOT_PROMISING;
 if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == low &
 tv1Big_load_diff == very_low) -> tv1Big_OUT = NOT_PROMISING;
 if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == low &
 tv1Big_load_diff == low) -> tv1Big_OUT = LESS_PROMISING;
 if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == low &
 tv1Big_load_diff == perfect) -> tv1Big_OUT = SLIGHTLY_PROMISING;
 if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == low &
 tv1Big_load_diff == high) -> tv1Big_OUT = LESS_PROMISING;
 if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == low &
 tv1Big_load_diff == very_high) -> tv1Big_OUT = NOT_PROMISING;
 if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == average &
 tv1Big_load_diff == very_low) -> tv1Big_OUT = LESS_PROMISING;
 if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == average &
 tv1Big_load_diff == low) -> tv1Big_OUT = NEUTRAL;
 if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == average &
 tv1Big_load_diff == perfect) -> tv1Big_OUT = PROMISING;

```

if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == average &
tv1Big_load_diff == high) -> tv1Big_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == average &
tv1Big_load_diff == very_high) -> tv1Big_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == above_average &
tv1Big_load_diff == very_low) -> tv1Big_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == above_average &
tv1Big_load_diff == low) -> tv1Big_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == above_average &
tv1Big_load_diff == perfect) -> tv1Big_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == above_average &
tv1Big_load_diff == high) -> tv1Big_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == above_average &
tv1Big_load_diff == very_high) -> tv1Big_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == high &
tv1Big_load_diff == very_low) -> tv1Big_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == high &
tv1Big_load_diff == low) -> tv1Big_OUT = PROMISING;
if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == high &
tv1Big_load_diff == perfect) -> tv1Big_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == high &
tv1Big_load_diff == high) -> tv1Big_OUT = PROMISING;
if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == high &
tv1Big_load_diff == very_high) -> tv1Big_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == very_high &
tv1Big_load_diff == very_low) -> tv1Big_OUT = PROMISING;
if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == very_high &
tv1Big_load_diff == low) -> tv1Big_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == very_high &
tv1Big_load_diff == perfect) -> tv1Big_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == very_high &
tv1Big_load_diff == high) -> tv1Big_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & tv1Big == On & tv1Big_pri == very_high &
tv1Big_load_diff == very_high) -> tv1Big_OUT = PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == very_low
& powertool1_load_diff == very_low) -> powertool1_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == very_low
& powertool1_load_diff == low) -> powertool1_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == very_low
& powertool1_load_diff == perfect) -> powertool1_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == very_low
& powertool1_load_diff == high) -> powertool1_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == very_low
& powertool1_load_diff == very_high) -> powertool1_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == low &
powertool1_load_diff == very_low) -> powertool1_OUT = NOT_PROMISING;

```

```

if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == low &
powertool1_load_diff == low) -> powertool1_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == low &
powertool1_load_diff == perfect) -> powertool1_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == low &
powertool1_load_diff == high) -> powertool1_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == low &
powertool1_load_diff == very_high) -> powertool1_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == average &
powertool1_load_diff == very_low) -> powertool1_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == average &
powertool1_load_diff == low) -> powertool1_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == average &
powertool1_load_diff == perfect) -> powertool1_OUT = PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == average &
powertool1_load_diff == high) -> powertool1_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == average &
powertool1_load_diff == very_high) -> powertool1_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri ==
above_average & powertool1_load_diff == very_low) -> powertool1_OUT =
NEUTRAL;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri ==
above_average & powertool1_load_diff == low) -> powertool1_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri ==
above_average & powertool1_load_diff == perfect) -> powertool1_OUT =
VERY_PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri ==
above_average & powertool1_load_diff == high) -> powertool1_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri ==
above_average & powertool1_load_diff == very_high) -> powertool1_OUT =
NEUTRAL;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == high &
powertool1_load_diff == very_low) -> powertool1_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == high &
powertool1_load_diff == low) -> powertool1_OUT = PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == high &
powertool1_load_diff == perfect) -> powertool1_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == high &
powertool1_load_diff == high) -> powertool1_OUT = PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == high &
powertool1_load_diff == very_high) -> powertool1_OUT =
SLIGHTLY_PROMISING;

```

if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == very_high & powertool1_load_diff == very_low) -> powertool1_OUT = PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == very_high & powertool1_load_diff == low) -> powertool1_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == very_high & powertool1_load_diff == perfect) -> powertool1_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == very_high & powertool1_load_diff == high) -> powertool1_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & powertool1 == On & powertool1_pri == very_high & powertool1_load_diff == very_high) -> powertool1_OUT = PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == very_low & stereo_load_diff == very_low) -> stereo_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == very_low & stereo_load_diff == low) -> stereo_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == very_low & stereo_load_diff == perfect) -> stereo_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == very_low & stereo_load_diff == high) -> stereo_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == very_low & stereo_load_diff == very_high) -> stereo_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == low & stereo_load_diff == very_low) -> stereo_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == low & stereo_load_diff == low) -> stereo_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == low & stereo_load_diff == perfect) -> stereo_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == low & stereo_load_diff == high) -> stereo_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == low & stereo_load_diff == very_high) -> stereo_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == average & stereo_load_diff == very_low) -> stereo_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == average & stereo_load_diff == low) -> stereo_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == average & stereo_load_diff == perfect) -> stereo_OUT = PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == average & stereo_load_diff == high) -> stereo_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == average & stereo_load_diff == very_high) -> stereo_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == above_average & stereo_load_diff == very_low) -> stereo_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == above_average & stereo_load_diff == low) -> stereo_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == above_average & stereo_load_diff == perfect) -> stereo_OUT = VERY_PROMISING;

if (RoomOCCUPIED == NO & stereo == On & stereo_pri == above_average & stereo_load_diff == high) -> stereo_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == above_average & stereo_load_diff == very_high) -> stereo_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == high & stereo_load_diff == very_low) -> stereo_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == high & stereo_load_diff == low) -> stereo_OUT = PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == high & stereo_load_diff == perfect) -> stereo_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == high & stereo_load_diff == high) -> stereo_OUT = PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == high & stereo_load_diff == very_high) -> stereo_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == very_high & stereo_load_diff == very_low) -> stereo_OUT = PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == very_high & stereo_load_diff == low) -> stereo_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == very_high & stereo_load_diff == perfect) -> stereo_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == very_high & stereo_load_diff == high) -> stereo_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & stereo == On & stereo_pri == very_high & stereo_load_diff == very_high) -> stereo_OUT = PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri == very_low & computer_load_diff == very_low) -> computer_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri == very_low & computer_load_diff == low) -> computer_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri == very_low & computer_load_diff == perfect) -> computer_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & computer == On & computer_pri == very_low & computer_load_diff == high) -> computer_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri == very_low & computer_load_diff == very_high) -> computer_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri == low & computer_load_diff == very_low) -> computer_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri == low & computer_load_diff == low) -> computer_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri == low & computer_load_diff == perfect) -> computer_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri == low & computer_load_diff == high) -> computer_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri == low & computer_load_diff == very_high) -> computer_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri == average & computer_load_diff == very_low) -> computer_OUT = LESS_PROMISING;

```

if (RoomOCCUPIED == NO & computer == On & computer_pri == average &
computer_load_diff == low) -> computer_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & computer == On & computer_pri == average &
computer_load_diff == perfect) -> computer_OUT = PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri == average &
computer_load_diff == high) -> computer_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & computer == On & computer_pri == average &
computer_load_diff == very_high) -> computer_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri ==
above_average & computer_load_diff == very_low) -> computer_OUT =
NEUTRAL;
if (RoomOCCUPIED == NO & computer == On & computer_pri ==
above_average & computer_load_diff == low) -> computer_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri ==
above_average & computer_load_diff == perfect) -> computer_OUT =
VERY_PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri ==
above_average & computer_load_diff == high) -> computer_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri ==
above_average & computer_load_diff == very_high) -> computer_OUT =
NEUTRAL;
if (RoomOCCUPIED == NO & computer == On & computer_pri == high &
computer_load_diff == very_low) -> computer_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri == high &
computer_load_diff == low) -> computer_OUT = PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri == high &
computer_load_diff == perfect) -> computer_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri == high &
computer_load_diff == high) -> computer_OUT = PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri == high &
computer_load_diff == very_high) -> computer_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri == very_high &
computer_load_diff == very_low) -> computer_OUT = PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri == very_high &
computer_load_diff == low) -> computer_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri == very_high &
computer_load_diff == perfect) -> computer_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri == very_high &
computer_load_diff == high) -> computer_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & computer == On & computer_pri == very_high &
computer_load_diff == very_high) -> computer_OUT = PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == very_low &
alwayson_load_diff == very_low) -> alwayson_OUT = NOT_PROMISING;

```

if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == very_low & alwayson_load_diff == low) -> alwayson_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == very_low & alwayson_load_diff == perfect) -> alwayson_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == very_low & alwayson_load_diff == high) -> alwayson_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == very_low & alwayson_load_diff == very_high) -> alwayson_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == low & alwayson_load_diff == very_low) -> alwayson_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == low & alwayson_load_diff == low) -> alwayson_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == low & alwayson_load_diff == perfect) -> alwayson_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == low & alwayson_load_diff == high) -> alwayson_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == low & alwayson_load_diff == very_high) -> alwayson_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == average & alwayson_load_diff == very_low) -> alwayson_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == average & alwayson_load_diff == low) -> alwayson_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == average & alwayson_load_diff == perfect) -> alwayson_OUT = PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == average & alwayson_load_diff == high) -> alwayson_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == average & alwayson_load_diff == very_high) -> alwayson_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == above_average & alwayson_load_diff == very_low) -> alwayson_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == above_average & alwayson_load_diff == low) -> alwayson_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == above_average & alwayson_load_diff == perfect) -> alwayson_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == above_average & alwayson_load_diff == high) -> alwayson_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == above_average & alwayson_load_diff == very_high) -> alwayson_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == high & alwayson_load_diff == very_low) -> alwayson_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == high & alwayson_load_diff == low) -> alwayson_OUT = PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == high & alwayson_load_diff == perfect) -> alwayson_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == high & alwayson_load_diff == high) -> alwayson_OUT = PROMISING;

```

if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == high &
alwayson_load_diff == very_high) -> alwayson_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == very_high &
alwayson_load_diff == very_low) -> alwayson_OUT = PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == very_high &
alwayson_load_diff == low) -> alwayson_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == very_high &
alwayson_load_diff == perfect) -> alwayson_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == very_high &
alwayson_load_diff == high) -> alwayson_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & alwayson == On & alwayson_pri == very_high &
alwayson_load_diff == very_high) -> alwayson_OUT = PROMISING;
}

```

```

system(TRoomOCCUPIED RoomOCCUPIED, TAppliance LRLight,
TAppliance_pri LRLight_pri, TAppliance_load_diff LRLight_load_diff,
TAppliance LRFan, TAppliance_pri LRFan_pri, TAppliance_load_diff
LRFan_load_diff,
TAppliance tv1Big, TAppliance_pri tv1Big_pri, TAppliance_load_diff
tv1Big_load_diff,
TAppliance powertool1, TAppliance_pri powertool1_pri, TAppliance_load_diff
powertool1_load_diff,
TAppliance stereo, TAppliance_pri stereo_pri, TAppliance_load_diff
stereo_load_diff,
TAppliance computer, TAppliance_pri computer_pri, TAppliance_load_diff
computer_load_diff,
TAppliance alwayson, TAppliance_pri alwayson_pri, TAppliance_load_diff
alwayson_load_diff
:OUT_TAPPLIANCE LRLight_OUT ,
OUT_TAPPLIANCE LRFan_OUT ,
OUT_TAPPLIANCE tv1Big_OUT ,
OUT_TAPPLIANCE powertool1_OUT ,
OUT_TAPPLIANCE stereo_OUT ,
OUT_TAPPLIANCE computer_OUT ,
OUT_TAPPLIANCE alwayson_OUT
){
global(RoomOCCUPIED, LRLight, LRLight_pri, LRLight_load_diff,
LRFan, LRFan_pri, LRFan_load_diff,
tv1Big, tv1Big_pri, tv1Big_load_diff,
powertool1, powertool1_pri, powertool1_load_diff,
stereo, stereo_pri, stereo_load_diff,
computer, computer_pri, computer_load_diff,
alwayson, alwayson_pri, alwayson_load_diff
: LRLight_OUT ,
LRFan_OUT ,
tv1Big_OUT ,

```



```
powertool1_OUT ,  
stereo_OUT ,  
computer_OUT ,  
alwayson_OUT  
) ;}
```

Information and Logic Rules for the Kitchen

```
operatorset opset {  
and xfl.prod();  
or xfl.max();  
defuz xfl.CenterOfArea();  
}
```

```
type TAppliance [0,2] {  
Off xfl.singleton(0);  
On xfl.singleton(1);  
Override xfl.singleton(2);  
}
```

```
type TAppliance_pri [0,19] {  
very_low xfl.trapezoid(0,0.1,1,2);  
low xfl.triangle(1,2,3);  
average xfl.triangle(2,3,4);  
above_average xfl.triangle(3,5,7);  
high xfl.triangle(5,6.5,8.5);  
very_high xfl.trapezoid(8,8.5,9,11);  
}
```

```
type TAppliance_load_diff [-1000,1000] {  
very_low xfl.trapezoid(-1000,-1000,-400,-200);  
low xfl.triangle(-700,-250,0);  
perfect xfl.triangle(-100,0,100);  
high xfl.triangle(0,250,700);  
very_high xfl.trapezoid(200,600,1000,1000);  
}
```

```
type TRoomOCCUPIED [0,1] {  
NO xfl.singleton(0);  
YES xfl.singleton(1);  
}
```

```
type OUT_TAPPLIANCE [0,10] {  
NO_CHANGE xfl.triangle(0,1,2);  
NOT_PROMISING xfl.triangle(3,4,5);  
LESS_PROMISING xfl.triangle(4,5,6);  
NEUTRAL xfl.triangle(5,6,7);  
SLIGHTLY_PROMISING xfl.triangle(6,7,8);  
PROMISING xfl.triangle(7,8,9);  
VERY_PROMISING xfl.triangle(8,9,10);
```

}

rulebase global(TRoomOCCUPIED RoomOCCUPIED, TAppliance dishwasher,
TAppliance_pri dishwasher_pri, TAppliance_load_diff dishwasher_load_diff,
TAppliance washer, TAppliance_pri washer_pri, TAppliance_load_diff
washer_load_diff,
TAppliance dryer, TAppliance_pri dryer_pri, TAppliance_load_diff
dryer_load_diff,
TAppliance kitchLight, TAppliance_pri kitchLight_pri, TAppliance_load_diff
kitchLight_load_diff,
TAppliance kitchFan, TAppliance_pri kitchFan_pri, TAppliance_load_diff
kitchFan_load_diff,
TAppliance smallRange1, TAppliance_pri smallRange1_pri,
TAppliance_load_diff smallRange1_load_diff,
TAppliance smallRange2, TAppliance_pri smallRange2_pri,
TAppliance_load_diff smallRange2_load_diff,
TAppliance largeRange1, TAppliance_pri largeRange1_pri,
TAppliance_load_diff largeRange1_load_diff,
TAppliance largeRange2, TAppliance_pri largeRange2_pri,
TAppliance_load_diff largeRange2_load_diff,
TAppliance toastOvn, TAppliance_pri toastOvn_pri, TAppliance_load_diff
toastOvn_load_diff,
TAppliance microwave, TAppliance_pri microwave_pri, TAppliance_load_diff
microwave_load_diff,
TAppliance refrig, TAppliance_pri refrig_pri, TAppliance_load_diff
refrig_load_diff,
TAppliance oven, TAppliance_pri oven_pri, TAppliance_load_diff
oven_load_diff,
TAppliance blowdrier1, TAppliance_pri blowdrier1_pri, TAppliance_load_diff
blowdrier1_load_diff,
TAppliance rangeFan, TAppliance_pri rangeFan_pri, TAppliance_load_diff
rangeFan_load_diff,
TAppliance disposal, TAppliance_pri disposal_pri, TAppliance_load_diff
disposal_load_diff
:OUT_TAPPLIANCE dishwasher_OUT ,
OUT_TAPPLIANCE washer_OUT ,
OUT_TAPPLIANCE dryer_OUT ,
OUT_TAPPLIANCE kitchLight_OUT ,
OUT_TAPPLIANCE kitchFan_OUT ,
OUT_TAPPLIANCE smallRange1_OUT ,
OUT_TAPPLIANCE smallRange2_OUT ,
OUT_TAPPLIANCE largeRange1_OUT ,
OUT_TAPPLIANCE largeRange2_OUT ,
OUT_TAPPLIANCE toastOvn_OUT ,
OUT_TAPPLIANCE microwave_OUT ,
OUT_TAPPLIANCE refrig_OUT ,

```

OUT_TAPPLIANCE oven_OUT ,
OUT_TAPPLIANCE blowdrier1_OUT ,
OUT_TAPPLIANCE rangeFan_OUT ,
OUT_TAPPLIANCE disposal_OUT
) using opset {
if (dishwasher == Off | dishwasher == Override | RoomOCCUPIED == YES) ->
dishwasher_OUT = NO_CHANGE ;
if (washer == Off | washer == Override | RoomOCCUPIED == YES) ->
washer_OUT = NO_CHANGE ;
if (dryer == Off | dryer == Override | RoomOCCUPIED == YES) -> dryer_OUT =
NO_CHANGE ;
if (kitchLight == Off | kitchLight == Override | RoomOCCUPIED == YES) ->
kitchLight_OUT = NO_CHANGE ;
if (kitchFan == Off | kitchFan == Override | RoomOCCUPIED == YES) ->
kitchFan_OUT = NO_CHANGE ;
if (smallRange1 == Off | smallRange1 == Override | RoomOCCUPIED == YES) ->
smallRange1_OUT = NO_CHANGE ;
if (smallRange2 == Off | smallRange2 == Override | RoomOCCUPIED == YES) ->
smallRange2_OUT = NO_CHANGE ;
if (largeRange1 == Off | largeRange1 == Override | RoomOCCUPIED == YES) ->
largeRange1_OUT = NO_CHANGE ;
if (largeRange2 == Off | largeRange2 == Override | RoomOCCUPIED == YES) ->
largeRange2_OUT = NO_CHANGE ;
if (toastOvn == Off | toastOvn == Override | RoomOCCUPIED == YES) ->
toastOvn_OUT = NO_CHANGE ;
if (microwave == Off | microwave == Override | RoomOCCUPIED == YES) ->
microwave_OUT = NO_CHANGE ;
if (refrig == Off | refrig == Override | RoomOCCUPIED == YES) -> refrig_OUT =
NO_CHANGE ;
if (oven == Off | oven == Override | RoomOCCUPIED == YES) -> oven_OUT =
NO_CHANGE ;
if (blowdrier1 == Off | blowdrier1 == Override | RoomOCCUPIED == YES) ->
blowdrier1_OUT = NO_CHANGE ;
if (rangeFan == Off | rangeFan == Override | RoomOCCUPIED == YES) ->
rangeFan_OUT = NO_CHANGE ;
if (disposal == Off | disposal == Override | RoomOCCUPIED == YES) ->
disposal_OUT = NO_CHANGE ;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == very_low
& dishwasher_load_diff == very_low) -> dishwasher_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == very_low
& dishwasher_load_diff == low) -> dishwasher_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == very_low
& dishwasher_load_diff == perfect) -> dishwasher_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == very_low
& dishwasher_load_diff == high) -> dishwasher_OUT = NOT_PROMISING;

```

```

if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == very_low
& dishwasher_load_diff == very_high) -> dishwasher_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == low &
dishwasher_load_diff == very_low) -> dishwasher_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == low &
dishwasher_load_diff == low) -> dishwasher_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == low &
dishwasher_load_diff == perfect) -> dishwasher_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == low &
dishwasher_load_diff == high) -> dishwasher_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == low &
dishwasher_load_diff == very_high) -> dishwasher_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == average &
dishwasher_load_diff == very_low) -> dishwasher_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == average &
dishwasher_load_diff == low) -> dishwasher_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == average &
dishwasher_load_diff == perfect) -> dishwasher_OUT = PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == average &
dishwasher_load_diff == high) -> dishwasher_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == average &
dishwasher_load_diff == very_high) -> dishwasher_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri ==
above_average & dishwasher_load_diff == very_low) -> dishwasher_OUT =
NEUTRAL;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri ==
above_average & dishwasher_load_diff == low) -> dishwasher_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri ==
above_average & dishwasher_load_diff == perfect) -> dishwasher_OUT =
VERY_PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri ==
above_average & dishwasher_load_diff == high) -> dishwasher_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri ==
above_average & dishwasher_load_diff == very_high) -> dishwasher_OUT =
NEUTRAL;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == high &
dishwasher_load_diff == very_low) -> dishwasher_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == high &
dishwasher_load_diff == low) -> dishwasher_OUT = PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == high &
dishwasher_load_diff == perfect) -> dishwasher_OUT = VERY_PROMISING;

```

if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == high & dishwasher_load_diff == high) -> dishwasher_OUT = PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == high & dishwasher_load_diff == very_high) -> dishwasher_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == very_high & dishwasher_load_diff == very_low) -> dishwasher_OUT = PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == very_high & dishwasher_load_diff == low) -> dishwasher_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == very_high & dishwasher_load_diff == perfect) -> dishwasher_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == very_high & dishwasher_load_diff == high) -> dishwasher_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & dishwasher == On & dishwasher_pri == very_high & dishwasher_load_diff == very_high) -> dishwasher_OUT = PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == very_low & washer_load_diff == very_low) -> washer_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == very_low & washer_load_diff == low) -> washer_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == very_low & washer_load_diff == perfect) -> washer_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & washer == On & washer_pri == very_low & washer_load_diff == high) -> washer_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == very_low & washer_load_diff == very_high) -> washer_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == low & washer_load_diff == very_low) -> washer_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == low & washer_load_diff == low) -> washer_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == low & washer_load_diff == perfect) -> washer_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == low & washer_load_diff == high) -> washer_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == low & washer_load_diff == very_high) -> washer_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == average & washer_load_diff == very_low) -> washer_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == average & washer_load_diff == low) -> washer_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & washer == On & washer_pri == average & washer_load_diff == perfect) -> washer_OUT = PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == average & washer_load_diff == high) -> washer_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & washer == On & washer_pri == average & washer_load_diff == very_high) -> washer_OUT = LESS_PROMISING;

if (RoomOCCUPIED == NO & washer == On & washer_pri == above_average & washer_load_diff == very_low) -> washer_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & washer == On & washer_pri == above_average & washer_load_diff == low) -> washer_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == above_average & washer_load_diff == perfect) -> washer_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == above_average & washer_load_diff == high) -> washer_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == above_average & washer_load_diff == very_high) -> washer_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & washer == On & washer_pri == high & washer_load_diff == very_low) -> washer_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == high & washer_load_diff == low) -> washer_OUT = PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == high & washer_load_diff == perfect) -> washer_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == high & washer_load_diff == high) -> washer_OUT = PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == high & washer_load_diff == very_high) -> washer_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == very_high & washer_load_diff == very_low) -> washer_OUT = PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == very_high & washer_load_diff == low) -> washer_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == very_high & washer_load_diff == perfect) -> washer_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == very_high & washer_load_diff == high) -> washer_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & washer == On & washer_pri == very_high & washer_load_diff == very_high) -> washer_OUT = PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == very_low & dryer_load_diff == very_low) -> dryer_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == very_low & dryer_load_diff == low) -> dryer_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == very_low & dryer_load_diff == perfect) -> dryer_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == very_low & dryer_load_diff == high) -> dryer_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == very_low & dryer_load_diff == very_high) -> dryer_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == low & dryer_load_diff == very_low) -> dryer_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == low & dryer_load_diff == low) -> dryer_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == low & dryer_load_diff == perfect) -> dryer_OUT = SLIGHTLY_PROMISING;

if (RoomOCCUPIED == NO & dryer == On & dryer_pri == low & dryer_load_diff == high) -> dryer_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == low & dryer_load_diff == very_high) -> dryer_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == average & dryer_load_diff == very_low) -> dryer_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == average & dryer_load_diff == low) -> dryer_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == average & dryer_load_diff == perfect) -> dryer_OUT = PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == average & dryer_load_diff == high) -> dryer_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == average & dryer_load_diff == very_high) -> dryer_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == above_average & dryer_load_diff == very_low) -> dryer_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == above_average & dryer_load_diff == low) -> dryer_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == above_average & dryer_load_diff == perfect) -> dryer_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == above_average & dryer_load_diff == high) -> dryer_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == above_average & dryer_load_diff == very_high) -> dryer_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == high & dryer_load_diff == very_low) -> dryer_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == high & dryer_load_diff == low) -> dryer_OUT = PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == high & dryer_load_diff == perfect) -> dryer_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == high & dryer_load_diff == high) -> dryer_OUT = PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == high & dryer_load_diff == very_high) -> dryer_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == very_high & dryer_load_diff == very_low) -> dryer_OUT = PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == very_high & dryer_load_diff == low) -> dryer_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == very_high & dryer_load_diff == perfect) -> dryer_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == very_high & dryer_load_diff == high) -> dryer_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & dryer == On & dryer_pri == very_high & dryer_load_diff == very_high) -> dryer_OUT = PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == very_low & kitchLight_load_diff == very_low) -> kitchLight_OUT = NOT_PROMISING;


```

if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == very_low &
kitchLight_load_diff == low) -> kitchLight_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == very_low &
kitchLight_load_diff == perfect) -> kitchLight_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == very_low &
kitchLight_load_diff == high) -> kitchLight_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == very_low &
kitchLight_load_diff == very_high) -> kitchLight_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == low &
kitchLight_load_diff == very_low) -> kitchLight_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == low &
kitchLight_load_diff == low) -> kitchLight_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == low &
kitchLight_load_diff == perfect) -> kitchLight_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == low &
kitchLight_load_diff == high) -> kitchLight_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == low &
kitchLight_load_diff == very_high) -> kitchLight_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == average &
kitchLight_load_diff == very_low) -> kitchLight_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == average &
kitchLight_load_diff == low) -> kitchLight_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == average &
kitchLight_load_diff == perfect) -> kitchLight_OUT = PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == average &
kitchLight_load_diff == high) -> kitchLight_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == average &
kitchLight_load_diff == very_high) -> kitchLight_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri ==
above_average & kitchLight_load_diff == very_low) -> kitchLight_OUT =
NEUTRAL;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri ==
above_average & kitchLight_load_diff == low) -> kitchLight_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri ==
above_average & kitchLight_load_diff == perfect) -> kitchLight_OUT =
VERY_PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri ==
above_average & kitchLight_load_diff == high) -> kitchLight_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri ==
above_average & kitchLight_load_diff == very_high) -> kitchLight_OUT =
NEUTRAL;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == high &
kitchLight_load_diff == very_low) -> kitchLight_OUT =
SLIGHTLY_PROMISING;

```

if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == high & kitchLight_load_diff == low) -> kitchLight_OUT = PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == high & kitchLight_load_diff == perfect) -> kitchLight_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == high & kitchLight_load_diff == high) -> kitchLight_OUT = PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == high & kitchLight_load_diff == very_high) -> kitchLight_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == very_high & kitchLight_load_diff == very_low) -> kitchLight_OUT = PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == very_high & kitchLight_load_diff == low) -> kitchLight_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == very_high & kitchLight_load_diff == perfect) -> kitchLight_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == very_high & kitchLight_load_diff == high) -> kitchLight_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & kitchLight == On & kitchLight_pri == very_high & kitchLight_load_diff == very_high) -> kitchLight_OUT = PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == very_low & kitchFan_load_diff == very_low) -> kitchFan_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == very_low & kitchFan_load_diff == low) -> kitchFan_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == very_low & kitchFan_load_diff == perfect) -> kitchFan_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == very_low & kitchFan_load_diff == high) -> kitchFan_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == very_low & kitchFan_load_diff == very_high) -> kitchFan_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == low & kitchFan_load_diff == very_low) -> kitchFan_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == low & kitchFan_load_diff == low) -> kitchFan_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == low & kitchFan_load_diff == perfect) -> kitchFan_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == low & kitchFan_load_diff == high) -> kitchFan_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == low & kitchFan_load_diff == very_high) -> kitchFan_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == average & kitchFan_load_diff == very_low) -> kitchFan_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == average & kitchFan_load_diff == low) -> kitchFan_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == average & kitchFan_load_diff == perfect) -> kitchFan_OUT = PROMISING;

if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == average & kitchFan_load_diff == high) -> kitchFan_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == average & kitchFan_load_diff == very_high) -> kitchFan_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == above_average & kitchFan_load_diff == very_low) -> kitchFan_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == above_average & kitchFan_load_diff == low) -> kitchFan_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == above_average & kitchFan_load_diff == perfect) -> kitchFan_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == above_average & kitchFan_load_diff == high) -> kitchFan_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == above_average & kitchFan_load_diff == very_high) -> kitchFan_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == high & kitchFan_load_diff == very_low) -> kitchFan_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == high & kitchFan_load_diff == low) -> kitchFan_OUT = PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == high & kitchFan_load_diff == perfect) -> kitchFan_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == high & kitchFan_load_diff == high) -> kitchFan_OUT = PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == high & kitchFan_load_diff == very_high) -> kitchFan_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == very_high & kitchFan_load_diff == very_low) -> kitchFan_OUT = PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == very_high & kitchFan_load_diff == low) -> kitchFan_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == very_high & kitchFan_load_diff == perfect) -> kitchFan_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == very_high & kitchFan_load_diff == high) -> kitchFan_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & kitchFan == On & kitchFan_pri == very_high & kitchFan_load_diff == very_high) -> kitchFan_OUT = PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == very_low & smallRange1_load_diff == very_low) -> smallRange1_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == very_low & smallRange1_load_diff == low) -> smallRange1_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == very_low & smallRange1_load_diff == perfect) -> smallRange1_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == very_low & smallRange1_load_diff == high) -> smallRange1_OUT = NOT_PROMISING;

```

if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri ==
very_low & smallRange1_load_diff == very_high) -> smallRange1_OUT =
NOT_PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == low &
smallRange1_load_diff == very_low) -> smallRange1_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == low &
smallRange1_load_diff == low) -> smallRange1_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == low &
smallRange1_load_diff == perfect) -> smallRange1_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == low &
smallRange1_load_diff == high) -> smallRange1_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == low &
smallRange1_load_diff == very_high) -> smallRange1_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == average
& smallRange1_load_diff == very_low) -> smallRange1_OUT =
LESS_PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == average
& smallRange1_load_diff == low) -> smallRange1_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == average
& smallRange1_load_diff == perfect) -> smallRange1_OUT = PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == average
& smallRange1_load_diff == high) -> smallRange1_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == average
& smallRange1_load_diff == very_high) -> smallRange1_OUT =
LESS_PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri ==
above_average & smallRange1_load_diff == very_low) -> smallRange1_OUT =
NEUTRAL;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri ==
above_average & smallRange1_load_diff == low) -> smallRange1_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri ==
above_average & smallRange1_load_diff == perfect) -> smallRange1_OUT =
VERY_PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri ==
above_average & smallRange1_load_diff == high) -> smallRange1_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri ==
above_average & smallRange1_load_diff == very_high) -> smallRange1_OUT =
NEUTRAL;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == high &
smallRange1_load_diff == very_low) -> smallRange1_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == high &
smallRange1_load_diff == low) -> smallRange1_OUT = PROMISING;

```

if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == high & smallRange1_load_diff == perfect) -> smallRange1_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == high & smallRange1_load_diff == high) -> smallRange1_OUT = PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == high & smallRange1_load_diff == very_high) -> smallRange1_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == very_high & smallRange1_load_diff == very_low) -> smallRange1_OUT = PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == very_high & smallRange1_load_diff == low) -> smallRange1_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == very_high & smallRange1_load_diff == perfect) -> smallRange1_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == very_high & smallRange1_load_diff == high) -> smallRange1_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & smallRange1 == On & smallRange1_pri == very_high & smallRange1_load_diff == very_high) -> smallRange1_OUT = PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri == very_low & smallRange2_load_diff == very_low) -> smallRange2_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri == very_low & smallRange2_load_diff == low) -> smallRange2_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri == very_low & smallRange2_load_diff == perfect) -> smallRange2_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri == very_low & smallRange2_load_diff == high) -> smallRange2_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri == very_low & smallRange2_load_diff == very_high) -> smallRange2_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri == low & smallRange2_load_diff == very_low) -> smallRange2_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri == low & smallRange2_load_diff == low) -> smallRange2_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri == low & smallRange2_load_diff == perfect) -> smallRange2_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri == low & smallRange2_load_diff == high) -> smallRange2_OUT = LESS_PROMISING;

```
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri == low &
smallRange2_load_diff == very_high) -> smallRange2_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri == average
& smallRange2_load_diff == very_low) -> smallRange2_OUT =
LESS_PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri == average
& smallRange2_load_diff == low) -> smallRange2_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri == average
& smallRange2_load_diff == perfect) -> smallRange2_OUT = PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri == average
& smallRange2_load_diff == high) -> smallRange2_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri == average
& smallRange2_load_diff == very_high) -> smallRange2_OUT =
LESS_PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri ==
above_average & smallRange2_load_diff == very_low) -> smallRange2_OUT =
NEUTRAL;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri ==
above_average & smallRange2_load_diff == low) -> smallRange2_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri ==
above_average & smallRange2_load_diff == perfect) -> smallRange2_OUT =
VERY_PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri ==
above_average & smallRange2_load_diff == high) -> smallRange2_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri ==
above_average & smallRange2_load_diff == very_high) -> smallRange2_OUT =
NEUTRAL;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri == high &
smallRange2_load_diff == very_low) -> smallRange2_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri == high &
smallRange2_load_diff == low) -> smallRange2_OUT = PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri == high &
smallRange2_load_diff == perfect) -> smallRange2_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri == high &
smallRange2_load_diff == high) -> smallRange2_OUT = PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri == high &
smallRange2_load_diff == very_high) -> smallRange2_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri ==
very_high & smallRange2_load_diff == very_low) -> smallRange2_OUT =
PROMISING;
```

```
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri ==
very_high & smallRange2_load_diff == low) -> smallRange2_OUT =
VERY_PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri ==
very_high & smallRange2_load_diff == perfect) -> smallRange2_OUT =
VERY_PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri ==
very_high & smallRange2_load_diff == high) -> smallRange2_OUT =
VERY_PROMISING;
if (RoomOCCUPIED == NO & smallRange2 == On & smallRange2_pri ==
very_high & smallRange2_load_diff == very_high) -> smallRange2_OUT =
PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri ==
very_low & largeRange1_load_diff == very_low) -> largeRange1_OUT =
NOT_PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri ==
very_low & largeRange1_load_diff == low) -> largeRange1_OUT =
NOT_PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri ==
very_low & largeRange1_load_diff == perfect) -> largeRange1_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri ==
very_low & largeRange1_load_diff == high) -> largeRange1_OUT =
NOT_PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri ==
very_low & largeRange1_load_diff == very_high) -> largeRange1_OUT =
NOT_PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == low &
largeRange1_load_diff == very_low) -> largeRange1_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == low &
largeRange1_load_diff == low) -> largeRange1_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == low &
largeRange1_load_diff == perfect) -> largeRange1_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == low &
largeRange1_load_diff == high) -> largeRange1_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == low &
largeRange1_load_diff == very_high) -> largeRange1_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == average
& largeRange1_load_diff == very_low) -> largeRange1_OUT =
LESS_PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == average
& largeRange1_load_diff == low) -> largeRange1_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == average
& largeRange1_load_diff == perfect) -> largeRange1_OUT = PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == average
& largeRange1_load_diff == high) -> largeRange1_OUT = NEUTRAL;
```

if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == average & largeRange1_load_diff == very_high) -> largeRange1_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == above_average & largeRange1_load_diff == very_low) -> largeRange1_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == above_average & largeRange1_load_diff == low) -> largeRange1_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == above_average & largeRange1_load_diff == perfect) -> largeRange1_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == above_average & largeRange1_load_diff == high) -> largeRange1_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == above_average & largeRange1_load_diff == very_high) -> largeRange1_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == high & largeRange1_load_diff == very_low) -> largeRange1_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == high & largeRange1_load_diff == low) -> largeRange1_OUT = PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == high & largeRange1_load_diff == perfect) -> largeRange1_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == high & largeRange1_load_diff == high) -> largeRange1_OUT = PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == high & largeRange1_load_diff == very_high) -> largeRange1_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == very_high & largeRange1_load_diff == very_low) -> largeRange1_OUT = PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == very_high & largeRange1_load_diff == low) -> largeRange1_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == very_high & largeRange1_load_diff == perfect) -> largeRange1_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == very_high & largeRange1_load_diff == high) -> largeRange1_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & largeRange1 == On & largeRange1_pri == very_high & largeRange1_load_diff == very_high) -> largeRange1_OUT = PROMISING;


```
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri ==
very_low & largeRange2_load_diff == very_low) -> largeRange2_OUT =
NOT_PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri ==
very_low & largeRange2_load_diff == low) -> largeRange2_OUT =
NOT_PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri ==
very_low & largeRange2_load_diff == perfect) -> largeRange2_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri ==
very_low & largeRange2_load_diff == high) -> largeRange2_OUT =
NOT_PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri ==
very_low & largeRange2_load_diff == very_high) -> largeRange2_OUT =
NOT_PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri == low &
largeRange2_load_diff == very_low) -> largeRange2_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri == low &
largeRange2_load_diff == low) -> largeRange2_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri == low &
largeRange2_load_diff == perfect) -> largeRange2_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri == low &
largeRange2_load_diff == high) -> largeRange2_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri == low &
largeRange2_load_diff == very_high) -> largeRange2_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri == average
& largeRange2_load_diff == very_low) -> largeRange2_OUT =
LESS_PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri == average
& largeRange2_load_diff == low) -> largeRange2_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri == average
& largeRange2_load_diff == perfect) -> largeRange2_OUT = PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri == average
& largeRange2_load_diff == high) -> largeRange2_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri == average
& largeRange2_load_diff == very_high) -> largeRange2_OUT =
LESS_PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri ==
above_average & largeRange2_load_diff == very_low) -> largeRange2_OUT =
NEUTRAL;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri ==
above_average & largeRange2_load_diff == low) -> largeRange2_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri ==
above_average & largeRange2_load_diff == perfect) -> largeRange2_OUT =
VERY_PROMISING;
```

```

if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri ==
above_average & largeRange2_load_diff == high) -> largeRange2_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri ==
above_average & largeRange2_load_diff == very_high) -> largeRange2_OUT =
NEUTRAL;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri == high &
largeRange2_load_diff == very_low) -> largeRange2_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri == high &
largeRange2_load_diff == low) -> largeRange2_OUT = PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri == high &
largeRange2_load_diff == perfect) -> largeRange2_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri == high &
largeRange2_load_diff == high) -> largeRange2_OUT = PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri == high &
largeRange2_load_diff == very_high) -> largeRange2_OUT =
SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri ==
very_high & largeRange2_load_diff == very_low) -> largeRange2_OUT =
PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri ==
very_high & largeRange2_load_diff == low) -> largeRange2_OUT =
VERY_PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri ==
very_high & largeRange2_load_diff == perfect) -> largeRange2_OUT =
VERY_PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri ==
very_high & largeRange2_load_diff == high) -> largeRange2_OUT =
VERY_PROMISING;
if (RoomOCCUPIED == NO & largeRange2 == On & largeRange2_pri ==
very_high & largeRange2_load_diff == very_high) -> largeRange2_OUT =
PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == very_low &
toastOvn_load_diff == very_low) -> toastOvn_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == very_low &
toastOvn_load_diff == low) -> toastOvn_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == very_low &
toastOvn_load_diff == perfect) -> toastOvn_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == very_low &
toastOvn_load_diff == high) -> toastOvn_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == very_low &
toastOvn_load_diff == very_high) -> toastOvn_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == low &
toastOvn_load_diff == very_low) -> toastOvn_OUT = NOT_PROMISING;

```

if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == low & toastOvn_load_diff == low) -> toastOvn_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == low & toastOvn_load_diff == perfect) -> toastOvn_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == low & toastOvn_load_diff == high) -> toastOvn_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == low & toastOvn_load_diff == very_high) -> toastOvn_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == average & toastOvn_load_diff == very_low) -> toastOvn_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == average & toastOvn_load_diff == low) -> toastOvn_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == average & toastOvn_load_diff == perfect) -> toastOvn_OUT = PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == average & toastOvn_load_diff == high) -> toastOvn_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == average & toastOvn_load_diff == very_high) -> toastOvn_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == above_average & toastOvn_load_diff == very_low) -> toastOvn_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == above_average & toastOvn_load_diff == low) -> toastOvn_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == above_average & toastOvn_load_diff == perfect) -> toastOvn_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == above_average & toastOvn_load_diff == high) -> toastOvn_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == above_average & toastOvn_load_diff == very_high) -> toastOvn_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == high & toastOvn_load_diff == very_low) -> toastOvn_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == high & toastOvn_load_diff == low) -> toastOvn_OUT = PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == high & toastOvn_load_diff == perfect) -> toastOvn_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == high & toastOvn_load_diff == high) -> toastOvn_OUT = PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == high & toastOvn_load_diff == very_high) -> toastOvn_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == very_high & toastOvn_load_diff == very_low) -> toastOvn_OUT = PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == very_high & toastOvn_load_diff == low) -> toastOvn_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == very_high & toastOvn_load_diff == perfect) -> toastOvn_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == very_high & toastOvn_load_diff == high) -> toastOvn_OUT = VERY_PROMISING;

if (RoomOCCUPIED == NO & toastOvn == On & toastOvn_pri == very_high & toastOvn_load_diff == very_high) -> toastOvn_OUT = PROMISING;
if (RoomOCCUPIED == NO & microwave == On & microwave_pri == very_low & microwave_load_diff == very_low) -> microwave_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & microwave == On & microwave_pri == very_low & microwave_load_diff == low) -> microwave_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & microwave == On & microwave_pri == very_low & microwave_load_diff == perfect) -> microwave_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & microwave == On & microwave_pri == very_low & microwave_load_diff == high) -> microwave_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & microwave == On & microwave_pri == very_low & microwave_load_diff == very_high) -> microwave_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & microwave == On & microwave_pri == low & microwave_load_diff == very_low) -> microwave_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & microwave == On & microwave_pri == low & microwave_load_diff == low) -> microwave_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & microwave == On & microwave_pri == low & microwave_load_diff == perfect) -> microwave_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & microwave == On & microwave_pri == low & microwave_load_diff == high) -> microwave_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & microwave == On & microwave_pri == low & microwave_load_diff == very_high) -> microwave_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & microwave == On & microwave_pri == average & microwave_load_diff == very_low) -> microwave_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & microwave == On & microwave_pri == average & microwave_load_diff == low) -> microwave_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & microwave == On & microwave_pri == average & microwave_load_diff == perfect) -> microwave_OUT = PROMISING;
if (RoomOCCUPIED == NO & microwave == On & microwave_pri == average & microwave_load_diff == high) -> microwave_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & microwave == On & microwave_pri == average & microwave_load_diff == very_high) -> microwave_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & microwave == On & microwave_pri == above_average & microwave_load_diff == very_low) -> microwave_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & microwave == On & microwave_pri == above_average & microwave_load_diff == low) -> microwave_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & microwave == On & microwave_pri == above_average & microwave_load_diff == perfect) -> microwave_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & microwave == On & microwave_pri == above_average & microwave_load_diff == high) -> microwave_OUT = SLIGHTLY_PROMISING;

if (RoomOCCUPIED == NO & microwave == On & microwave_pri == above_average & microwave_load_diff == very_high) -> microwave_OUT = NEUTRAL;

if (RoomOCCUPIED == NO & microwave == On & microwave_pri == high & microwave_load_diff == very_low) -> microwave_OUT = SLIGHTLY_PROMISING;

if (RoomOCCUPIED == NO & microwave == On & microwave_pri == high & microwave_load_diff == low) -> microwave_OUT = PROMISING;

if (RoomOCCUPIED == NO & microwave == On & microwave_pri == high & microwave_load_diff == perfect) -> microwave_OUT = VERY_PROMISING;

if (RoomOCCUPIED == NO & microwave == On & microwave_pri == high & microwave_load_diff == high) -> microwave_OUT = PROMISING;

if (RoomOCCUPIED == NO & microwave == On & microwave_pri == high & microwave_load_diff == very_high) -> microwave_OUT = SLIGHTLY_PROMISING;

if (RoomOCCUPIED == NO & microwave == On & microwave_pri == very_high & microwave_load_diff == very_low) -> microwave_OUT = PROMISING;

if (RoomOCCUPIED == NO & microwave == On & microwave_pri == very_high & microwave_load_diff == low) -> microwave_OUT = VERY_PROMISING;

if (RoomOCCUPIED == NO & microwave == On & microwave_pri == very_high & microwave_load_diff == perfect) -> microwave_OUT = VERY_PROMISING;

if (RoomOCCUPIED == NO & microwave == On & microwave_pri == very_high & microwave_load_diff == high) -> microwave_OUT = VERY_PROMISING;

if (RoomOCCUPIED == NO & microwave == On & microwave_pri == very_high & microwave_load_diff == very_high) -> microwave_OUT = PROMISING;

if (RoomOCCUPIED == NO & refrig == On & refrig_pri == very_low & refrig_load_diff == very_low) -> refrig_OUT = NOT_PROMISING;

if (RoomOCCUPIED == NO & refrig == On & refrig_pri == very_low & refrig_load_diff == low) -> refrig_OUT = NOT_PROMISING;

if (RoomOCCUPIED == NO & refrig == On & refrig_pri == very_low & refrig_load_diff == perfect) -> refrig_OUT = NEUTRAL;

if (RoomOCCUPIED == NO & refrig == On & refrig_pri == very_low & refrig_load_diff == high) -> refrig_OUT = NOT_PROMISING;

if (RoomOCCUPIED == NO & refrig == On & refrig_pri == very_low & refrig_load_diff == very_high) -> refrig_OUT = NOT_PROMISING;

if (RoomOCCUPIED == NO & refrig == On & refrig_pri == low & refrig_load_diff == very_low) -> refrig_OUT = NOT_PROMISING;

if (RoomOCCUPIED == NO & refrig == On & refrig_pri == low & refrig_load_diff == low) -> refrig_OUT = LESS_PROMISING;

if (RoomOCCUPIED == NO & refrig == On & refrig_pri == low & refrig_load_diff == perfect) -> refrig_OUT = SLIGHTLY_PROMISING;

if (RoomOCCUPIED == NO & refrig == On & refrig_pri == low & refrig_load_diff == high) -> refrig_OUT = LESS_PROMISING;

if (RoomOCCUPIED == NO & refrig == On & refrig_pri == low & refrig_load_diff == very_high) -> refrig_OUT = NOT_PROMISING;

if (RoomOCCUPIED == NO & refig == On & refig_pri == average & refig_load_diff == very_low) -> refig_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & refig == On & refig_pri == average & refig_load_diff == low) -> refig_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & refig == On & refig_pri == average & refig_load_diff == perfect) -> refig_OUT = PROMISING;
if (RoomOCCUPIED == NO & refig == On & refig_pri == average & refig_load_diff == high) -> refig_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & refig == On & refig_pri == average & refig_load_diff == very_high) -> refig_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & refig == On & refig_pri == above_average & refig_load_diff == very_low) -> refig_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & refig == On & refig_pri == above_average & refig_load_diff == low) -> refig_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & refig == On & refig_pri == above_average & refig_load_diff == perfect) -> refig_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & refig == On & refig_pri == above_average & refig_load_diff == high) -> refig_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & refig == On & refig_pri == above_average & refig_load_diff == very_high) -> refig_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & refig == On & refig_pri == high & refig_load_diff == very_low) -> refig_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & refig == On & refig_pri == high & refig_load_diff == low) -> refig_OUT = PROMISING;
if (RoomOCCUPIED == NO & refig == On & refig_pri == high & refig_load_diff == perfect) -> refig_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & refig == On & refig_pri == high & refig_load_diff == high) -> refig_OUT = PROMISING;
if (RoomOCCUPIED == NO & refig == On & refig_pri == high & refig_load_diff == very_high) -> refig_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & refig == On & refig_pri == very_high & refig_load_diff == very_low) -> refig_OUT = PROMISING;
if (RoomOCCUPIED == NO & refig == On & refig_pri == very_high & refig_load_diff == low) -> refig_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & refig == On & refig_pri == very_high & refig_load_diff == perfect) -> refig_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & refig == On & refig_pri == very_high & refig_load_diff == high) -> refig_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & refig == On & refig_pri == very_high & refig_load_diff == very_high) -> refig_OUT = PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == very_low & oven_load_diff == very_low) -> oven_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == very_low & oven_load_diff == low) -> oven_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == very_low & oven_load_diff == perfect) -> oven_OUT = NEUTRAL;

if (RoomOCCUPIED == NO & oven == On & oven_pri == very_low & oven_load_diff == high) -> oven_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == very_low & oven_load_diff == very_high) -> oven_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == low & oven_load_diff == very_low) -> oven_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == low & oven_load_diff == low) -> oven_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == low & oven_load_diff == perfect) -> oven_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == low & oven_load_diff == high) -> oven_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == low & oven_load_diff == very_high) -> oven_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == average & oven_load_diff == very_low) -> oven_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == average & oven_load_diff == low) -> oven_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & oven == On & oven_pri == average & oven_load_diff == perfect) -> oven_OUT = PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == average & oven_load_diff == high) -> oven_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & oven == On & oven_pri == average & oven_load_diff == very_high) -> oven_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == above_average & oven_load_diff == very_low) -> oven_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & oven == On & oven_pri == above_average & oven_load_diff == low) -> oven_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == above_average & oven_load_diff == perfect) -> oven_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == above_average & oven_load_diff == high) -> oven_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == above_average & oven_load_diff == very_high) -> oven_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & oven == On & oven_pri == high & oven_load_diff == very_low) -> oven_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == high & oven_load_diff == low) -> oven_OUT = PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == high & oven_load_diff == perfect) -> oven_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == high & oven_load_diff == high) -> oven_OUT = PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == high & oven_load_diff == very_high) -> oven_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == very_high & oven_load_diff == very_low) -> oven_OUT = PROMISING;

if (RoomOCCUPIED == NO & oven == On & oven_pri == very_high & oven_load_diff == low) -> oven_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == very_high & oven_load_diff == perfect) -> oven_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == very_high & oven_load_diff == high) -> oven_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & oven == On & oven_pri == very_high & oven_load_diff == very_high) -> oven_OUT = PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == very_low & blowdrier1_load_diff == very_low) -> blowdrier1_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == very_low & blowdrier1_load_diff == low) -> blowdrier1_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == very_low & blowdrier1_load_diff == perfect) -> blowdrier1_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == very_low & blowdrier1_load_diff == high) -> blowdrier1_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == very_low & blowdrier1_load_diff == very_high) -> blowdrier1_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == low & blowdrier1_load_diff == very_low) -> blowdrier1_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == low & blowdrier1_load_diff == low) -> blowdrier1_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == low & blowdrier1_load_diff == perfect) -> blowdrier1_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == low & blowdrier1_load_diff == high) -> blowdrier1_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == low & blowdrier1_load_diff == very_high) -> blowdrier1_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == average & blowdrier1_load_diff == very_low) -> blowdrier1_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == average & blowdrier1_load_diff == low) -> blowdrier1_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == average & blowdrier1_load_diff == perfect) -> blowdrier1_OUT = PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == average & blowdrier1_load_diff == high) -> blowdrier1_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == average & blowdrier1_load_diff == very_high) -> blowdrier1_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == above_average & blowdrier1_load_diff == very_low) -> blowdrier1_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == above_average & blowdrier1_load_diff == low) -> blowdrier1_OUT = SLIGHTLY_PROMISING;

if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == above_average & blowdrier1_load_diff == perfect) -> blowdrier1_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == above_average & blowdrier1_load_diff == high) -> blowdrier1_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == above_average & blowdrier1_load_diff == very_high) -> blowdrier1_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == high & blowdrier1_load_diff == very_low) -> blowdrier1_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == high & blowdrier1_load_diff == low) -> blowdrier1_OUT = PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == high & blowdrier1_load_diff == perfect) -> blowdrier1_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == high & blowdrier1_load_diff == high) -> blowdrier1_OUT = PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == high & blowdrier1_load_diff == very_high) -> blowdrier1_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == very_high & blowdrier1_load_diff == very_low) -> blowdrier1_OUT = PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == very_high & blowdrier1_load_diff == low) -> blowdrier1_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == very_high & blowdrier1_load_diff == perfect) -> blowdrier1_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == very_high & blowdrier1_load_diff == high) -> blowdrier1_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & blowdrier1 == On & blowdrier1_pri == very_high & blowdrier1_load_diff == very_high) -> blowdrier1_OUT = PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == very_low & rangeFan_load_diff == very_low) -> rangeFan_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == very_low & rangeFan_load_diff == low) -> rangeFan_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == very_low & rangeFan_load_diff == perfect) -> rangeFan_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == very_low & rangeFan_load_diff == high) -> rangeFan_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == very_low & rangeFan_load_diff == very_high) -> rangeFan_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == low & rangeFan_load_diff == very_low) -> rangeFan_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == low & rangeFan_load_diff == low) -> rangeFan_OUT = LESS_PROMISING;

if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == low & rangeFan_load_diff == perfect) -> rangeFan_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == low & rangeFan_load_diff == high) -> rangeFan_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == low & rangeFan_load_diff == very_high) -> rangeFan_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == average & rangeFan_load_diff == very_low) -> rangeFan_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == average & rangeFan_load_diff == low) -> rangeFan_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == average & rangeFan_load_diff == perfect) -> rangeFan_OUT = PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == average & rangeFan_load_diff == high) -> rangeFan_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == average & rangeFan_load_diff == very_high) -> rangeFan_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == above_average & rangeFan_load_diff == very_low) -> rangeFan_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == above_average & rangeFan_load_diff == low) -> rangeFan_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == above_average & rangeFan_load_diff == perfect) -> rangeFan_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == above_average & rangeFan_load_diff == high) -> rangeFan_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == above_average & rangeFan_load_diff == very_high) -> rangeFan_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == high & rangeFan_load_diff == very_low) -> rangeFan_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == high & rangeFan_load_diff == low) -> rangeFan_OUT = PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == high & rangeFan_load_diff == perfect) -> rangeFan_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == high & rangeFan_load_diff == high) -> rangeFan_OUT = PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == high & rangeFan_load_diff == very_high) -> rangeFan_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == very_high & rangeFan_load_diff == very_low) -> rangeFan_OUT = PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == very_high & rangeFan_load_diff == low) -> rangeFan_OUT = VERY_PROMISING;

if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == very_high & rangeFan_load_diff == perfect) -> rangeFan_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == very_high & rangeFan_load_diff == high) -> rangeFan_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & rangeFan == On & rangeFan_pri == very_high & rangeFan_load_diff == very_high) -> rangeFan_OUT = PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == very_low & disposal_load_diff == very_low) -> disposal_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == very_low & disposal_load_diff == low) -> disposal_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == very_low & disposal_load_diff == perfect) -> disposal_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == very_low & disposal_load_diff == high) -> disposal_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == very_low & disposal_load_diff == very_high) -> disposal_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == low & disposal_load_diff == very_low) -> disposal_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == low & disposal_load_diff == low) -> disposal_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == low & disposal_load_diff == perfect) -> disposal_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == low & disposal_load_diff == high) -> disposal_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == low & disposal_load_diff == very_high) -> disposal_OUT = NOT_PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == average & disposal_load_diff == very_low) -> disposal_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == average & disposal_load_diff == low) -> disposal_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == average & disposal_load_diff == perfect) -> disposal_OUT = PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == average & disposal_load_diff == high) -> disposal_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == average & disposal_load_diff == very_high) -> disposal_OUT = LESS_PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == above_average & disposal_load_diff == very_low) -> disposal_OUT = NEUTRAL;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == above_average & disposal_load_diff == low) -> disposal_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == above_average & disposal_load_diff == perfect) -> disposal_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == above_average & disposal_load_diff == high) -> disposal_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == above_average & disposal_load_diff == very_high) -> disposal_OUT = NEUTRAL;

```

if (RoomOCCUPIED == NO & disposal == On & disposal_pri == high &
disposal_load_diff == very_low) -> disposal_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == high &
disposal_load_diff == low) -> disposal_OUT = PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == high &
disposal_load_diff == perfect) -> disposal_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == high &
disposal_load_diff == high) -> disposal_OUT = PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == high &
disposal_load_diff == very_high) -> disposal_OUT = SLIGHTLY_PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == very_high &
disposal_load_diff == very_low) -> disposal_OUT = PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == very_high &
disposal_load_diff == low) -> disposal_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == very_high &
disposal_load_diff == perfect) -> disposal_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == very_high &
disposal_load_diff == high) -> disposal_OUT = VERY_PROMISING;
if (RoomOCCUPIED == NO & disposal == On & disposal_pri == very_high &
disposal_load_diff == very_high) -> disposal_OUT = PROMISING;
}

```

```

system(TRoomOCCUPIED RoomOCCUPIED, TAppliance dishwasher,
TAppliance_pri dishwasher_pri, TAppliance_load_diff dishwasher_load_diff,
TAppliance washer, TAppliance_pri washer_pri, TAppliance_load_diff
washer_load_diff,
TAppliance dryer, TAppliance_pri dryer_pri, TAppliance_load_diff
dryer_load_diff,
TAppliance kitchLight, TAppliance_pri kitchLight_pri, TAppliance_load_diff
kitchLight_load_diff,
TAppliance kitchFan, TAppliance_pri kitchFan_pri, TAppliance_load_diff
kitchFan_load_diff,
TAppliance smallRange1, TAppliance_pri smallRange1_pri,
TAppliance_load_diff smallRange1_load_diff,
TAppliance smallRange2, TAppliance_pri smallRange2_pri,
TAppliance_load_diff smallRange2_load_diff,
TAppliance largeRange1, TAppliance_pri largeRange1_pri,
TAppliance_load_diff largeRange1_load_diff,
TAppliance largeRange2, TAppliance_pri largeRange2_pri,
TAppliance_load_diff largeRange2_load_diff,
TAppliance toastOvn, TAppliance_pri toastOvn_pri, TAppliance_load_diff
toastOvn_load_diff,
TAppliance microwave, TAppliance_pri microwave_pri, TAppliance_load_diff
microwave_load_diff,
TAppliance refrig, TAppliance_pri refrig_pri, TAppliance_load_diff
refrig_load_diff,

```

```

TAppliance oven, TAppliance_pri oven_pri, TAppliance_load_diff
oven_load_diff,
TAppliance blowdrier1, TAppliance_pri blowdrier1_pri, TAppliance_load_diff
blowdrier1_load_diff,
TAppliance rangeFan, TAppliance_pri rangeFan_pri, TAppliance_load_diff
rangeFan_load_diff,
TAppliance disposal, TAppliance_pri disposal_pri, TAppliance_load_diff
disposal_load_diff
:OUT_TAPPLIANCE dishwasher_OUT ,
OUT_TAPPLIANCE washer_OUT ,
OUT_TAPPLIANCE dryer_OUT ,
OUT_TAPPLIANCE kitchLight_OUT ,
OUT_TAPPLIANCE kitchFan_OUT ,
OUT_TAPPLIANCE smallRange1_OUT ,
OUT_TAPPLIANCE smallRange2_OUT ,
OUT_TAPPLIANCE largeRange1_OUT ,
OUT_TAPPLIANCE largeRange2_OUT ,
OUT_TAPPLIANCE toastOvn_OUT ,
OUT_TAPPLIANCE microwave_OUT ,
OUT_TAPPLIANCE refrig_OUT ,
OUT_TAPPLIANCE oven_OUT ,
OUT_TAPPLIANCE blowdrier1_OUT ,
OUT_TAPPLIANCE rangeFan_OUT ,
OUT_TAPPLIANCE disposal_OUT
){
global(RoomOCCUPIED, dishwasher, dishwasher_pri, dishwasher_load_diff,
washer, washer_pri, washer_load_diff,
dryer, dryer_pri, dryer_load_diff,
kitchLight, kitchLight_pri, kitchLight_load_diff,
kitchFan, kitchFan_pri, kitchFan_load_diff,
smallRange1, smallRange1_pri, smallRange1_load_diff,
smallRange2, smallRange2_pri, smallRange2_load_diff,
largeRange1, largeRange1_pri, largeRange1_load_diff,
largeRange2, largeRange2_pri, largeRange2_load_diff,
toastOvn, toastOvn_pri, toastOvn_load_diff,
microwave, microwave_pri, microwave_load_diff,
refrig, refrig_pri, refrig_load_diff,
oven, oven_pri, oven_load_diff,
blowdrier1, blowdrier1_pri, blowdrier1_load_diff,
rangeFan, rangeFan_pri, rangeFan_load_diff,
disposal, disposal_pri, disposal_load_diff
: dishwasher_OUT ,
washer_OUT ,
dryer_OUT ,
kitchLight_OUT ,
kitchFan_OUT ,

```

```
smallRange1_OUT ,  
smallRange2_OUT ,  
largeRange1_OUT ,  
largeRange2_OUT ,  
toastOvn_OUT ,  
microwave_OUT ,  
refrig_OUT ,  
oven_OUT ,  
blowdrier1_OUT ,  
rangeFan_OUT ,  
disposal_OUT  
) ;}
```

Appendix B

Hardware Specifications, Schematics and Layout of the Range Controller

Hardware Specifications of the Range Controller

Introduction:

The Range Controller board is the controller that is capable of interfacing to inputs from knobs (controlling each of the four top heating elements and the two oven heating elements) and the six solid state relays controlling power delivered to each of those six heating elements. This board is based on the MC9S12NE64 micro-controller. The boards provides input analog interface to the six knobs and output digital interface to the low-voltage control side of the six solid state relays. This board communicated with a PC through its Ethernet interface.

Hardware Description:

The Range Controller board is based on the MC9S12NE64 micro-controller operating at 25 MHz. The MC9S12NE64 has 8KB of internal SRAM, 64KB of internal flash EEPROM, Ethernet interface, 8 channels of A/D conversion, and some I/O ports. The boards provides interface from the analog signals provided by the six knobs to the A/D converters of the CPU, and provides interface from output lines to the low-voltage control side of the six solid state relays. This board provides an Ethernet interface to a PC, and contains circuitry to detect AC phase zero-crossing. The board receives stepped down AC power, which is rectified and regulated to provide power (3.3V and 5V). A BDM port is also available. The complete schematics and layout for the board are provided at the end of this Appendix.

The following describes the memory map (single chip). Software initialization to proper CPU internal registers should be made to conform to this mapping.

Address Range	Device
\$0000 - \$1FFF	Internal Registers Block
\$2000 - \$3FFE	Internal SRAM (8KB)
\$4000 - \$7FFF	Direct access to 3 rd page of internal flash EEPROM (16KB) - Page \$3E
\$8000 - \$BFFF	Page window (16KB) to access 1 st , 2 nd , 3 rd , or 4 th pages of internal flash EEPROM (16KB each) through the PPAGE register – Pages \$3C, \$3D, \$3E, or \$3F, respectively.
\$C000 - \$FFFF	Direct access to the 4 th page of internal flash EEPROM (16KB) – Page \$3F

The MC9S12NE64 CPU has several hardware resources. Some of the used resources are as follows.

(1) Port L(6:0) (output) is used as follows.

PL(4:0): EtherNet lights status.

PL(6:5): Not used.

- (2) Port G(7:0) (input/output) lines are used as follows.
PG(5:0): Output signals to relays.
PG(6): Input signal from phase detect circuitry (PH_DCT*), active low.
PG(7): Not used.
- (3) Port AD(7:0) (input) lines are used as follows:
PAD(5:0): Analog signals from the six knobs (0 to 3.3V), where 0V represents off state, and 3.3V represents fully on state (100%).
PAD(7:6): Not used.
- (4) IRQ* (PE1) input of the CPU is connected to the signal from phase detect circuitry (PH_DCT*), active low.

Initialization:

- (1) The CPU registers should be initialized properly. Port lines should be initialized as properly.
- (2) IRQ* may be enabled if desired.
- (3) Ethernet interface should be initialized.
- (4) PG(5:0) output lines should be all initialized to high (element relays are off).

Operation:

The CPU activities include the following.

- (1) Communicating with the A/D converters to get knob settings.
- (2) Processing knob settings, and providing necessary output(s) to the element relays.
- (3) Monitoring Ethernet interface, and taking proper action(s).

Communications between the MC9S12NE64 CPU and a PC:

The MC9S12NE64 CPU communicates with a PC through its Ethernet interface. The CPU receives commands from the PC and provides the requested status and actions. The following table provides a list of commands and their corresponding responses.

Command	Meaning	Response
^A~	Request number of loads	Number of elements (6)
^B~	Request total maximum KW	Total maximum KW rating of appliance
^Cn~	Request maximum KW of load n	Maximum KW rating of load n
^D~	Request actual total power level %	Actual total power level % of all loads
^En~	Request actual power level % of load n	Actual power level % of load n
^Fnp~	Set power level of load n to p%	Change actual power level of load n to p%, and acknowledge by sending actual power level % of load n