US 20100185336A1

(19) United States
(12) Patent Application Publication
Rovnyak et al.
(10) Pub. No.: US 2010/0185336 A1
(43) Pub. Date: Jul. 22, 2010

(54) INTEGRATED AND OPTIMIZED DISTRIBUTED GENERATION AND INTERCONNECT SYSTEM CONTROLLER

(76) Inventors: **Steven M. Rovnyak**, Indianapolis, IN (US); **Yaobin Chen**, Carmel, IN (US); **Yong Sheng**, Zionsville, IN (US); **Vikram Singh Rajput**, Indianapolis, IN (US); **Terry Pahis**, Anderson, IN (US); **Larry George**, Greenfield, IN (US); **Jeff Malkoff**, Noblesville, IN (US); **Xiaohui Hu**, Indianapolis, IN (US)

Correspondence Address:
BINGHAM MCHALE LLP
2700 MARKET TOWER, 10 WEST MARKET STREET
INDIANAPOLIS, IN 46204-4900 (US)

(21) Appl. No.: **12/355,972**

(22) Filed: **Jan. 19, 2009**

Publication Classification

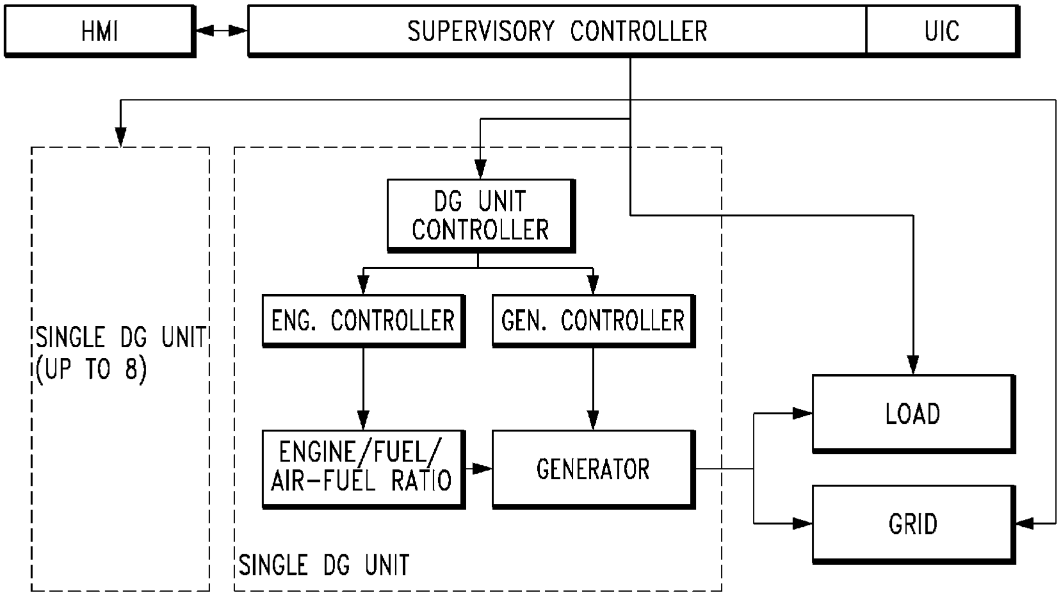
(51) **Int. Cl.**

G06F 1/26	(2006.01)
G06F 15/18	(2006.01)
G05B 15/02	(2006.01)
G01R 19/00	(2006.01)
G01R 31/08	(2006.01)
G06F 19/00	(2006.01)

(52) **U.S. Cl.** **700/287**; 706/12; 702/64; 702/59; 700/9

(57) **ABSTRACT**

Systems and methods for designing and controlling distributed generation resources are disclosed. Monolithic or distributed controllers control the power requests sent to each distributed generation resource to achieve optimal efficiency using rule-based and/or fuzzy logic-based control techniques. High-impedance fault (HIF) detection circuitry allows power export to the utility grid in normal operating circumstances, and when an islanding condition is detected, ceases that export within two seconds and prevents further export until the condition is cleared.



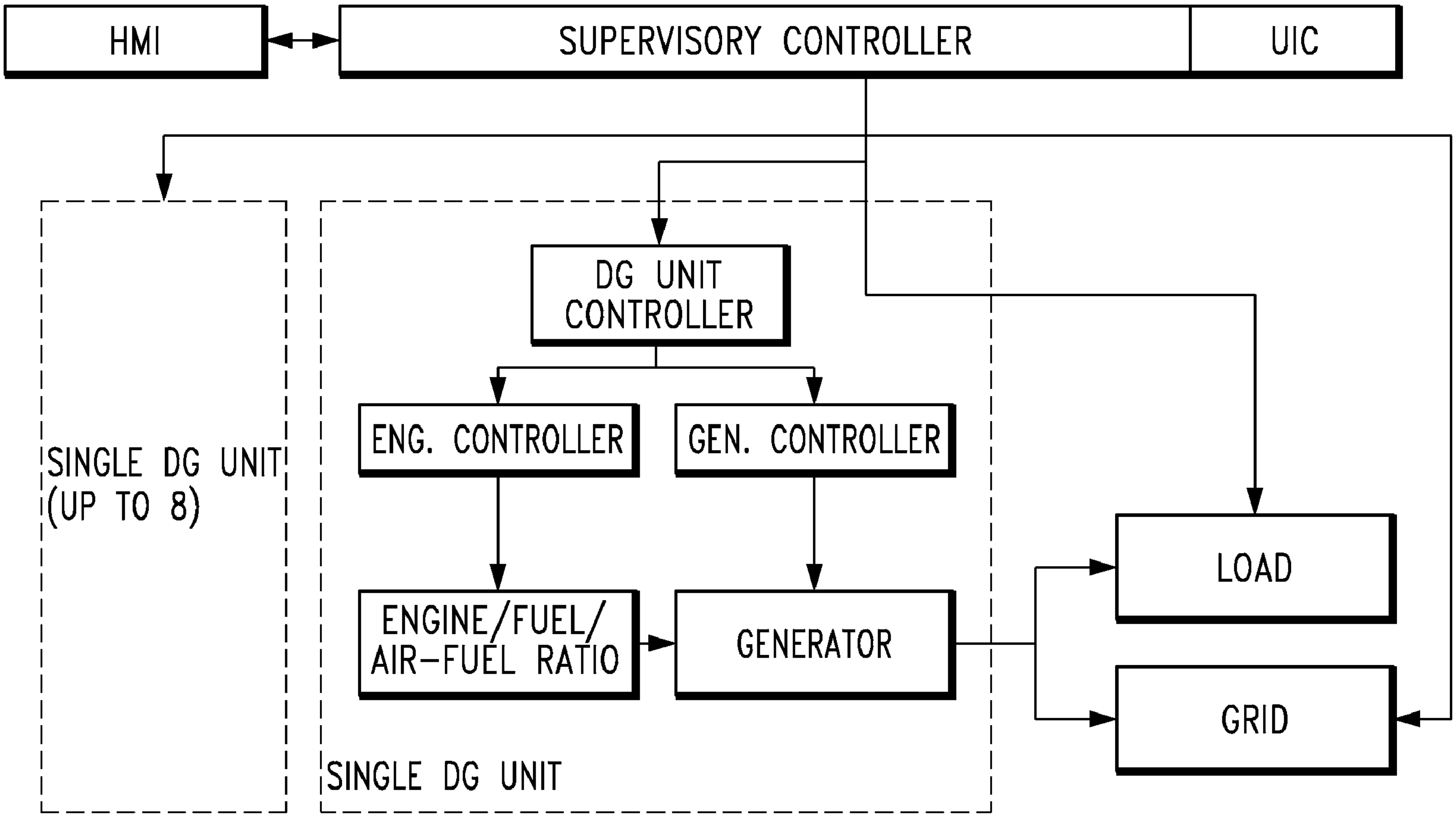

HMI
SUPERVISORY CONTROLLER
UIC
DG UNIT CONTROLLER
ENG. CONTROLLER
GEN. CONTROLLER
ENGINE/FUEL/ AIR-FUEL RATIO
GENERATOR
SINGLE DG UNIT
SINGLE DG UNIT (UP TO 8)
LOAD
GRID

Fig. 1

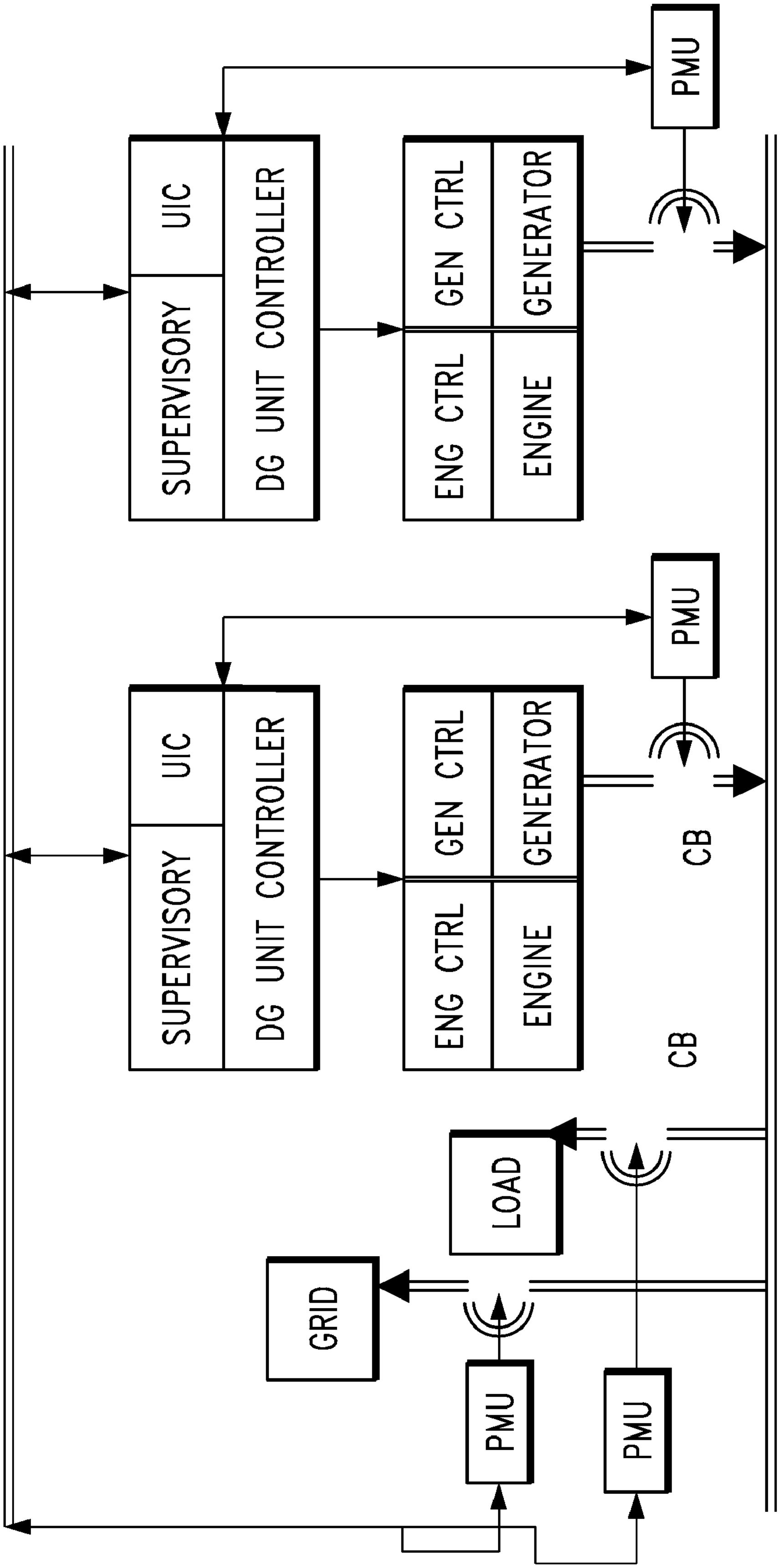

SUPERVISORY
UIC
DG UNIT CONTROLLER
ENG CTRL
GEN CTRL
ENGINE
GENERATOR
PMU
SUPERVISORY
UIC
DG UNIT CONTROLLER
ENG CTRL
GEN CTRL
ENGINE
GENERATOR
PMU
CB
CB
LOAD
GRID
PMU
PMU

Fig. 2

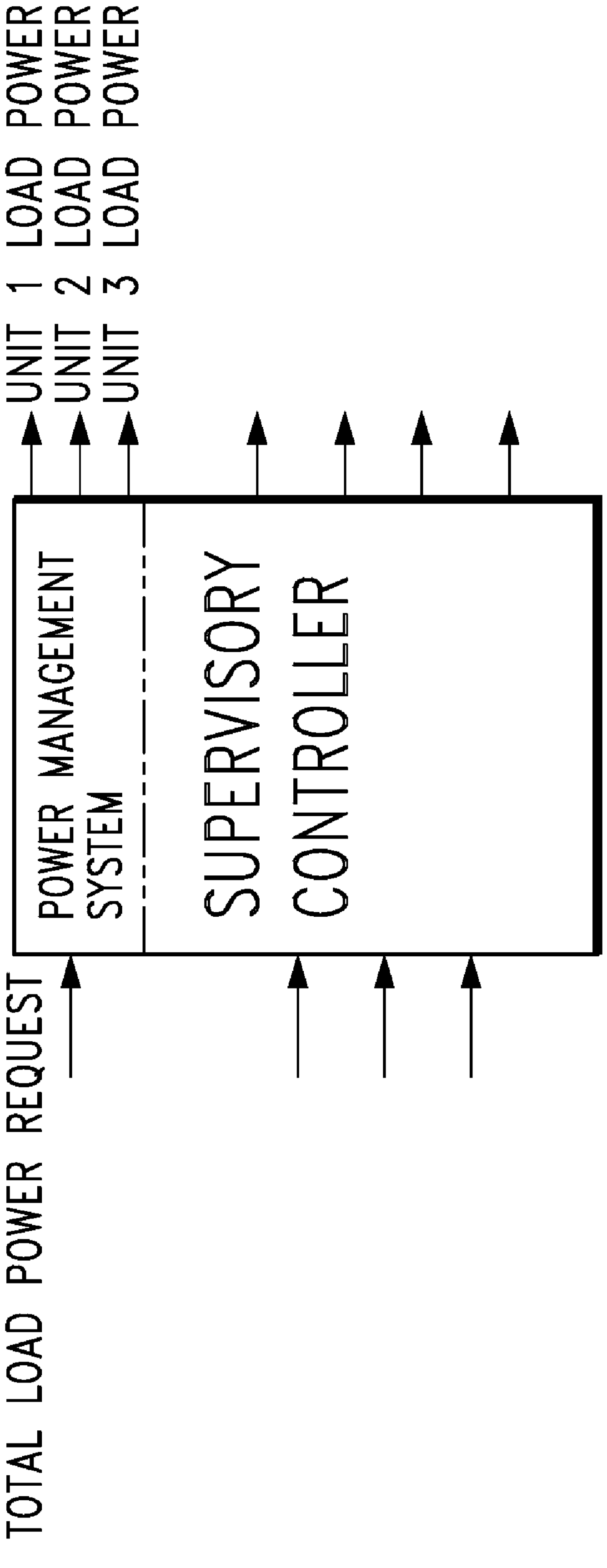

UNIT 1 LOAD POWER
UNIT 2 LOAD POWER
UNIT 3 LOAD POWER
POWER MANAGEMENT SYSTEM
SUPERVISORY CONTROLLER
TOTAL LOAD POWER REQUEST

Fig. 3

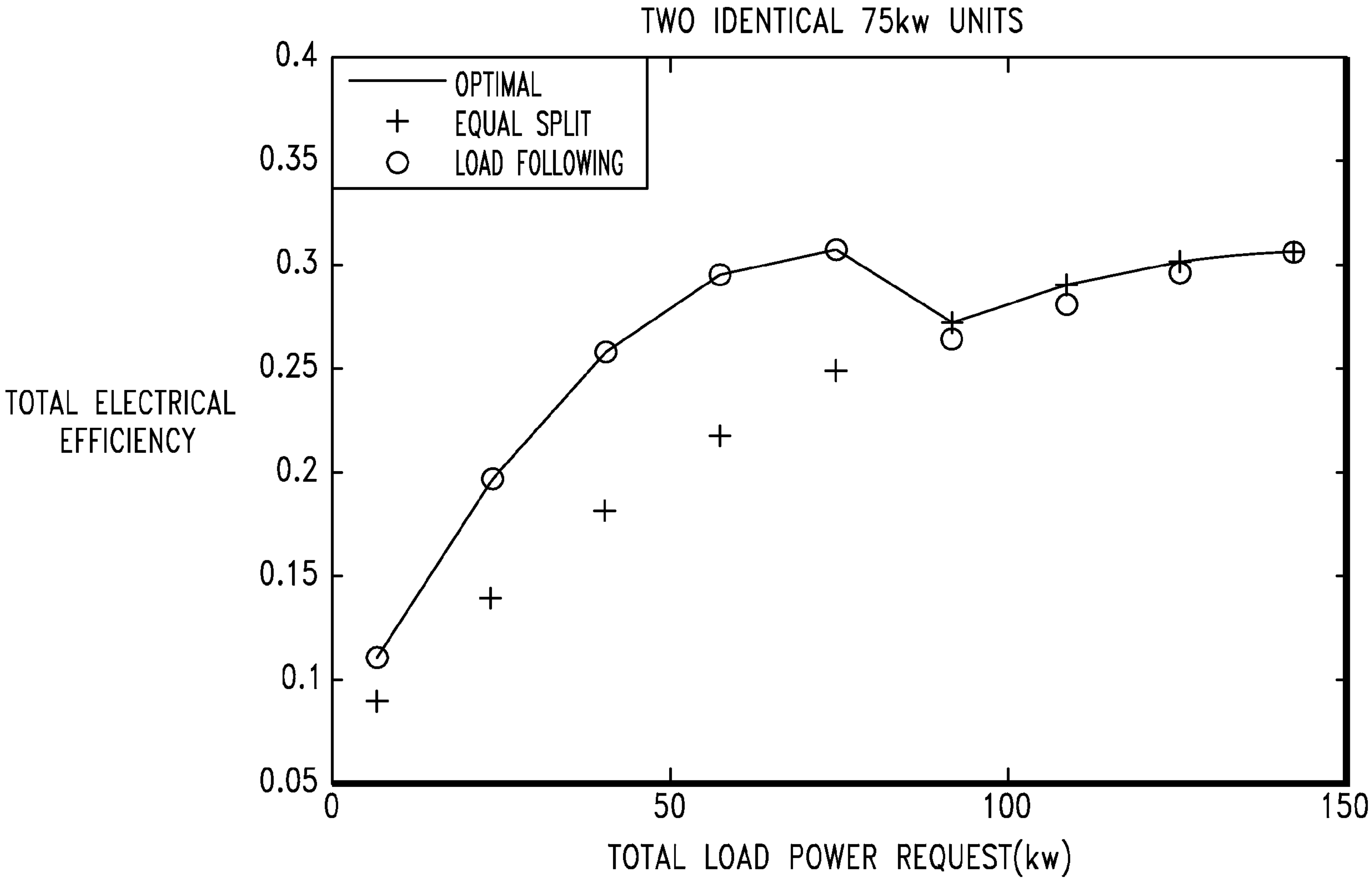

TWO IDENTICAL 75kw UNITS
OPTIMAL
+ EQUAL SPLIT
○ LOAD FOLLOWING
TOTAL ELECTRICAL EFFICIENCY
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
50
100
150
TOTAL LOAD POWER REQUEST(kw)

Fig. 4

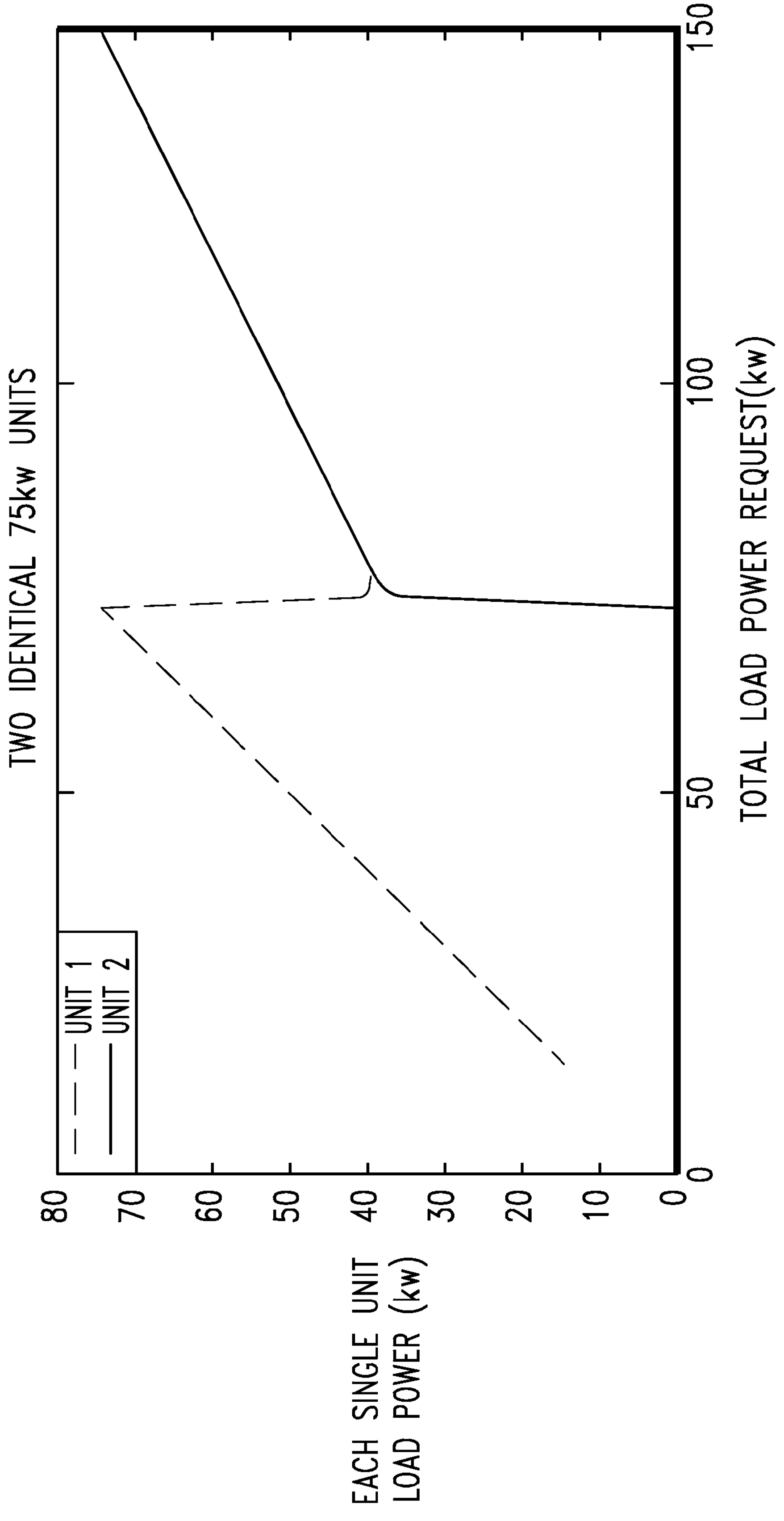

TWO IDENTICAL 75kw UNITS
UNIT 1
UNIT 2
EACH SINGLE UNIT
LOAD POWER (kw)
80
70
60
50
40
30
20
10
0
0
50
100
150
TOTAL LOAD POWER REQUEST(kw)

Fig. 5

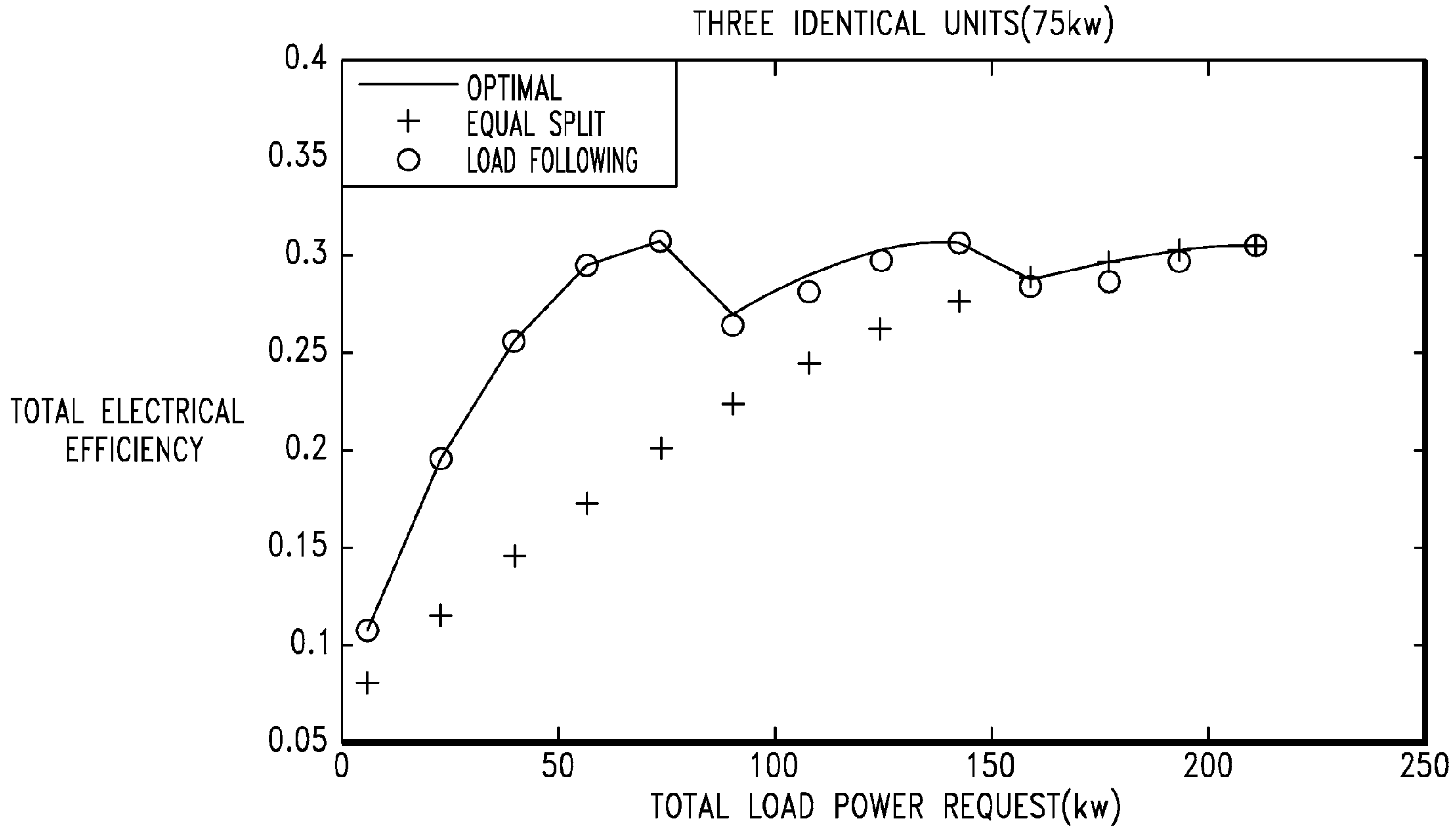

THREE IDENTICAL UNITS(75kw)
TOTAL ELECTRICAL EFFICIENCY
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
50
100
150
200
250
TOTAL LOAD POWER REQUEST(kw)
OPTIMAL
+ EQUAL SPLIT
○ LOAD FOLLOWING

Fig. 6

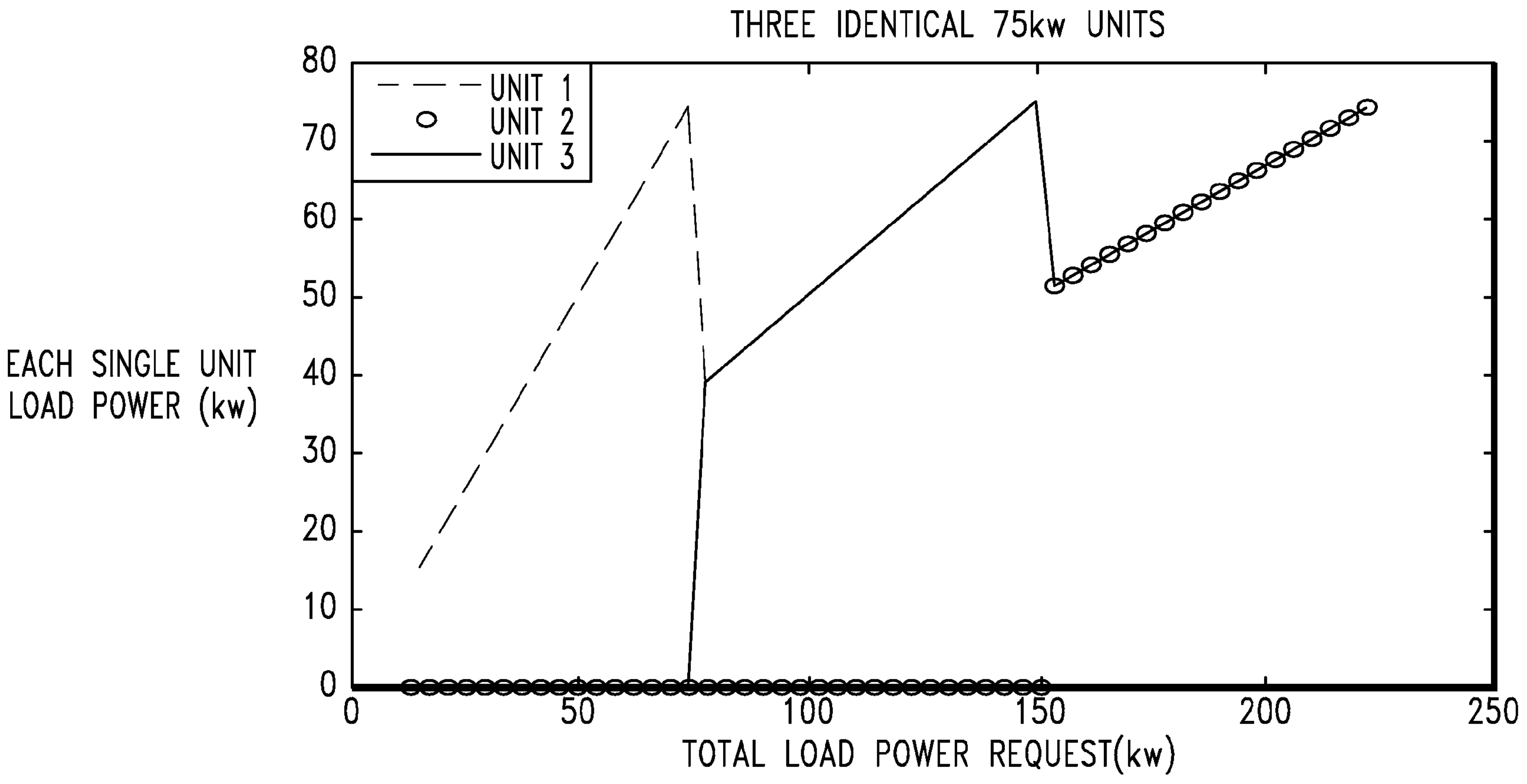

THREE IDENTICAL 75kw UNITS
UNIT 1
UNIT 2
UNIT 3
EACH SINGLE UNIT LOAD POWER (kw)
80
70
60
50
40
30
20
10
0
0
50
100
150
200
250
TOTAL LOAD POWER REQUEST(kw)

Fig. 7

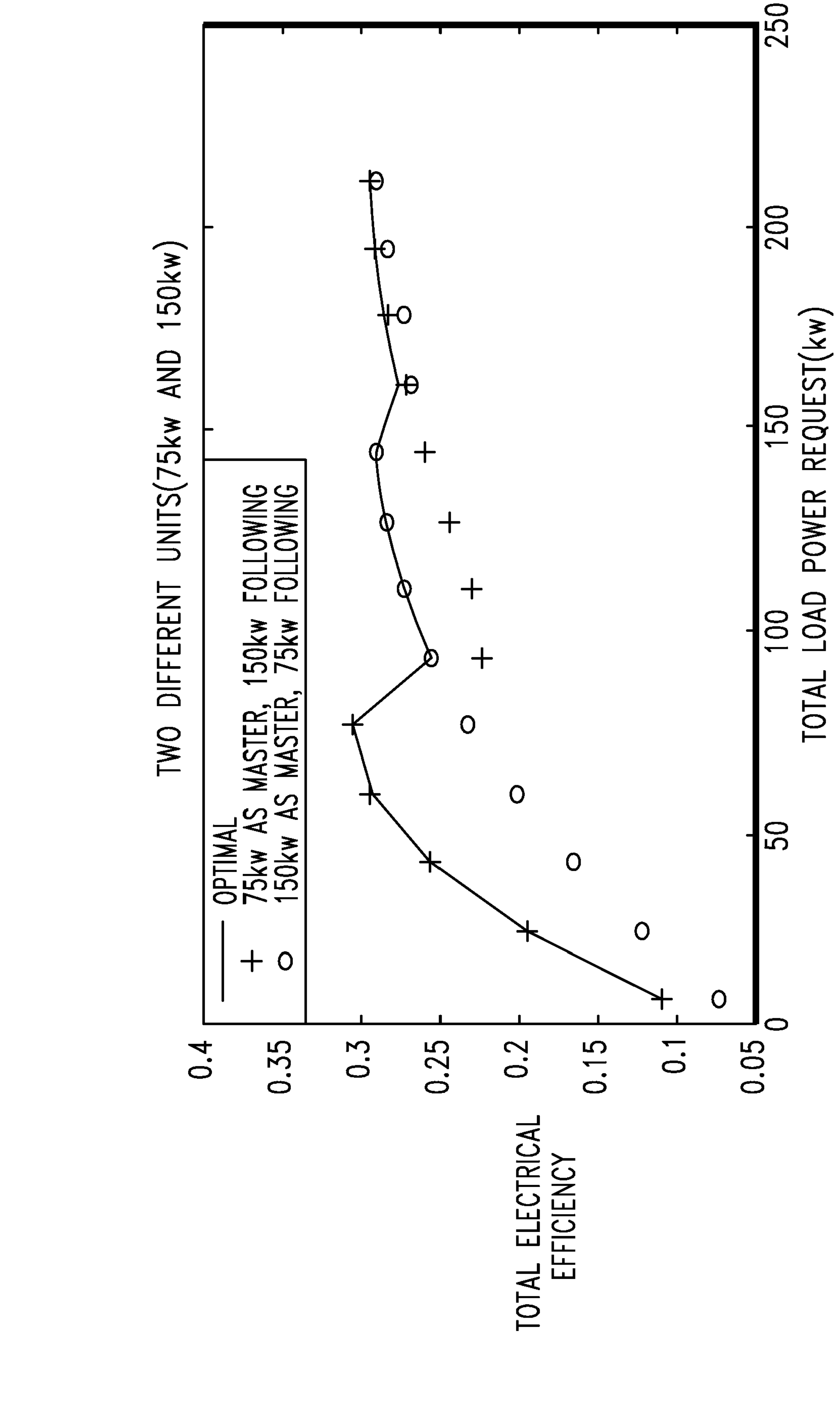

TWO DIFFERENT UNITS(75kw AND 150kw)
OPTIMAL
75kw AS MASTER, 150kw FOLLOWING
150kw AS MASTER, 75kw FOLLOWING
TOTAL ELECTRICAL EFFICIENCY
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
50
100
150
200
250
TOTAL LOAD POWER REQUEST(kw)

Fig. 8

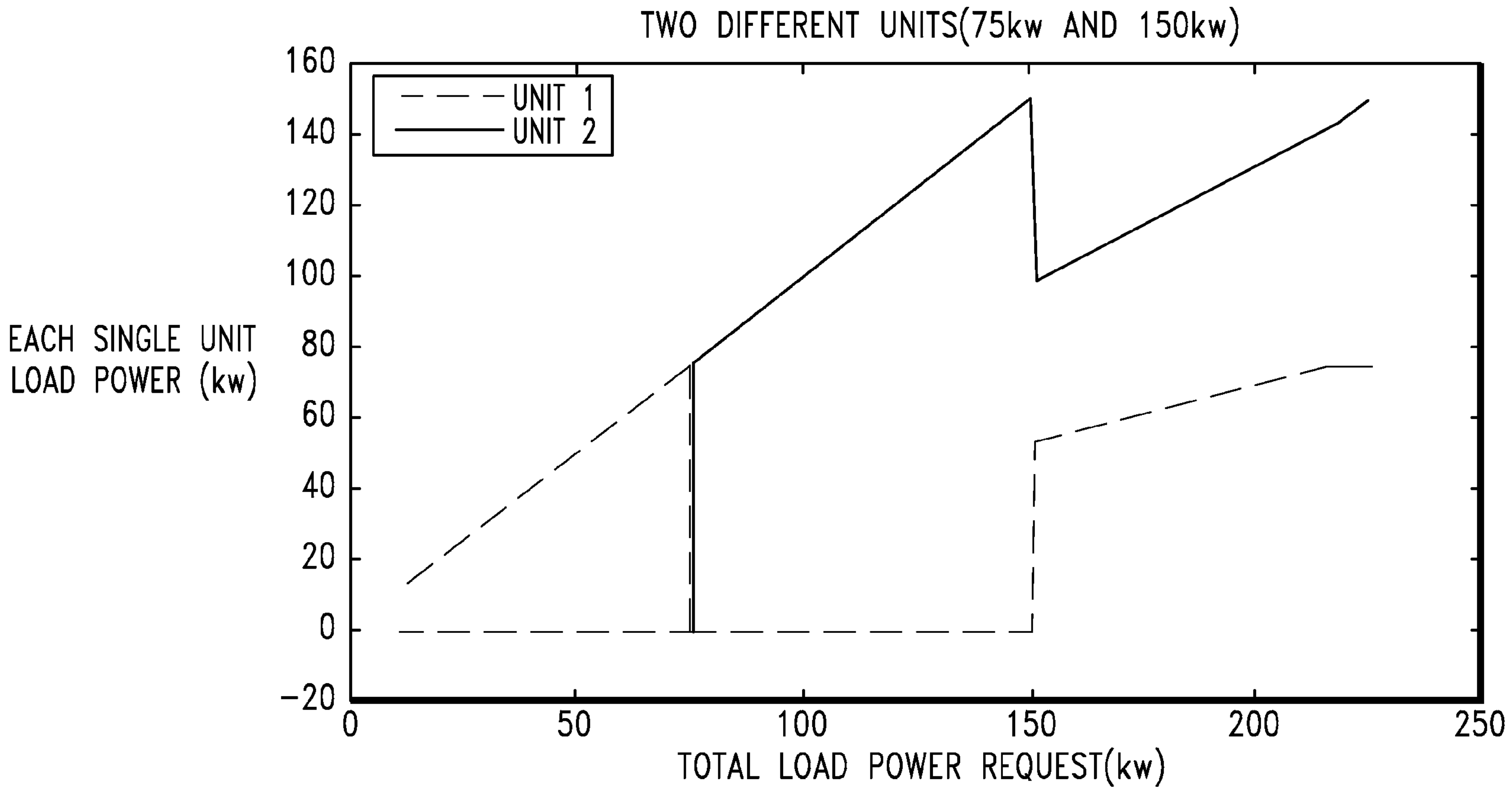

TWO DIFFERENT UNITS(75kw AND 150kw)
UNIT 1
UNIT 2
EACH SINGLE UNIT LOAD POWER (kw)
160
140
120
100
80
60
40
20
0
−20
0
50
100
150
200
250
TOTAL LOAD POWER REQUEST(kw)

Fig. 9

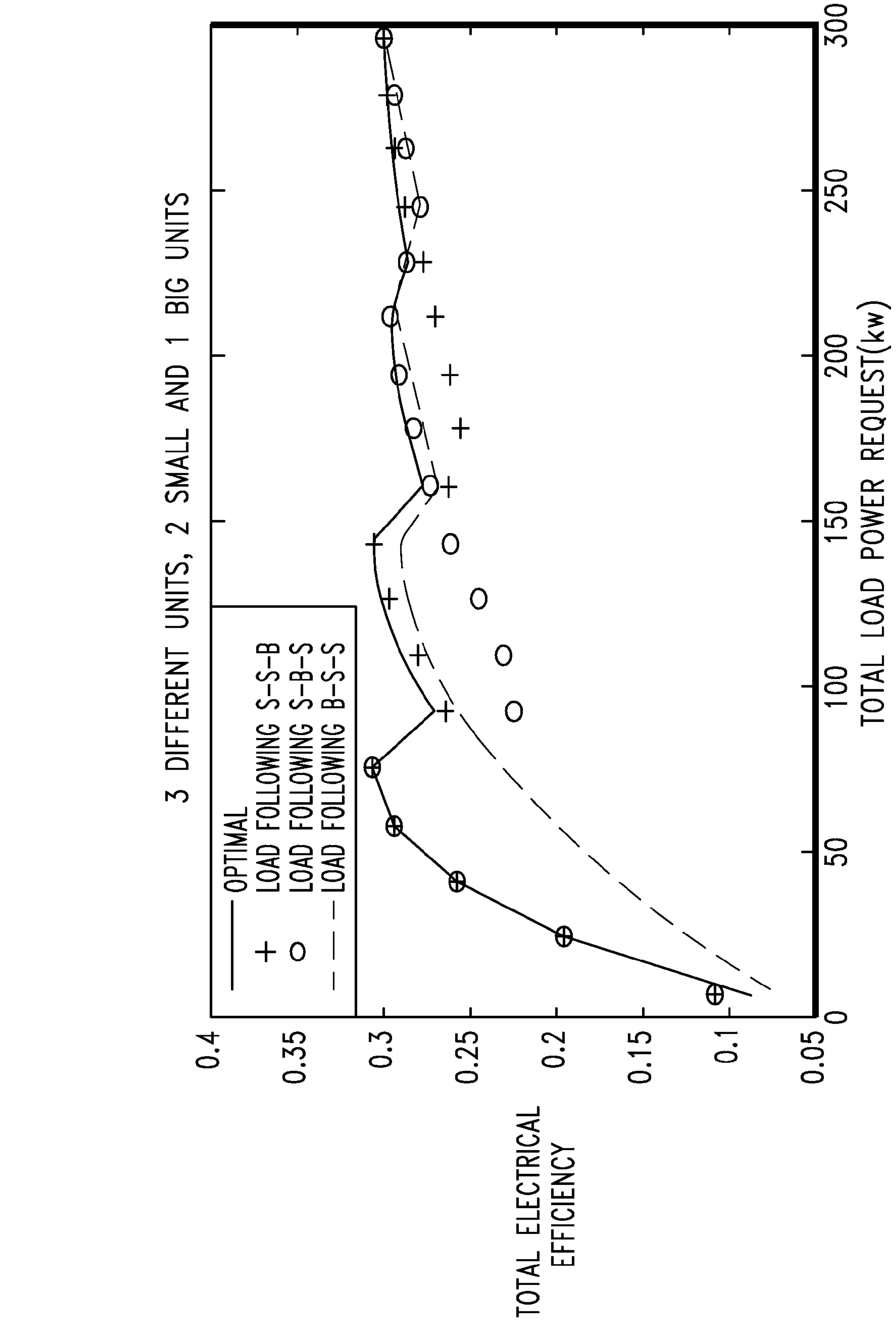

3 DIFFERENT UNITS, 2 SMALL AND 1 BIG UNITS
TOTAL ELECTRICAL EFFICIENCY
TOTAL LOAD POWER REQUEST(kw)
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
50
100
150
200
250
300
OPTIMAL
LOAD FOLLOWING S-S-B
LOAD FOLLOWING S-B-S
LOAD FOLLOWING B-S-S

Fig. 10

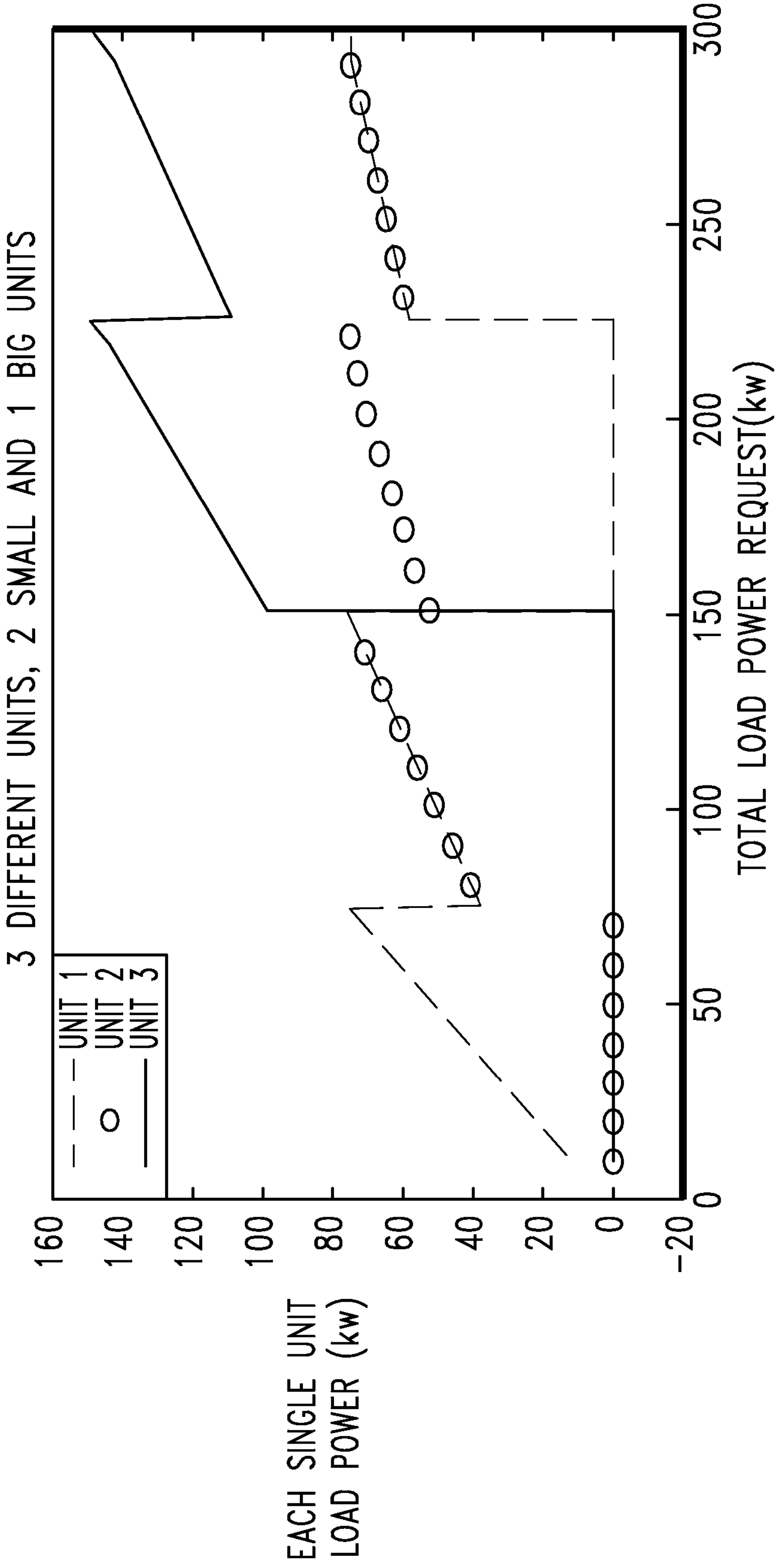

3 DIFFERENT UNITS, 2 SMALL AND 1 BIG UNITS
UNIT 1
UNIT 2
UNIT 3
EACH SINGLE UNIT LOAD POWER (kw)
160
140
120
100
80
60
40
20
0
-20
TOTAL LOAD POWER REQUEST(kw)
0
50
100
150
200
250
300

Fig. 11

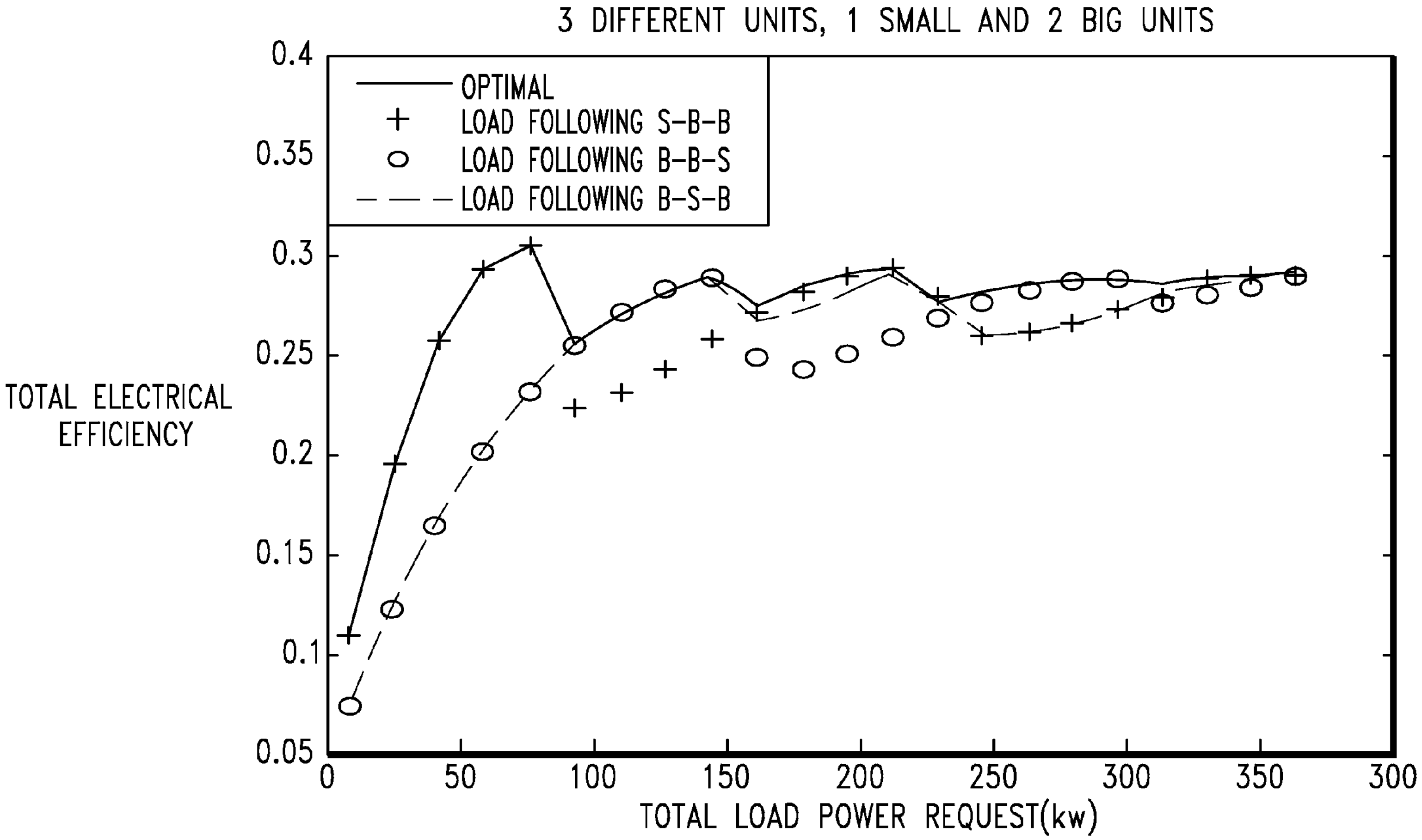

3 DIFFERENT UNITS, 1 SMALL AND 2 BIG UNITS
TOTAL ELECTRICAL EFFICIENCY
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
50
100
150
200
250
300
350
300
TOTAL LOAD POWER REQUEST(kw)
OPTIMAL
LOAD FOLLOWING S-B-B
LOAD FOLLOWING B-B-S
LOAD FOLLOWING B-S-B

Fig. 12

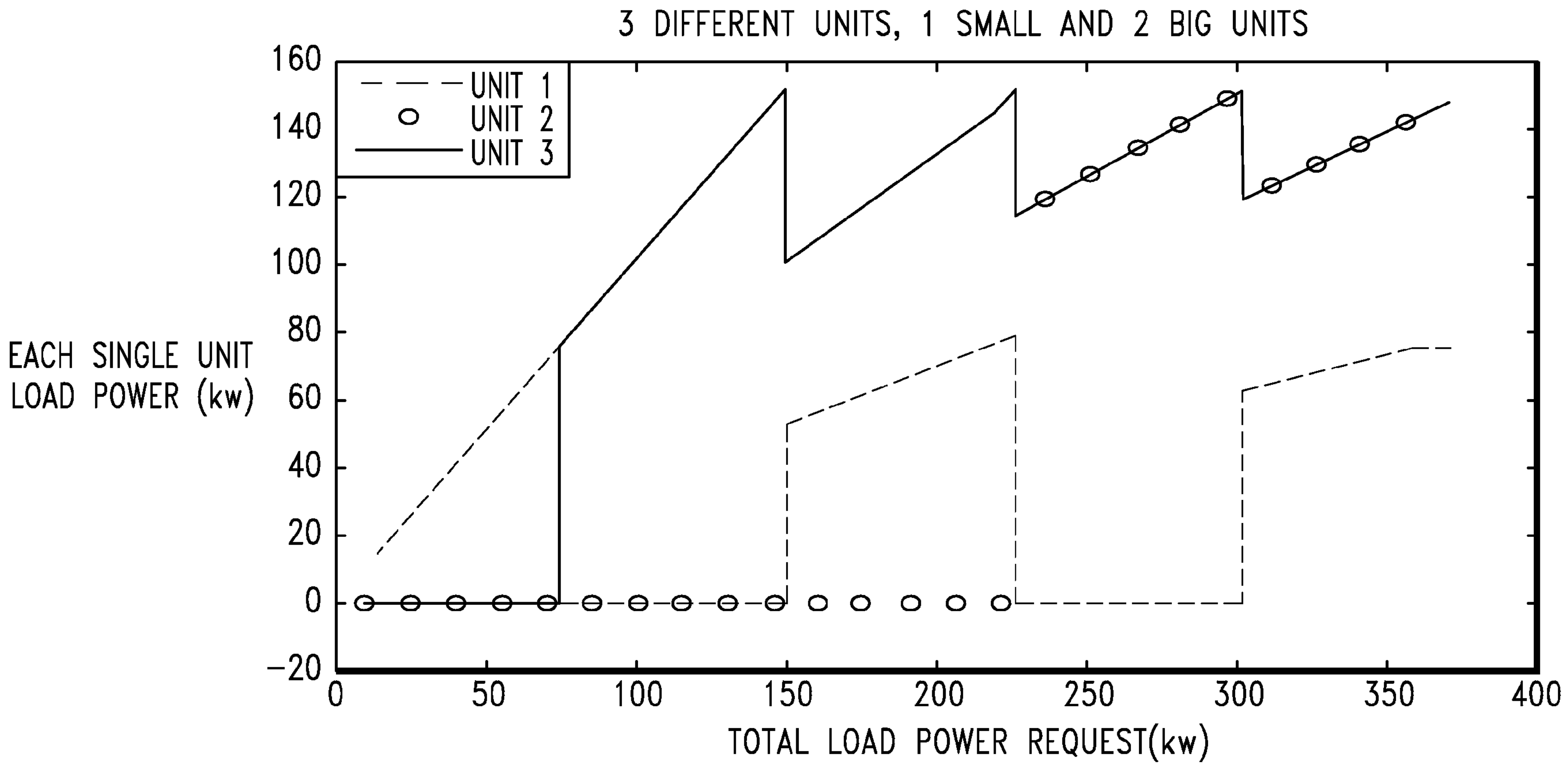

3 DIFFERENT UNITS, 1 SMALL AND 2 BIG UNITS
UNIT 1
UNIT 2
UNIT 3
EACH SINGLE UNIT LOAD POWER (kw)
160
140
120
100
80
60
40
20
0
−20
0
50
100
150
200
250
300
350
400
TOTAL LOAD POWER REQUEST(kw)

Fig. 13

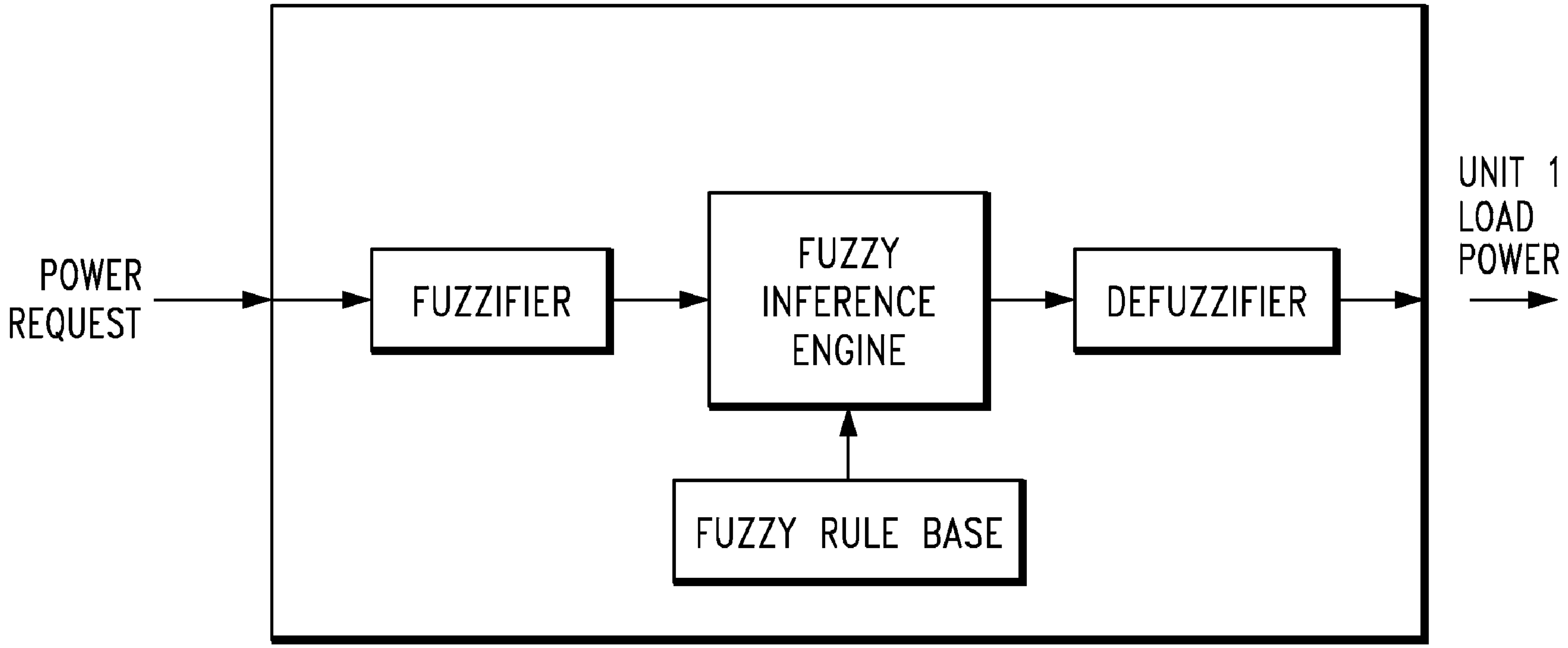

POWER
REQUEST
FUZZIFIER
FUZZY
INFERENCE
ENGINE
FUZZY RULE BASE
DEFUZZIFIER
UNIT 1
LOAD
POWER

Fig. 14

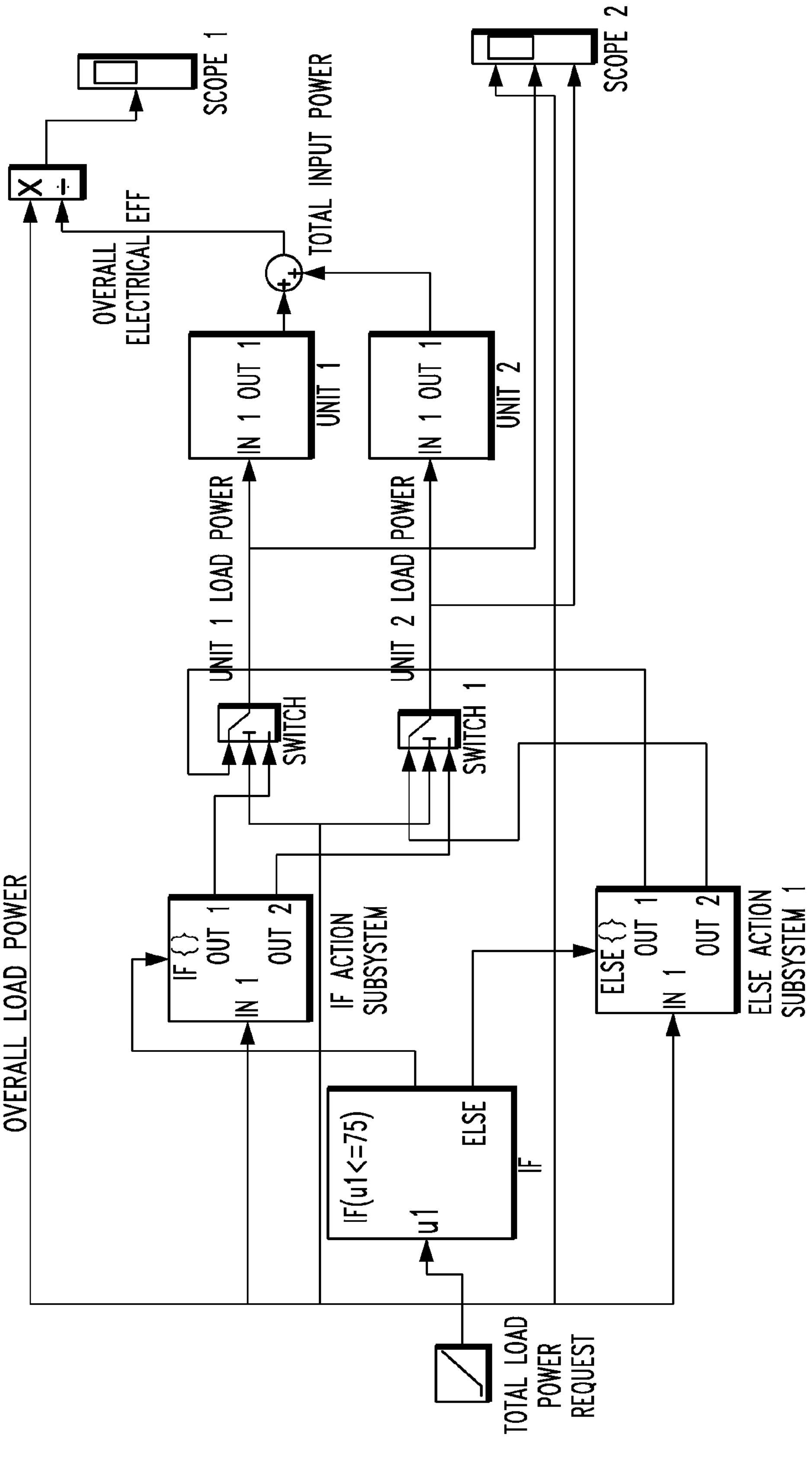

TOTAL LOAD
POWER
REQUEST
OVERALL LOAD POWER
u1
IF(u1<=75)
ELSE
IF
IF { }
IN 1
OUT 1
OUT 2
IF ACTION
SUBSYSTEM
ELSE { }
IN 1
OUT 1
OUT 2
ELSE ACTION
SUBSYSTEM 1
SWITCH
SWITCH 1
UNIT 1 LOAD POWER
UNIT 2 LOAD POWER
IN 1 OUT 1
UNIT 1
IN 1 OUT 1
UNIT 2
+
+
TOTAL INPUT POWER
OVERALL
ELECTRICAL EFF
X
÷
SCOPE 1
SCOPE 2

Fig. 15

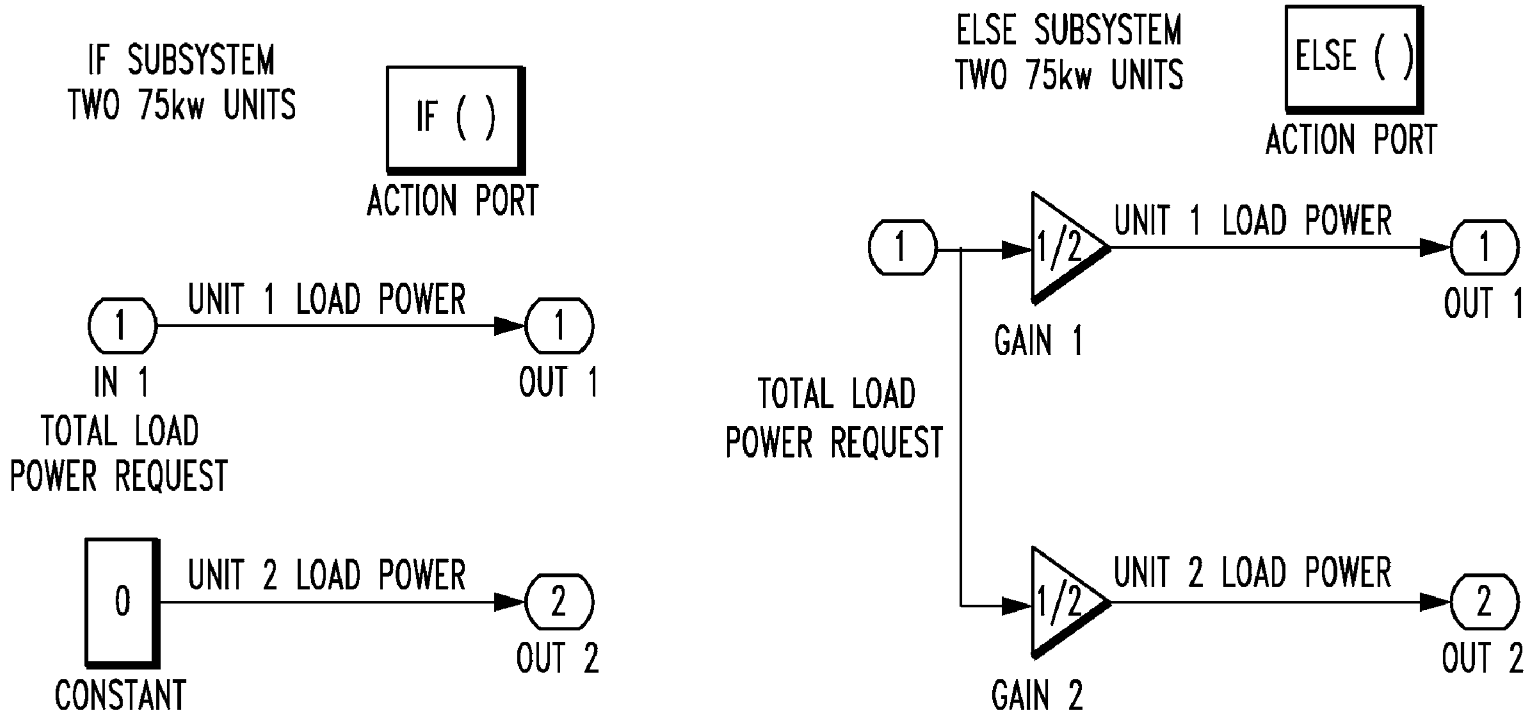

IF SUBSYSTEM
TWO 75kw UNITS
IF ( )
ACTION PORT
1
IN 1
TOTAL LOAD
POWER REQUEST
UNIT 1 LOAD POWER
1
OUT 1
0
CONSTANT
UNIT 2 LOAD POWER
2
OUT 2
ELSE SUBSYSTEM
TWO 75kw UNITS
ELSE ( )
ACTION PORT
1
TOTAL LOAD
POWER REQUEST
1/2
GAIN 1
UNIT 1 LOAD POWER
1
OUT 1
1/2
GAIN 2
UNIT 2 LOAD POWER
2
OUT 2

Fig. 16

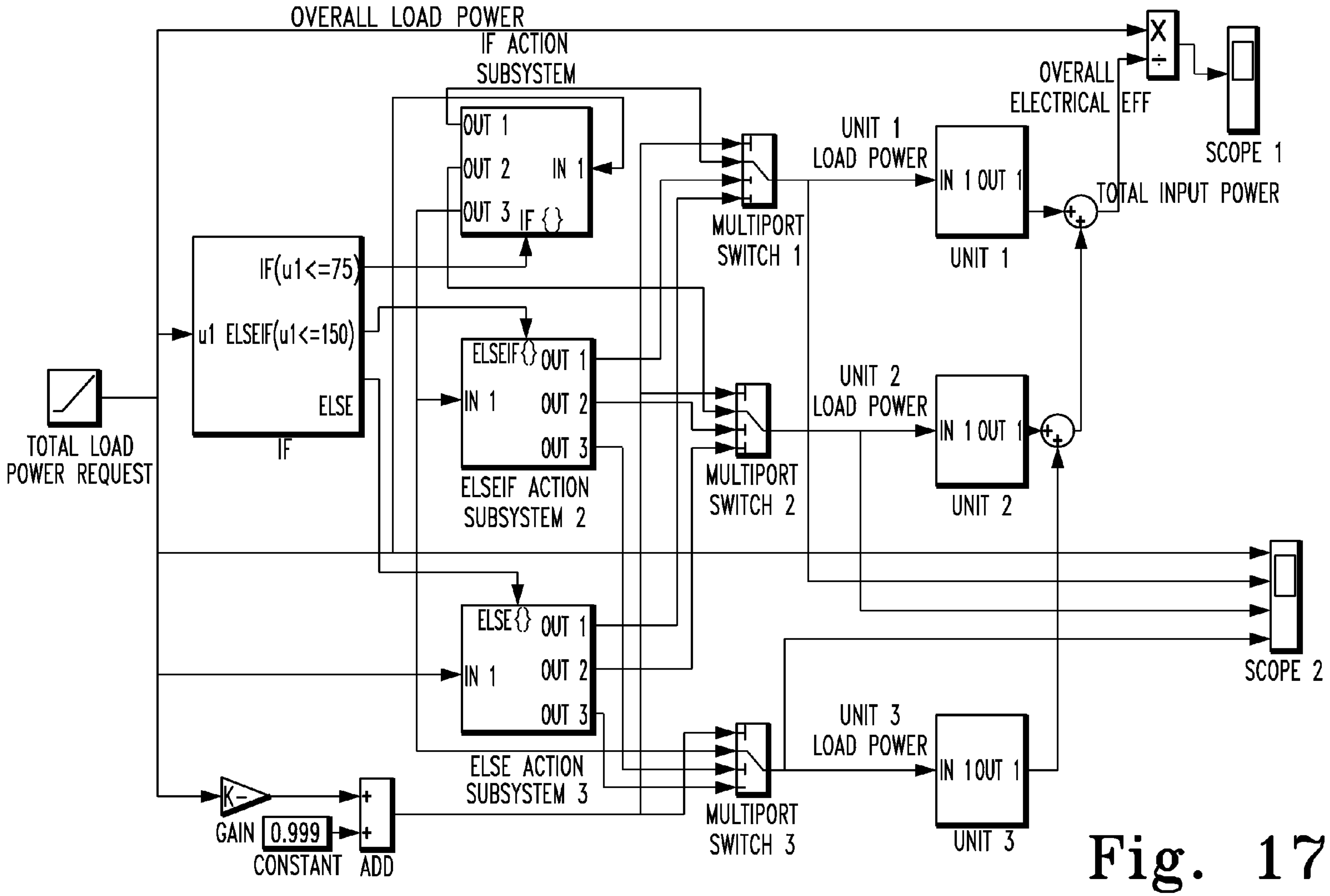

OVERALL LOAD POWER
IF ACTION SUBSYSTEM
OUT 1
OUT 2
IN 1
OUT 3
IF { }
IF(u1<=75)
u1 ELSEIF(u1<=150)
ELSE
IF
TOTAL LOAD POWER REQUEST
ELSEIF { }
OUT 1
IN 1
OUT 2
OUT 3
ELSEIF ACTION SUBSYSTEM 2
ELSE { }
OUT 1
IN 1
OUT 2
OUT 3
ELSE ACTION SUBSYSTEM 3
K-
GAIN
0.999
CONSTANT
+
+
ADD
MULTIPORT SWITCH 1
MULTIPORT SWITCH 2
MULTIPORT SWITCH 3
UNIT 1 LOAD POWER
UNIT 2 LOAD POWER
UNIT 3 LOAD POWER
IN 1 OUT 1
UNIT 1
IN 1 OUT 1
UNIT 2
IN 1OUT 1
UNIT 3
+
+
TOTAL INPUT POWER
OVERALL ELECTRICAL EFF
X
÷
SCOPE 1
SCOPE 2

Fig. 17

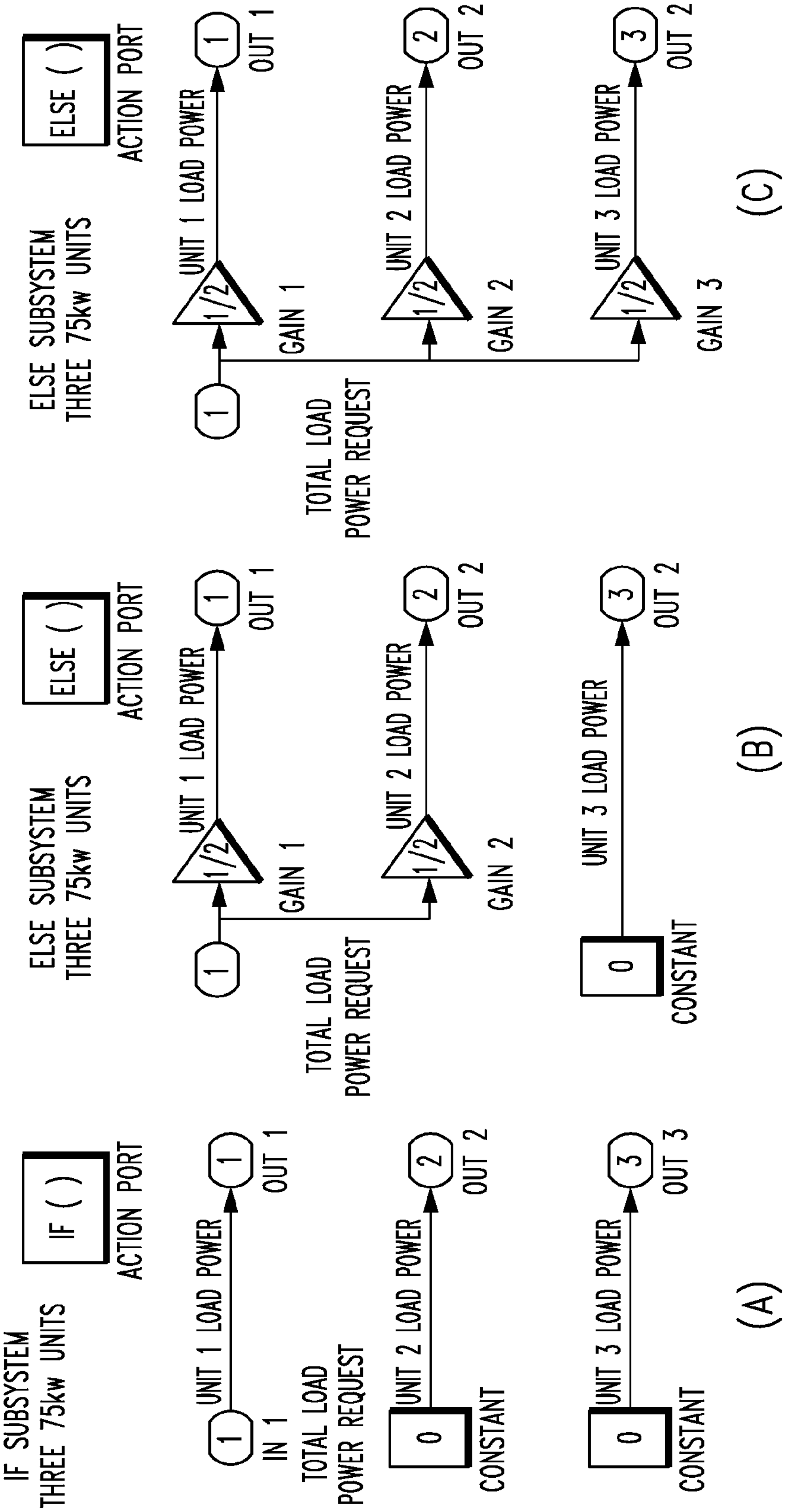

IF SUBSYSTEM
THREE 75kw UNITS
IF ( )
ACTION PORT
1
IN 1
TOTAL LOAD
POWER REQUEST
UNIT 1 LOAD POWER
1
OUT 1
0
CONSTANT
UNIT 2 LOAD POWER
2
OUT 2
0
CONSTANT
UNIT 3 LOAD POWER
3
OUT 3
(A)
ELSE SUBSYSTEM
THREE 75kw UNITS
ELSE ( )
ACTION PORT
1
TOTAL LOAD
POWER REQUEST
1/2
GAIN 1
UNIT 1 LOAD POWER
1
OUT 1
1/2
GAIN 2
UNIT 2 LOAD POWER
2
OUT 2
0
CONSTANT
UNIT 3 LOAD POWER
3
OUT 2
(B)
ELSE SUBSYSTEM
THREE 75kw UNITS
ELSE ( )
ACTION PORT
1
TOTAL LOAD
POWER REQUEST
1/2
GAIN 1
UNIT 1 LOAD POWER
1
OUT 1
1/2
GAIN 2
UNIT 2 LOAD POWER
2
OUT 2
1/2
GAIN 3
UNIT 3 LOAD POWER
3
OUT 2
(C)

Fig. 18

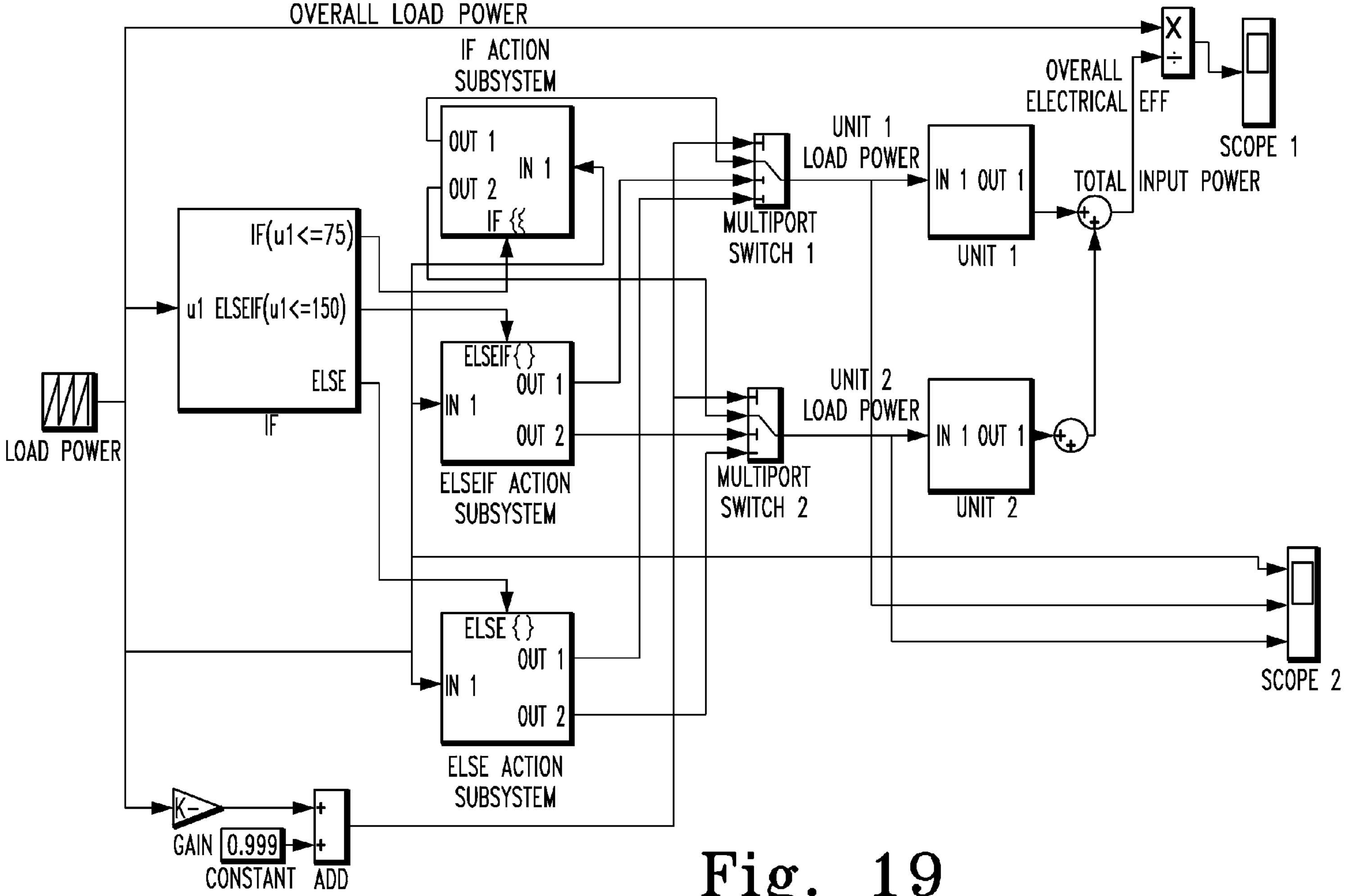

OVERALL LOAD POWER
IF ACTION SUBSYSTEM
OUT 1
OUT 2
IN 1
IF {}
IF(u1<=75)
u1 ELSEIF(u1<=150)
ELSE
IF
LOAD POWER
ELSEIF{ }
OUT 1
IN 1
OUT 2
ELSEIF ACTION SUBSYSTEM
ELSE{ }
OUT 1
IN 1
OUT 2
ELSE ACTION SUBSYSTEM
K-
GAIN
0.999
CONSTANT
ADD
MULTIPORT SWITCH 1
MULTIPORT SWITCH 2
UNIT 1 LOAD POWER
UNIT 2 LOAD POWER
IN 1 OUT 1
UNIT 1
IN 1 OUT 1
UNIT 2
TOTAL INPUT POWER
OVERALL ELECTRICAL EFF
X
÷
SCOPE 1
SCOPE 2

Fig. 19

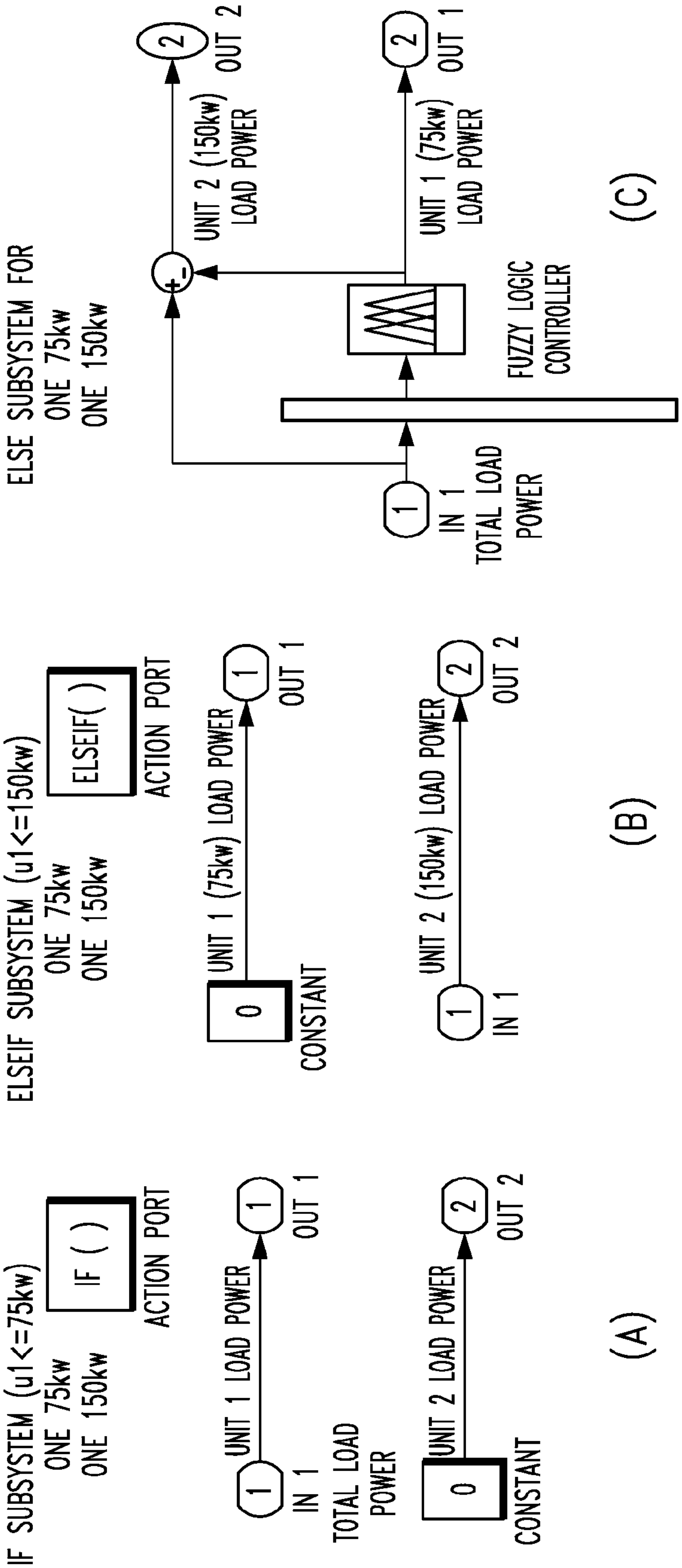

IF SUBSYSTEM (u1<=75kw)
ONE 75kw
ONE 150kw
IF ( )
ACTION PORT
1
IN 1
TOTAL LOAD
POWER
UNIT 1 LOAD POWER
1
OUT 1
0
CONSTANT
UNIT 2 LOAD POWER
2
OUT 2
(A)
ELSEIF SUBSYSTEM (u1<=150kw)
ONE 75kw
ONE 150kw
ELSEIF( )
ACTION PORT
0
CONSTANT
UNIT 1 (75kw) LOAD POWER
1
OUT 1
1
IN 1
UNIT 2 (150kw) LOAD POWER
2
OUT 2
(B)
ELSE SUBSYSTEM FOR
ONE 75kw
ONE 150kw
1
IN 1
TOTAL LOAD
POWER
FUZZY LOGIC
CONTROLLER
+
-
UNIT 2 (150kw)
LOAD POWER
2
OUT 2
UNIT 1 (75kw)
LOAD POWER
2
OUT 1
(C)

Fig. 20

Fig. 21

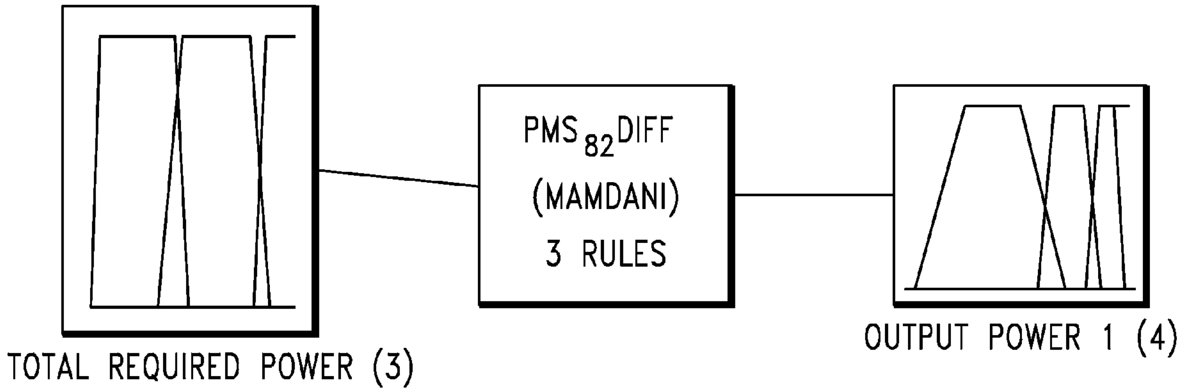
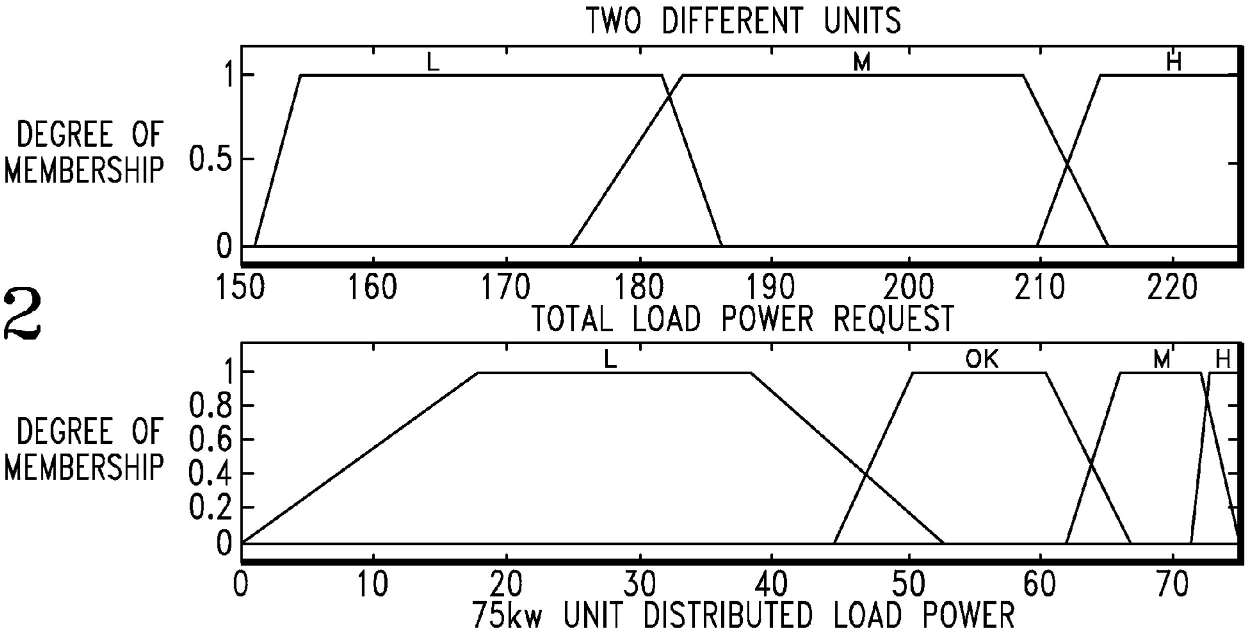

TOTAL REQUIRED POWER (3)
$PMS_{82}DIFF$
(MAMDANI)
3 RULES
OUTPUT POWER 1 (4)

Fig. 22


TWO DIFFERENT UNITS
DEGREE OF MEMBERSHIP
1
0.5
0
L
M
H
150
160
170
180
190
200
210
220
TOTAL LOAD POWER REQUEST
DEGREE OF MEMBERSHIP
1
0.8
0.6
0.4
0.2
0
L
OK
M
H
0
10
20
30
40
50
60
70
75kw UNIT DISTRIBUTED LOAD POWER

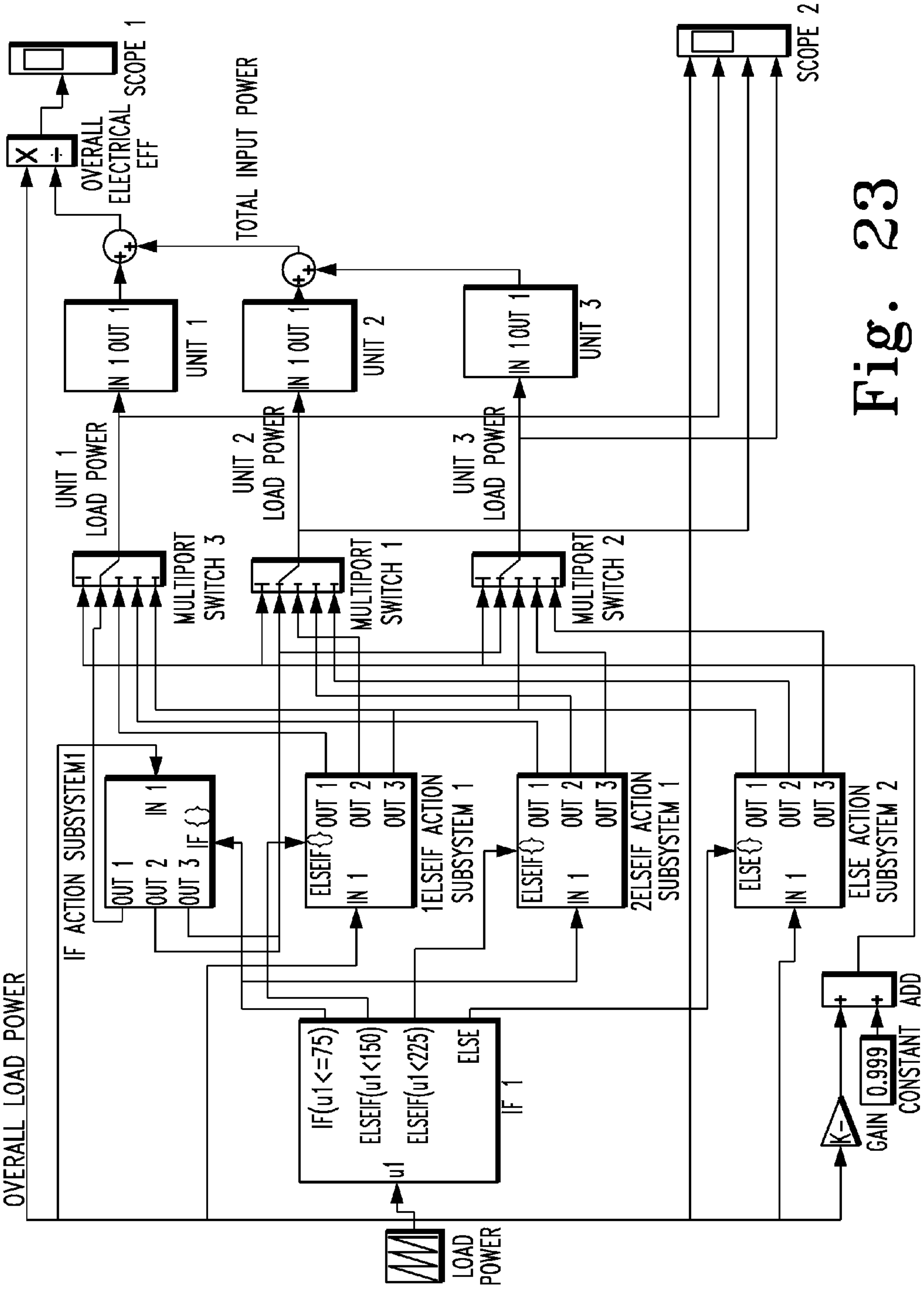

OVERALL LOAD POWER
LOAD POWER
IF(u1<=75)
ELSEIF(u1<150)
ELSEIF(u1<225)
ELSE
u1
IF 1
IF ACTION SUBSYSTEM1
OUT 1
OUT 2
OUT 3
IN 1
IF { }
ELSEIF { }
OUT 1
OUT 2
OUT 3
IN 1
1ELSEIF ACTION SUBSYSTEM 1
ELSEIF { }
OUT 1
OUT 2
OUT 3
IN 1
2ELSEIF ACTION SUBSYSTEM 1
ELSE { }
OUT 1
OUT 2
OUT 3
IN 1
ELSE ACTION SUBSYSTEM 2
K-
GAIN
0.999
CONSTANT
ADD
+
+
MULTIPORT SWITCH 3
MULTIPORT SWITCH 1
MULTIPORT SWITCH 2
UNIT 1 LOAD POWER
UNIT 2 LOAD POWER
UNIT 3 LOAD POWER
IN 1 OUT 1
UNIT 1
IN 1 OUT 1
UNIT 2
IN 1 OUT 1
UNIT 3
+
+
+
+
TOTAL INPUT POWER
X
÷
OVERALL ELECTRICAL EFF
SCOPE 1
SCOPE 2

Fig. 23

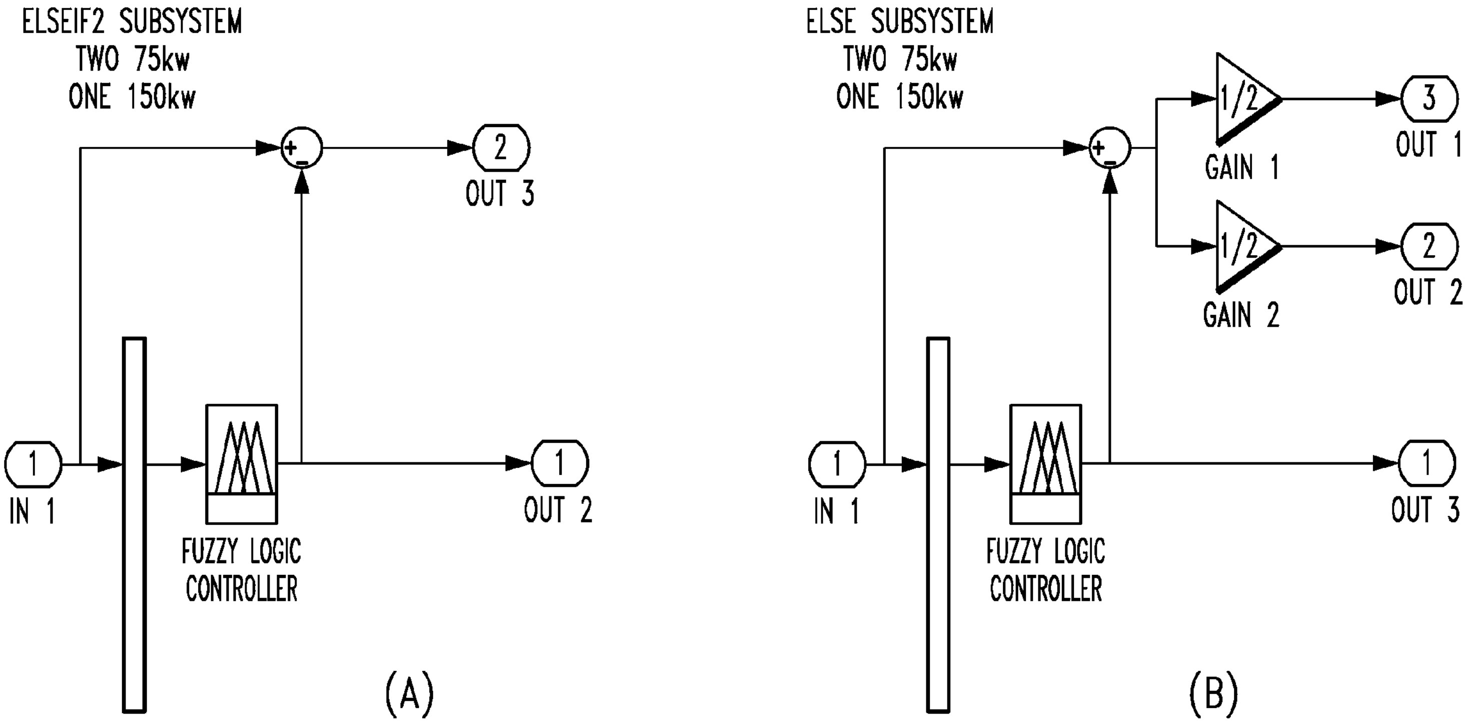

ELSEIF2 SUBSYSTEM
TWO 75kw
ONE 150kw
+
-
2
OUT 3
1
IN 1
FUZZY LOGIC
CONTROLLER
1
OUT 2
(A)
ELSE SUBSYSTEM
TWO 75kw
ONE 150kw
+
-
1/2
GAIN 1
1/2
GAIN 2
3
OUT 1
2
OUT 2
1
IN 1
FUZZY LOGIC
CONTROLLER
1
OUT 3
(B)

Fig. 24

Fig. 25

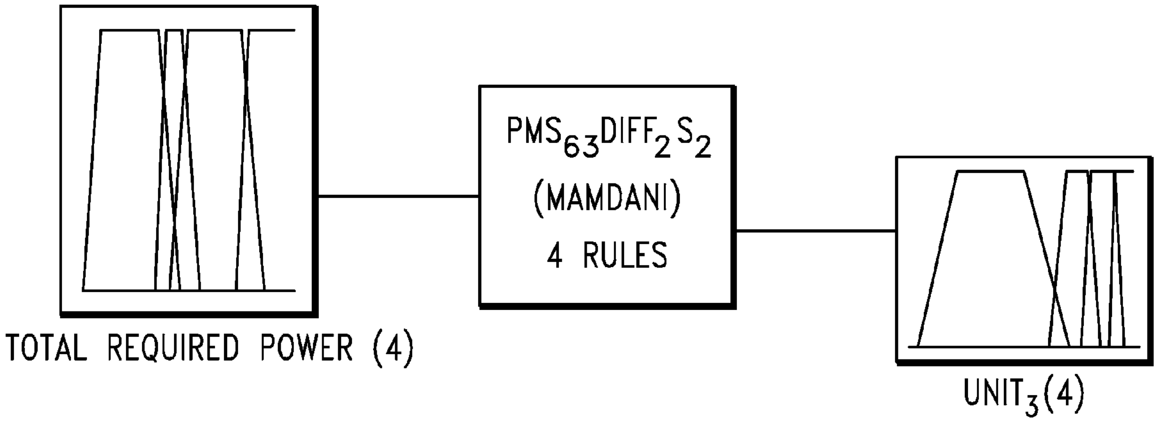
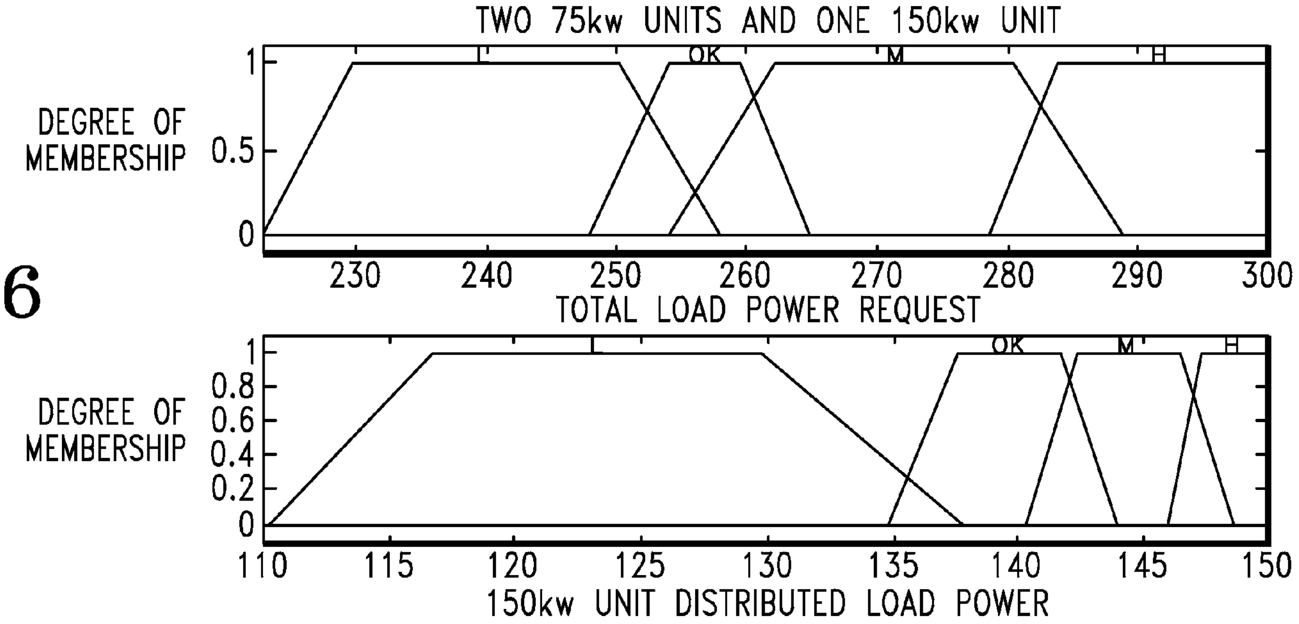
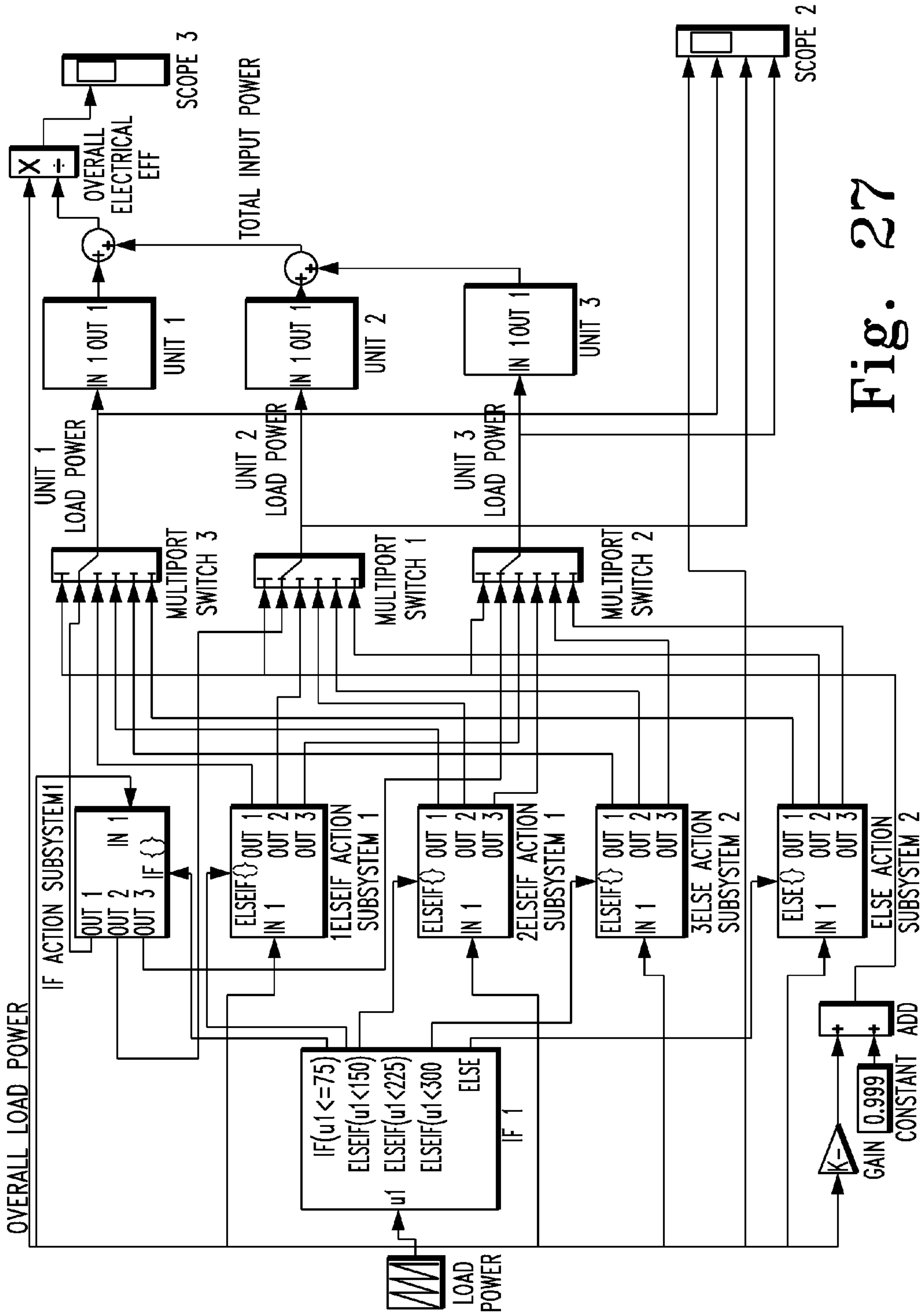

TOTAL REQUIRED POWER (4)
PMS$_{63}$DIFF$_2$S$_2$
(MAMDANI)
4 RULES
UNIT$_3$(4)

Fig. 26


TWO 75kw UNITS AND ONE 150kw UNIT
DEGREE OF MEMBERSHIP
1
0.5
0
L
OK
M
H
230
240
250
260
270
280
290
300
TOTAL LOAD POWER REQUEST
DEGREE OF MEMBERSHIP
1
0.8
0.6
0.4
0.2
0
L
OK
M
H
110
115
120
125
130
135
140
145
150
150kw UNIT DISTRIBUTED LOAD POWER



LOAD POWER
u1
IF(u1<=75)
ELSEIF(u1<150)
ELSEIF(u1<225)
ELSEIF(u1<300
ELSE
IF 1
OVERALL LOAD POWER
IF ACTION SUBSYSTEM1
OUT 1
OUT 2
OUT 3
IN 1
IF{}
ELSEIF{}
IN 1
OUT 1
OUT 2
OUT 3
1ELSEIF ACTION SUBSYSTEM 1
2ELSEIF ACTION SUBSYSTEM 1
3ELSE ACTION SUBSYSTEM 2
ELSE{}
ELSE ACTION SUBSYSTEM 2
K-
GAIN
0.999
CONSTANT
ADD
MULTIPORT SWITCH 3
MULTIPORT SWITCH 1
MULTIPORT SWITCH 2
UNIT 1 LOAD POWER
UNIT 2 LOAD POWER
UNIT 3 LOAD POWER
IN 1 OUT 1
UNIT 1
UNIT 2
UNIT 3
TOTAL INPUT POWER
X
÷
OVERALL ELECTRICAL EFF
SCOPE 3
SCOPE 2

Fig. 27

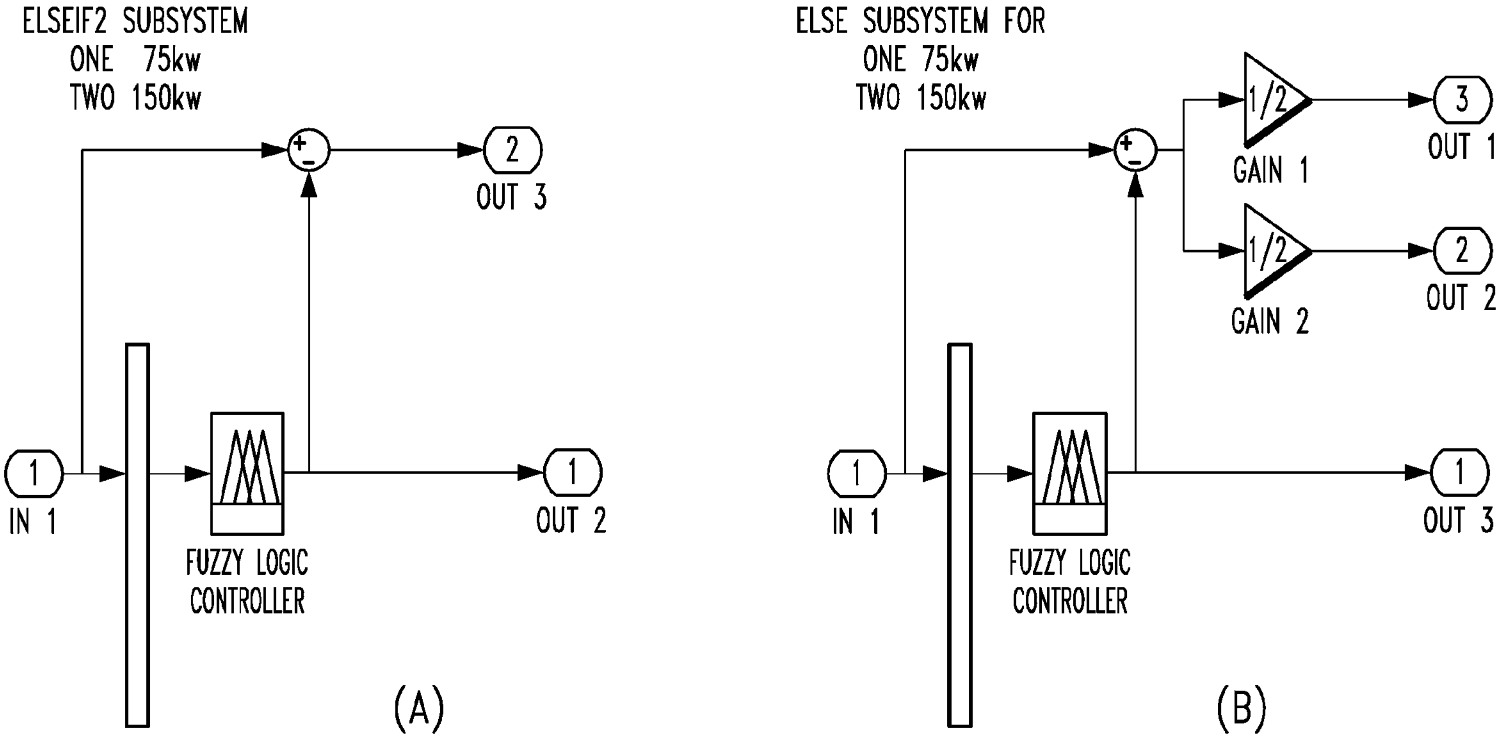

ELSEIF2 SUBSYSTEM
ONE 75kw
TWO 150kw
1
IN 1
FUZZY LOGIC
CONTROLLER
+
-
2
OUT 3
1
OUT 2
(A)
ELSE SUBSYSTEM FOR
ONE 75kw
TWO 150kw
1
IN 1
FUZZY LOGIC
CONTROLLER
+
-
1/2
GAIN 1
1/2
GAIN 2
3
OUT 1
2
OUT 2
1
OUT 3
(B)

Fig. 28

Fig. 29

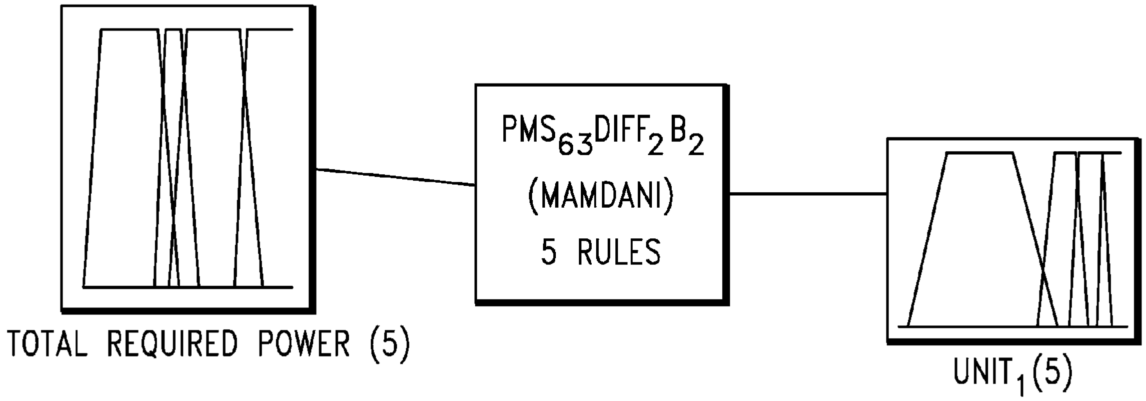
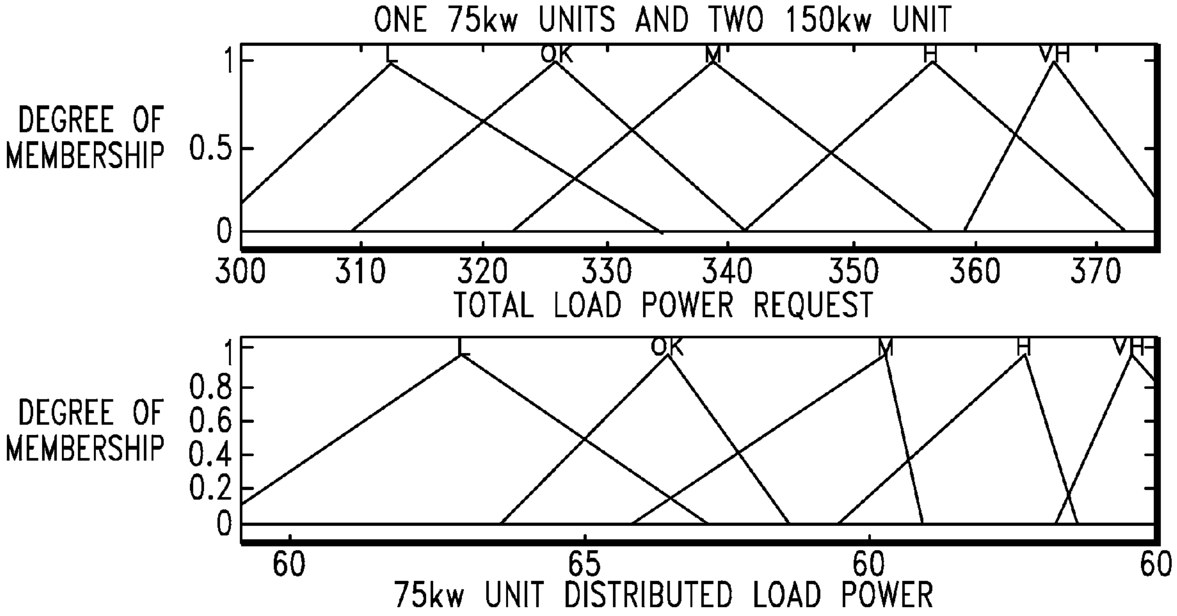

TOTAL REQUIRED POWER (5)
$PMS_{63}DIFF_2B_2$
(MAMDANI)
5 RULES
$UNIT_1(5)$

Fig. 30


ONE 75kw UNITS AND TWO 150kw UNIT
DEGREE OF MEMBERSHIP
1
0.5
0
L
OK
M
H
VH
300
310
320
330
340
350
360
370
TOTAL LOAD POWER REQUEST
DEGREE OF MEMBERSHIP
1
0.8
0.6
0.4
0.2
0
L
OK
M
H
VH
60
65
60
60
75kw UNIT DISTRIBUTED LOAD POWER

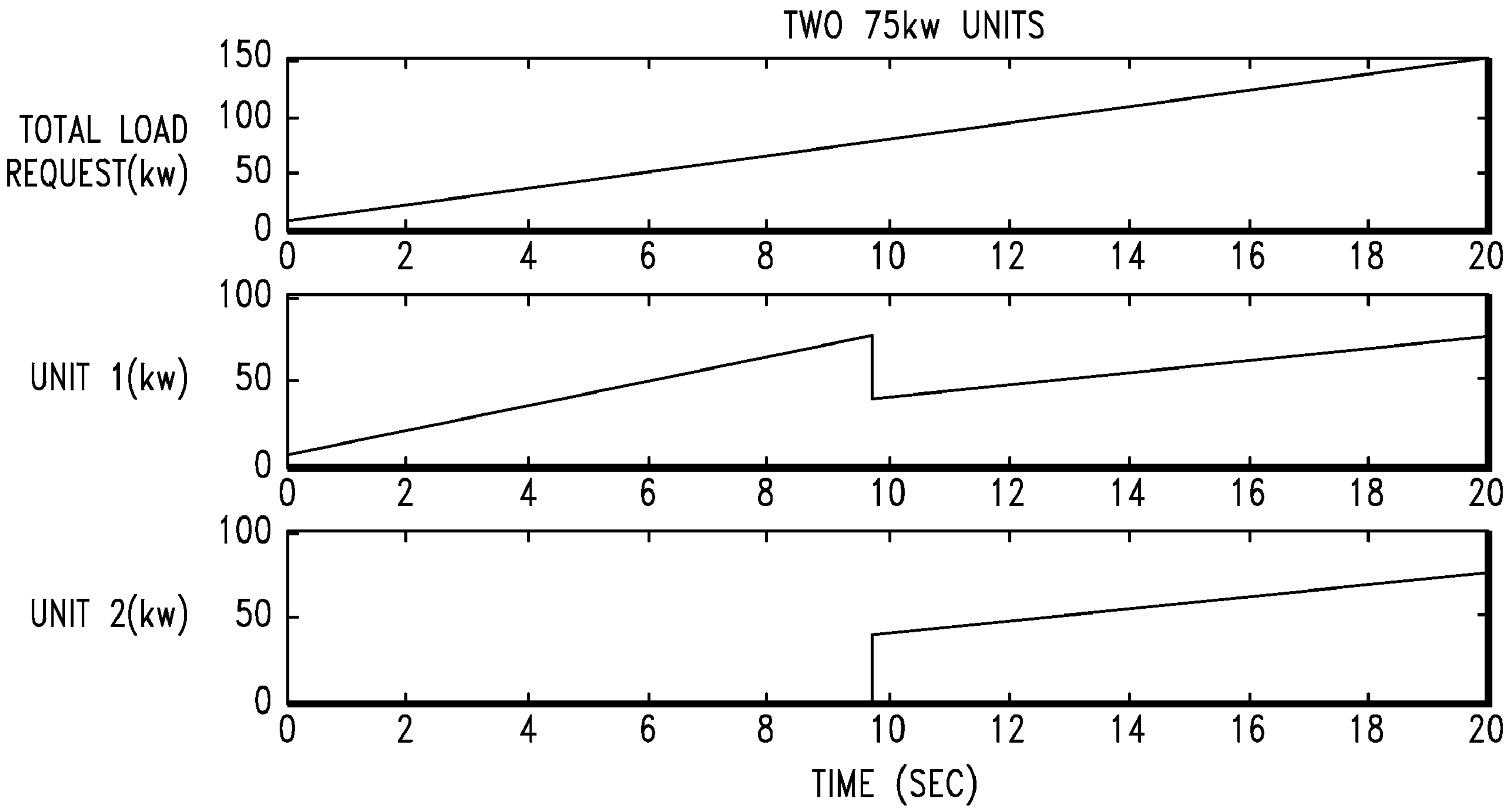

TWO 75kw UNITS
TOTAL LOAD REQUEST(kw)
150
100
50
0
UNIT 1(kw)
100
50
0
UNIT 2(kw)
100
50
0
0
2
4
6
8
10
12
14
16
18
20
TIME (SEC)

Fig. 31

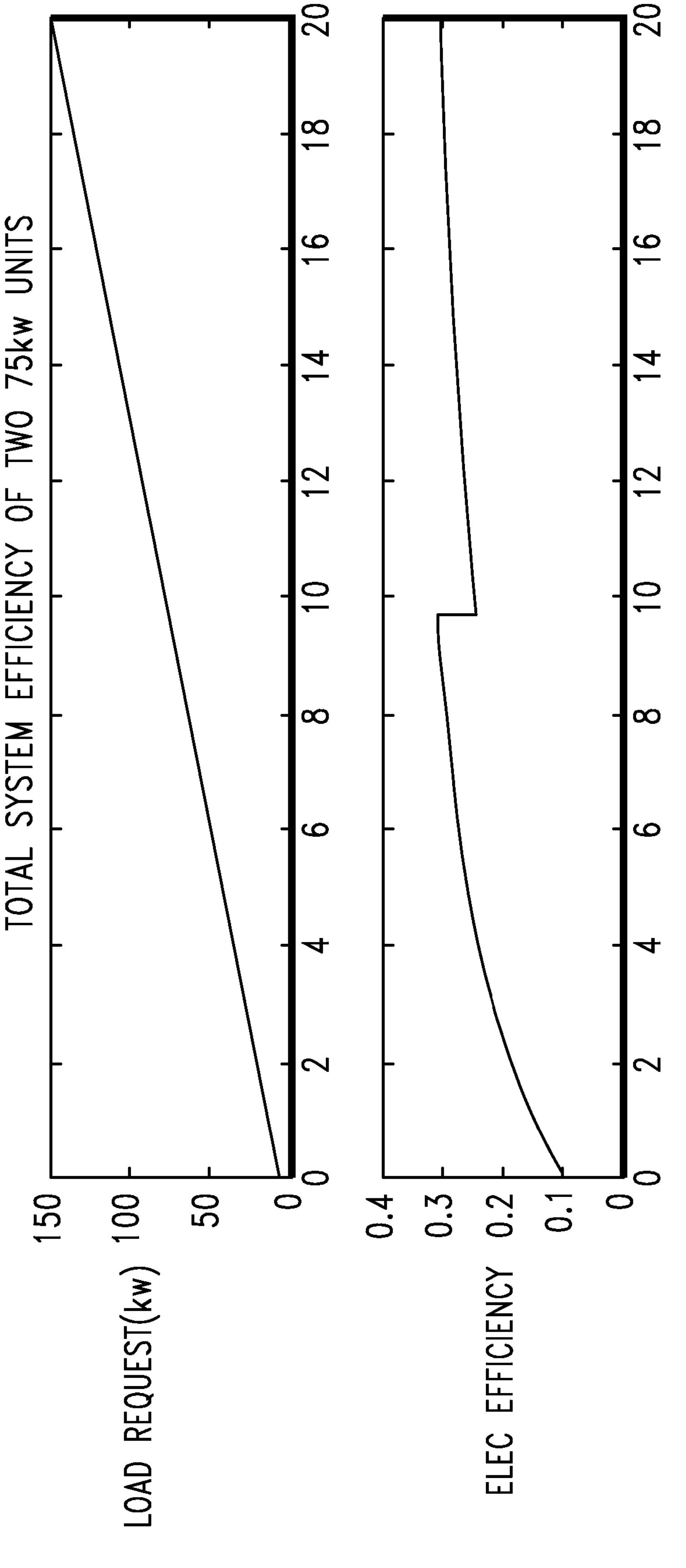

TOTAL SYSTEM EFFICIENCY OF TWO 75kw UNITS
LOAD REQUEST(kw)
150
100
50
0
0
2
4
6
8
10
12
14
16
18
20
ELEC EFFICIENCY
0.4
0.3
0.2
0.1
0
0
2
4
6
8
10
12
14
16
18
20

Fig. 32

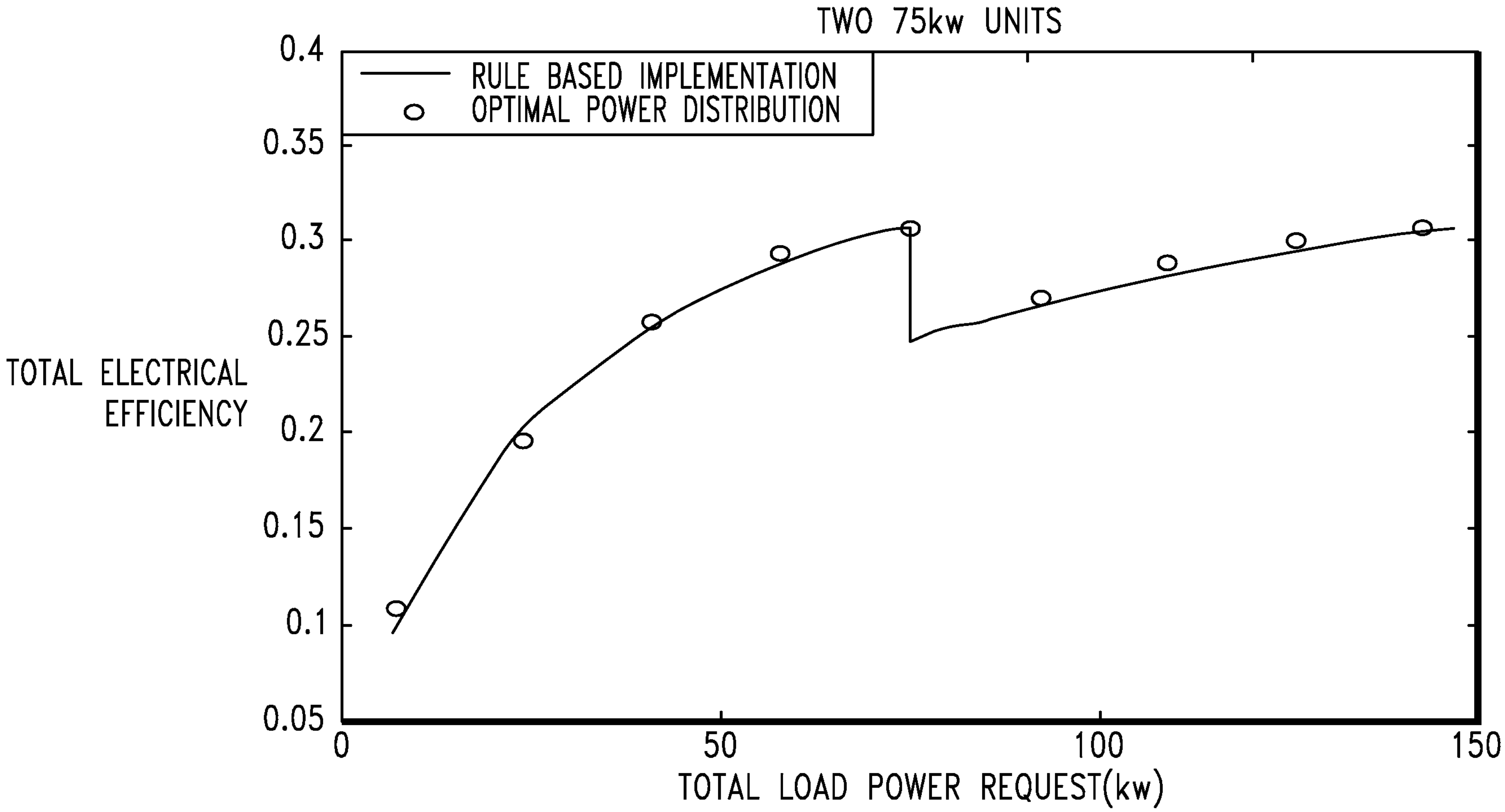

TWO 75kw UNITS
— RULE BASED IMPLEMENTATION
○ OPTIMAL POWER DISTRIBUTION
TOTAL ELECTRICAL EFFICIENCY
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
50
100
150
TOTAL LOAD POWER REQUEST(kw)

Fig. 33

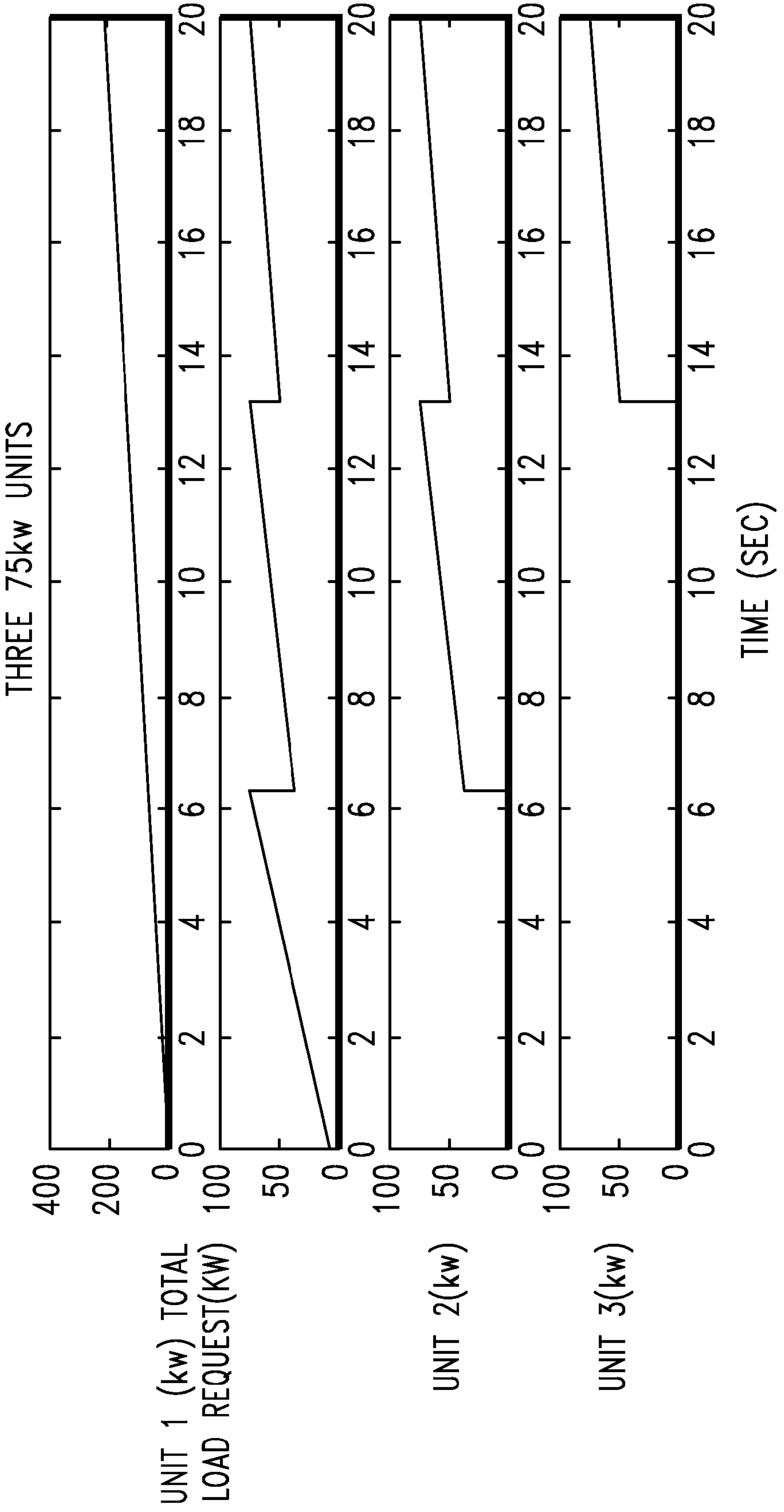

THREE 75kw UNITS
UNIT 1 (kw) TOTAL LOAD REQUEST(KW)
400
200
0
UNIT 2(kw)
UNIT 3(kw)
100
50
0
0
2
4
6
8
10
12
14
16
18
20
TIME (SEC)

Fig. 34

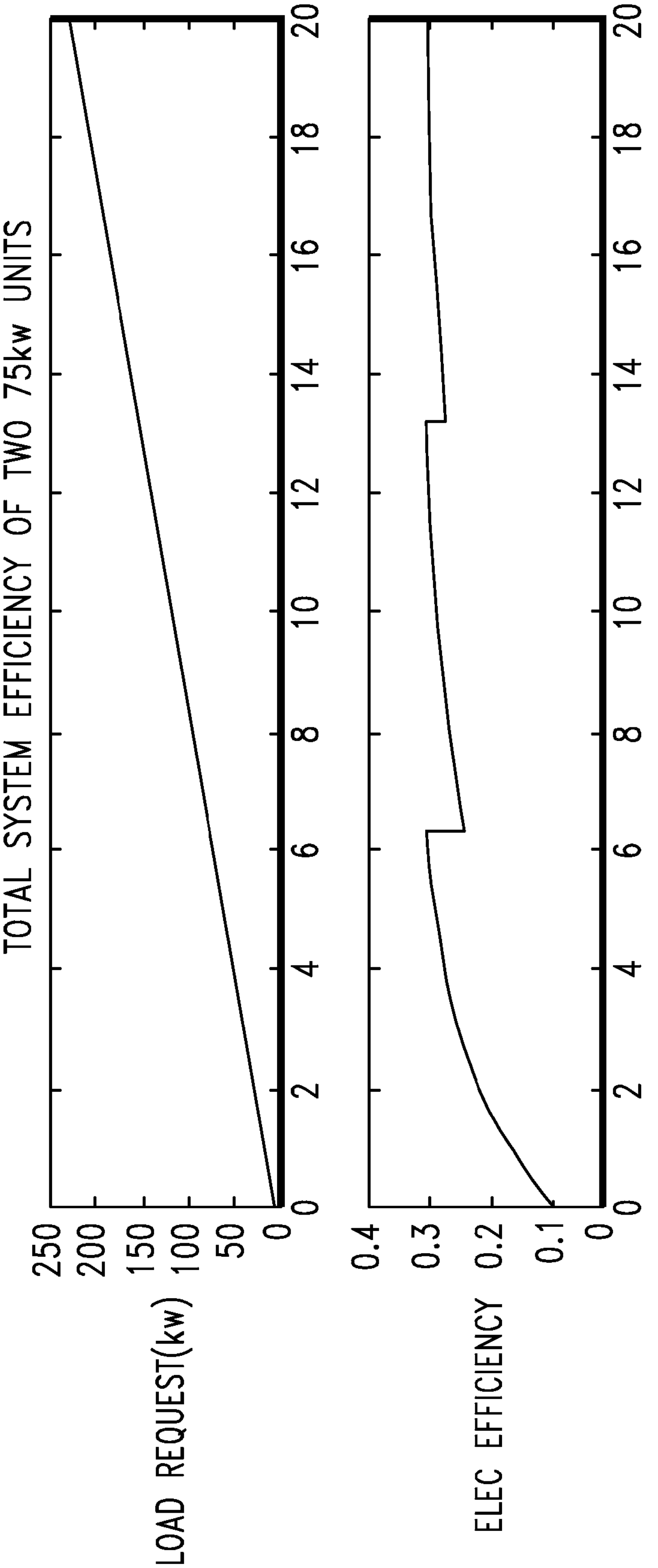

TOTAL SYSTEM EFFICIENCY OF TWO 75kw UNITS
LOAD REQUEST(kw)
250
200
150
100
50
0
ELEC EFFICIENCY
0.4
0.3
0.2
0.1
0
0
2
4
6
8
10
12
14
16
18
20

Fig. 35

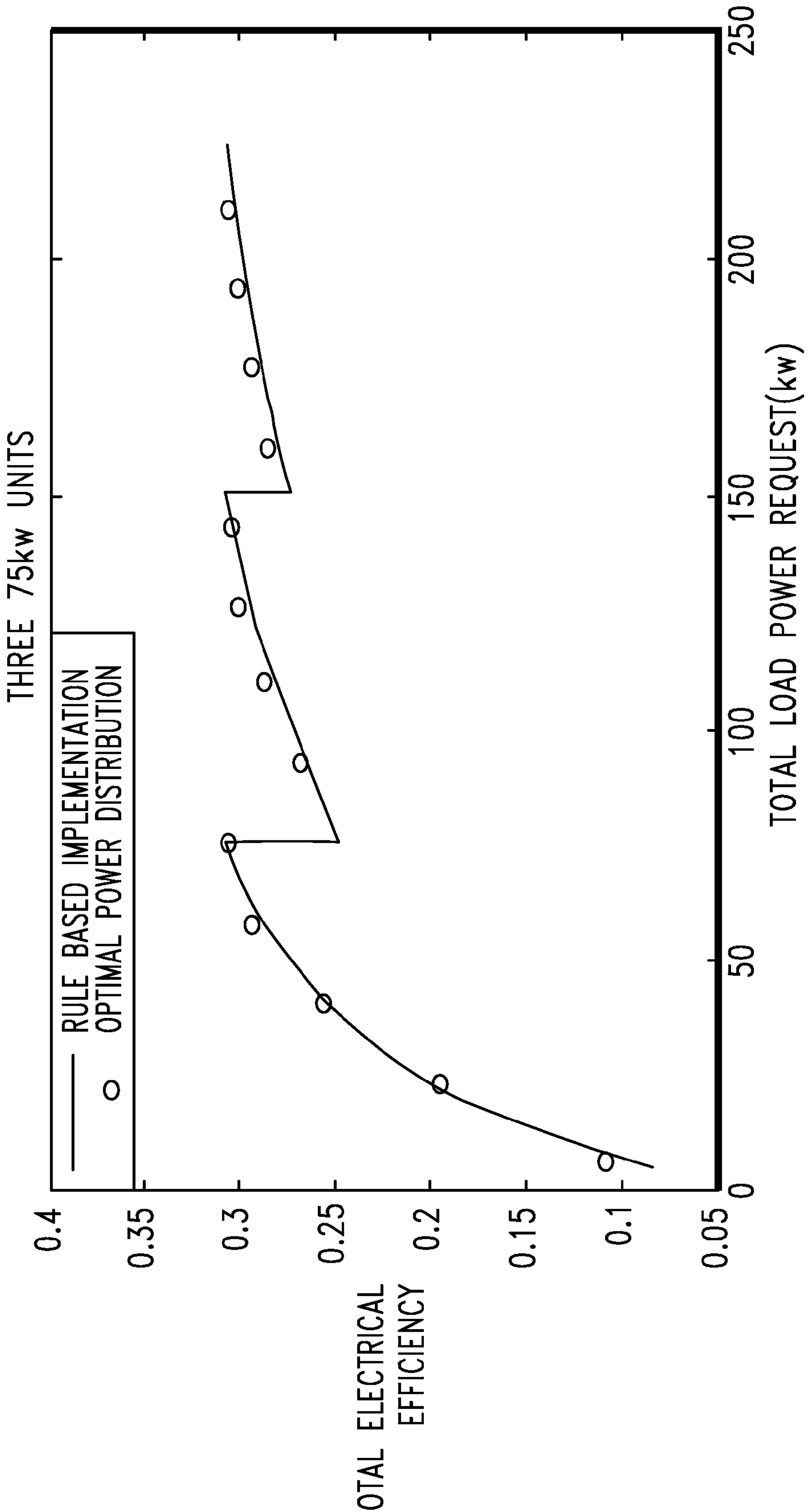

THREE 75kw UNITS
RULE BASED IMPLEMENTATION
OPTIMAL POWER DISTRIBUTION
TOTAL ELECTRICAL EFFICIENCY
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
50
100
150
200
250
TOTAL LOAD POWER REQUEST(kw)

Fig. 36

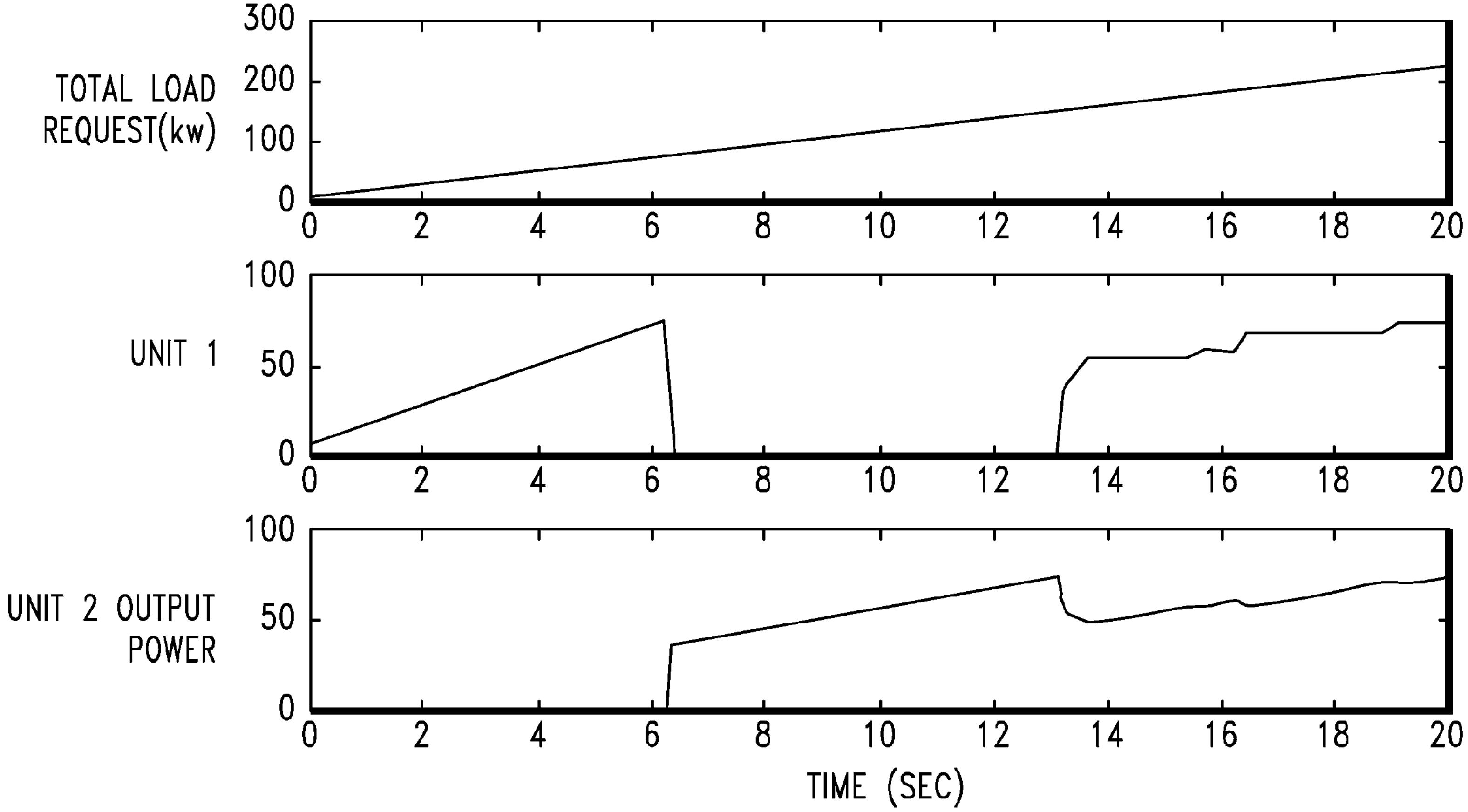

TOTAL LOAD
REQUEST(kw)
300
200
100
0
UNIT 1
100
50
0
UNIT 2 OUTPUT
POWER
100
50
0
0
2
4
6
8
10
12
14
16
18
20
TIME (SEC)

Fig. 37

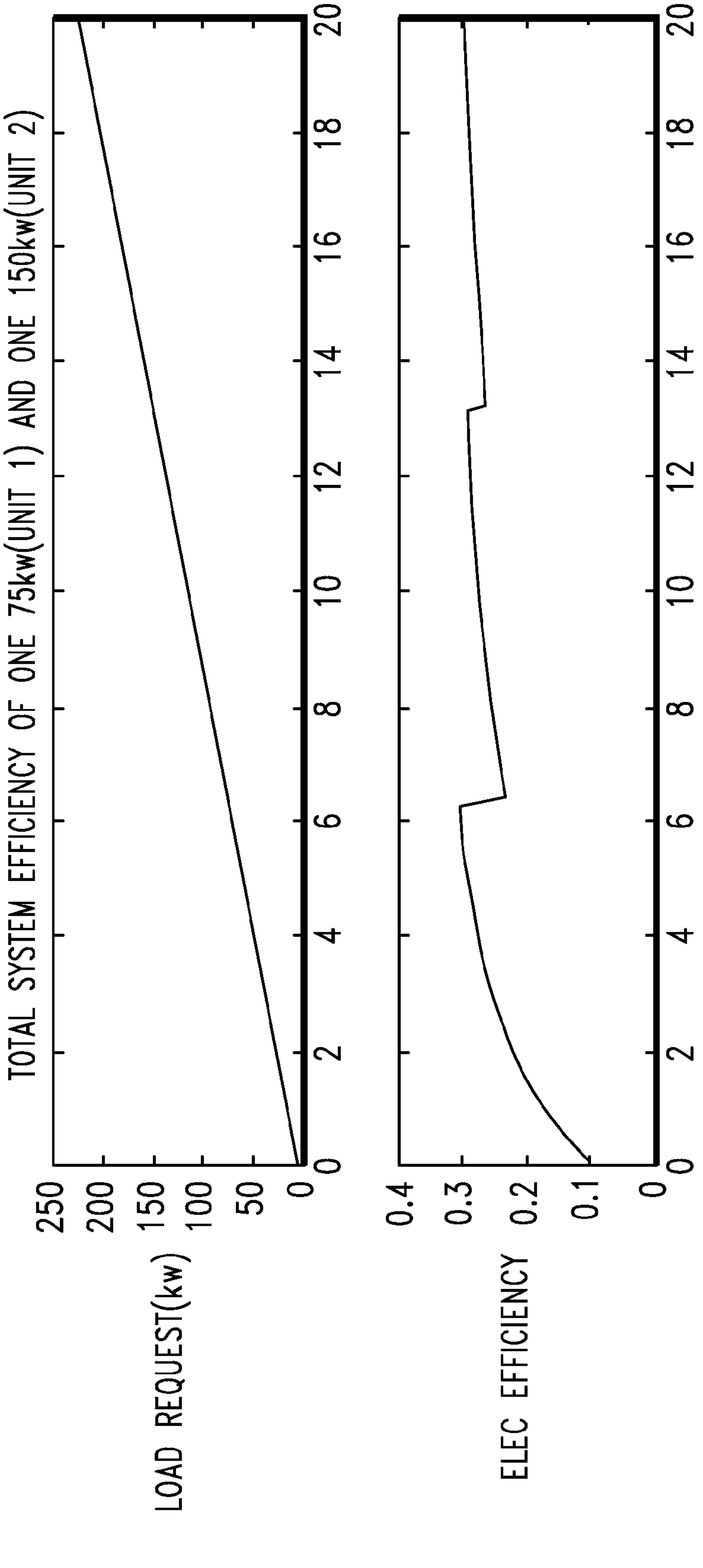

TOTAL SYSTEM EFFICIENCY OF ONE 75kw(UNIT 1) AND ONE 150kw(UNIT 2)
LOAD REQUEST(kw)
250
200
150
100
50
0
0
2
4
6
8
10
12
14
16
18
20
ELEC EFFICIENCY
0.4
0.3
0.2
0.1
0
0
2
4
6
8
10
12
14
16
18
20

Fig. 38

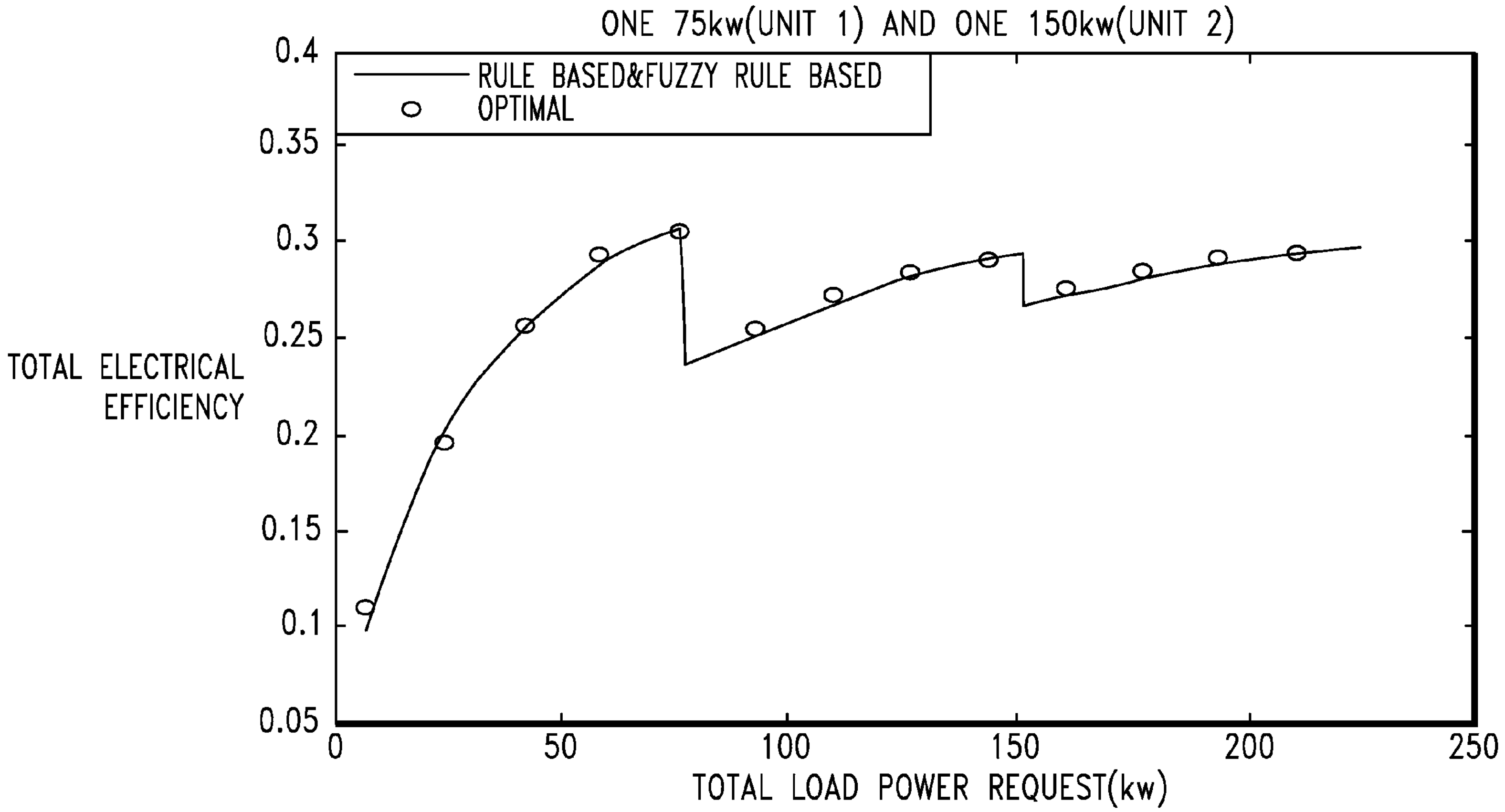

ONE 75kw(UNIT 1) AND ONE 150kw(UNIT 2)
RULE BASED&FUZZY RULE BASED
○ OPTIMAL
TOTAL ELECTRICAL EFFICIENCY
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
50
100
150
200
250
TOTAL LOAD POWER REQUEST(kw)

Fig. 39

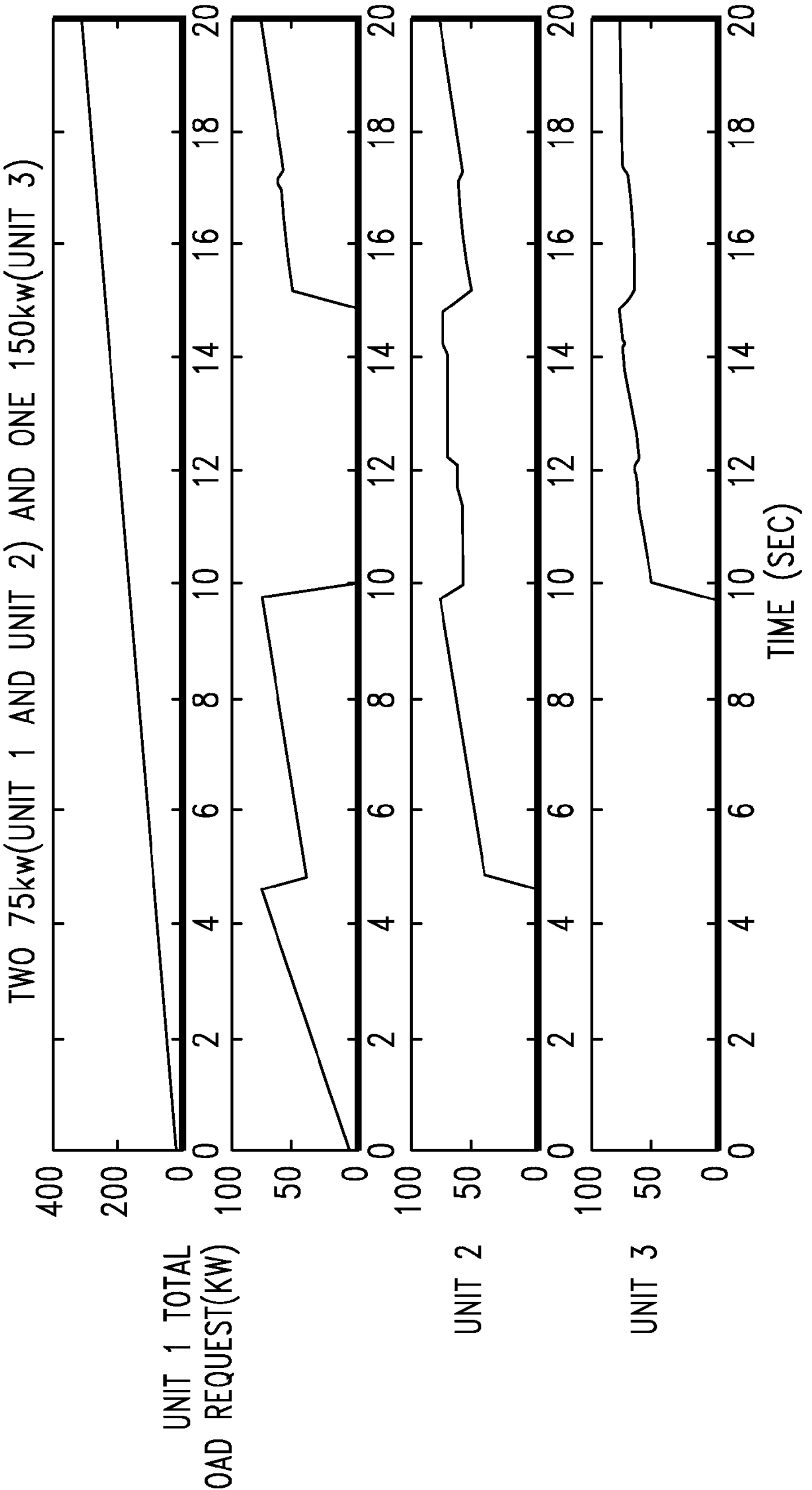

TWO 75kw(UNIT 1 AND UNIT 2) AND ONE 150kw(UNIT 3)
TOTAL LOAD REQUEST(KW)
400
200
0
UNIT 1
100
50
0
UNIT 2
100
50
0
UNIT 3
100
50
0
0
2
4
6
8
10
12
14
16
18
20
TIME (SEC)

Fig. 40

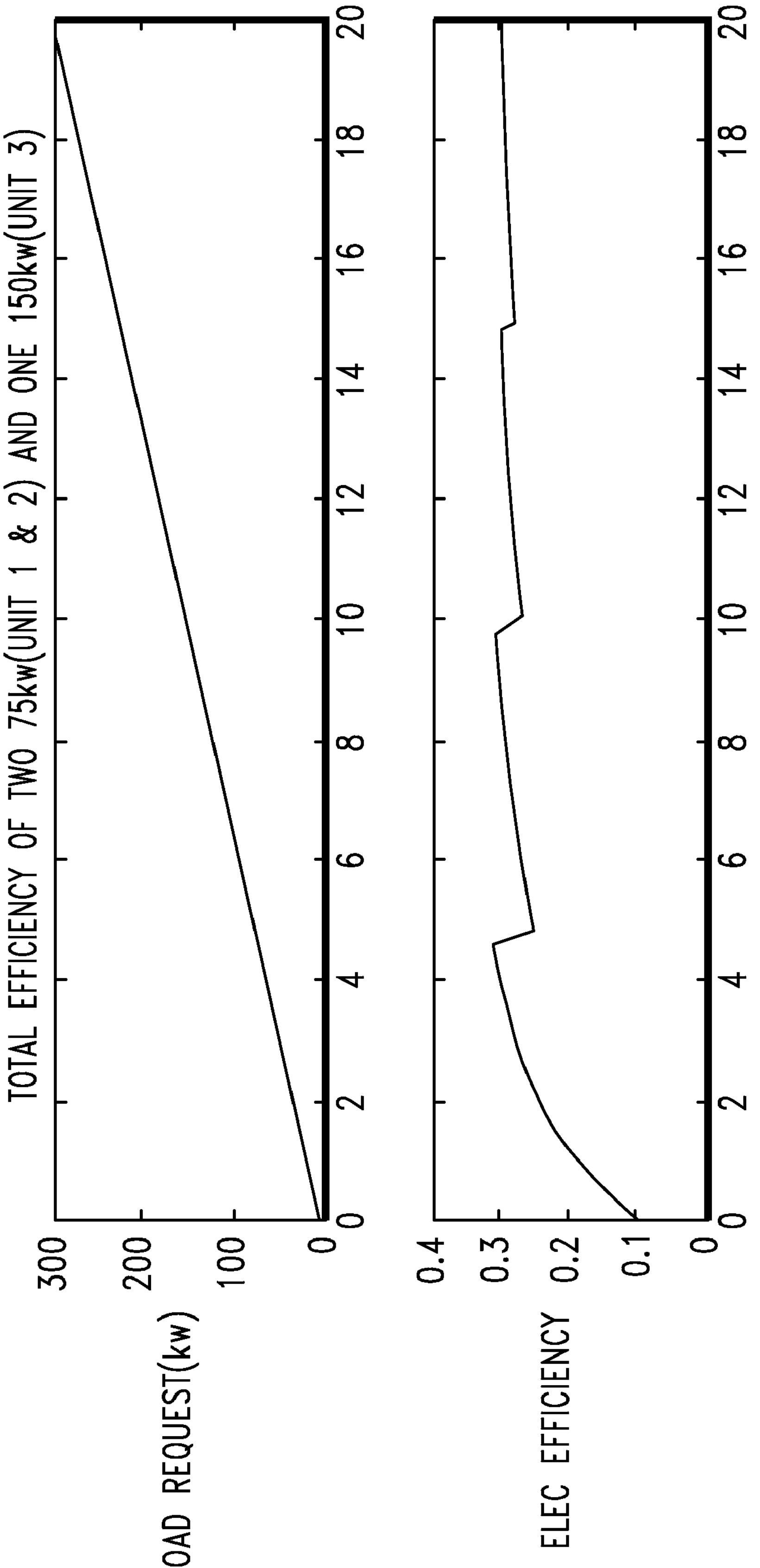

TOTAL EFFICIENCY OF TWO 75kw(UNIT 1 & 2) AND ONE 150kw(UNIT 3)
LOAD REQUEST(kw)
300
200
100
0
0
2
4
6
8
10
12
14
16
18
20
ELEC EFFICIENCY
0.4
0.3
0.2
0.1
0
0
2
4
6
8
10
12
14
16
18
20

Fig. 41

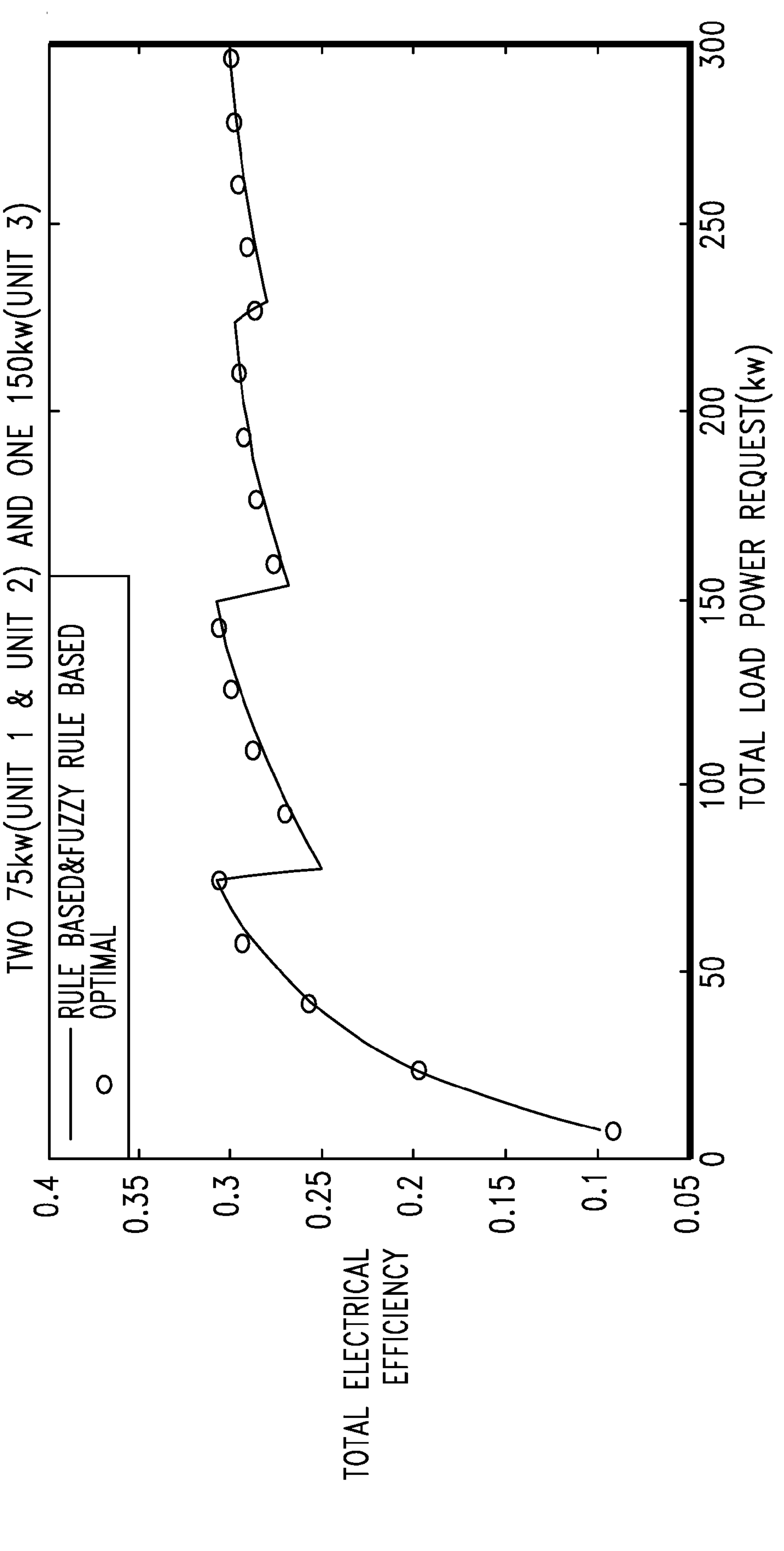

TWO 75kw(UNIT 1 & UNIT 2) AND ONE 150kw(UNIT 3)
RULE BASED&FUZZY RULE BASED
OPTIMAL
TOTAL ELECTRICAL EFFICIENCY
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
50
100
150
200
250
300
TOTAL LOAD POWER REQUEST(kw)

Fig. 42

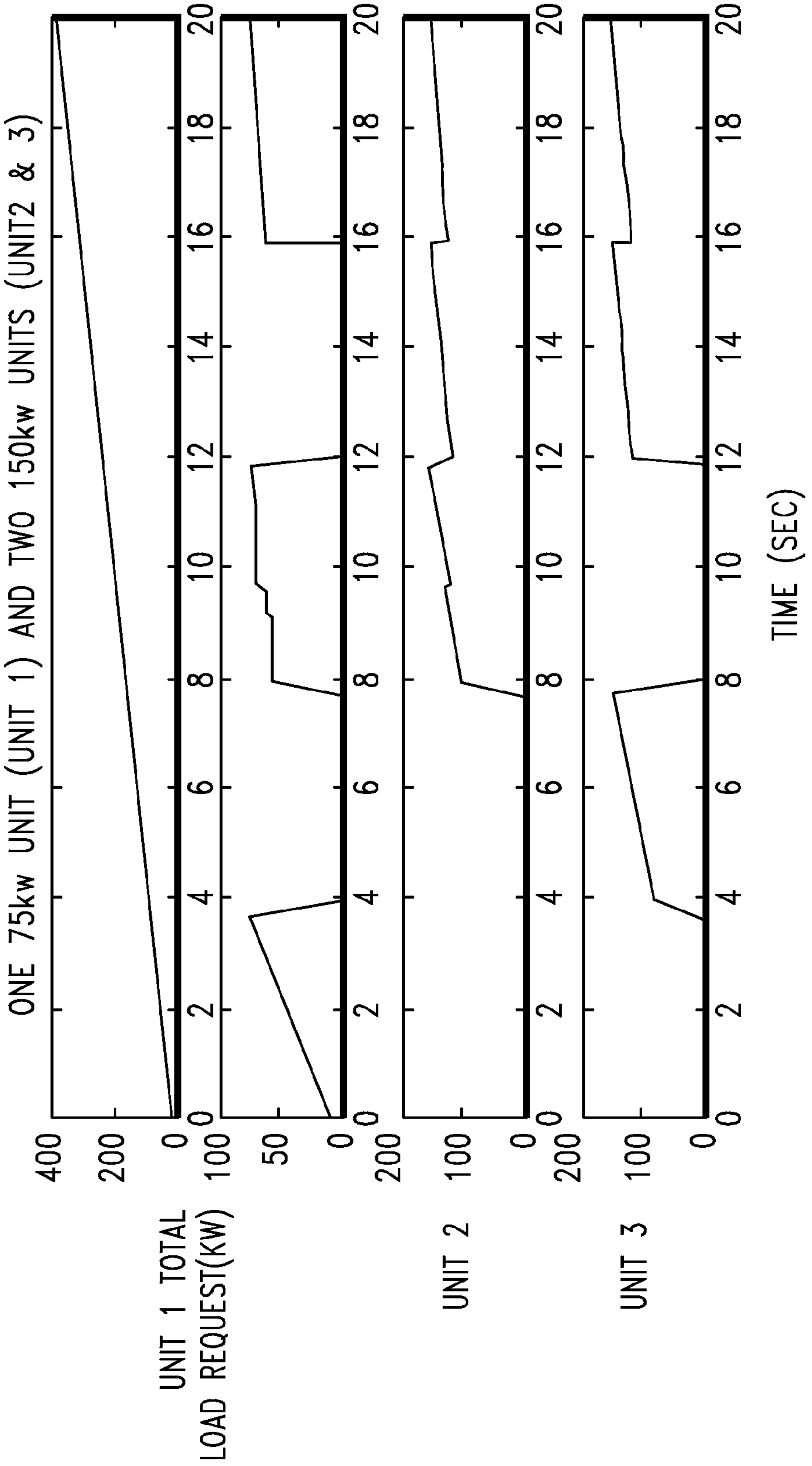

ONE 75kw UNIT (UNIT 1) AND TWO 150kw UNITS (UNIT2 & 3)
UNIT 1 TOTAL
LOAD REQUEST(KW)
400
200
0
100
50
0
UNIT 2
200
100
0
UNIT 3
200
100
0
0
2
4
6
8
10
12
14
16
18
20
TIME (SEC)

Fig. 43

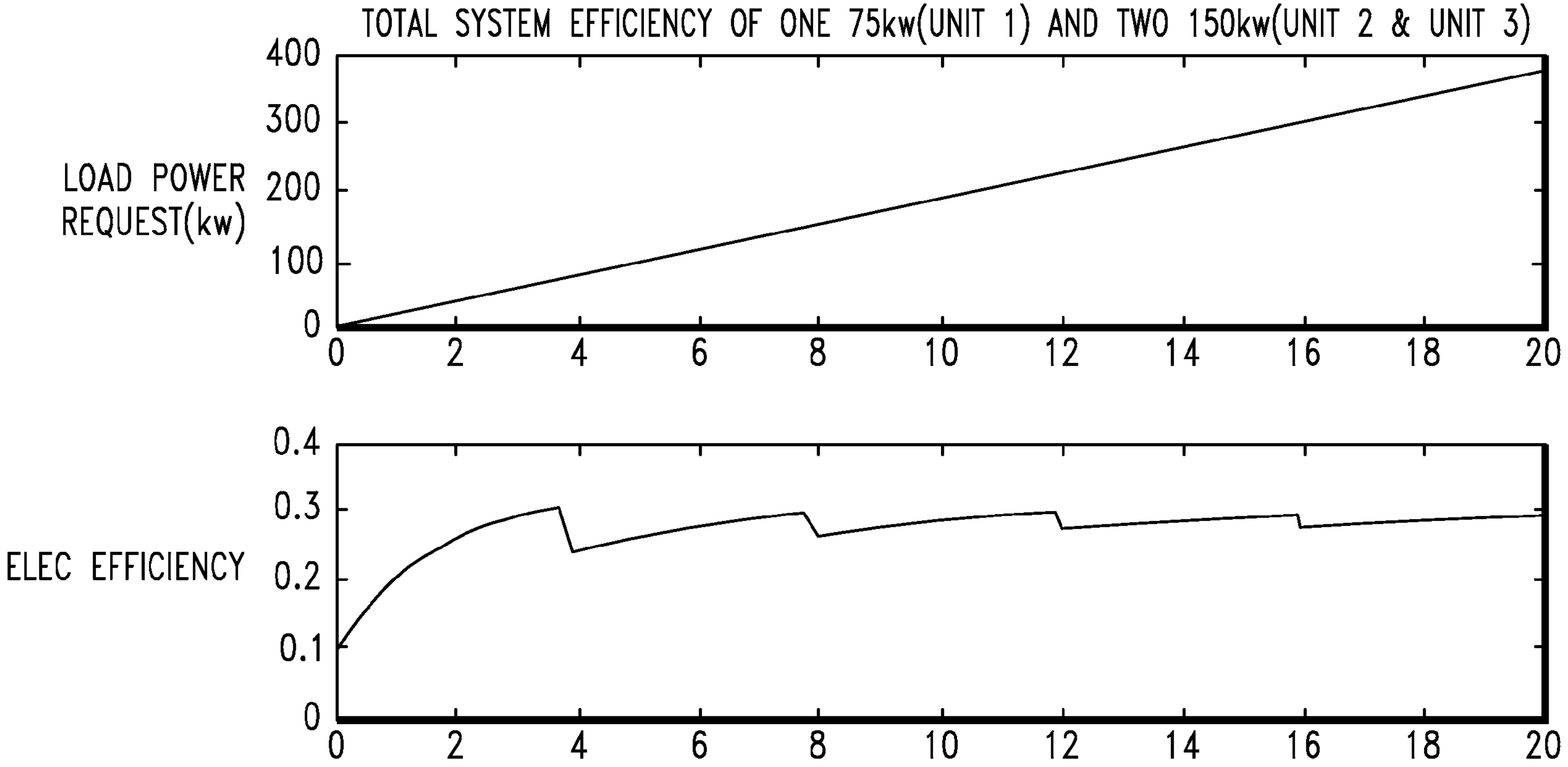

TOTAL SYSTEM EFFICIENCY OF ONE 75kw(UNIT 1) AND TWO 150kw(UNIT 2 & UNIT 3)
LOAD POWER REQUEST(kw)
400
300
200
100
0
0
2
4
6
8
10
12
14
16
18
20
ELEC EFFICIENCY
0.4
0.3
0.2
0.1
0
0
2
4
6
8
10
12
14
16
18
20

Fig. 44

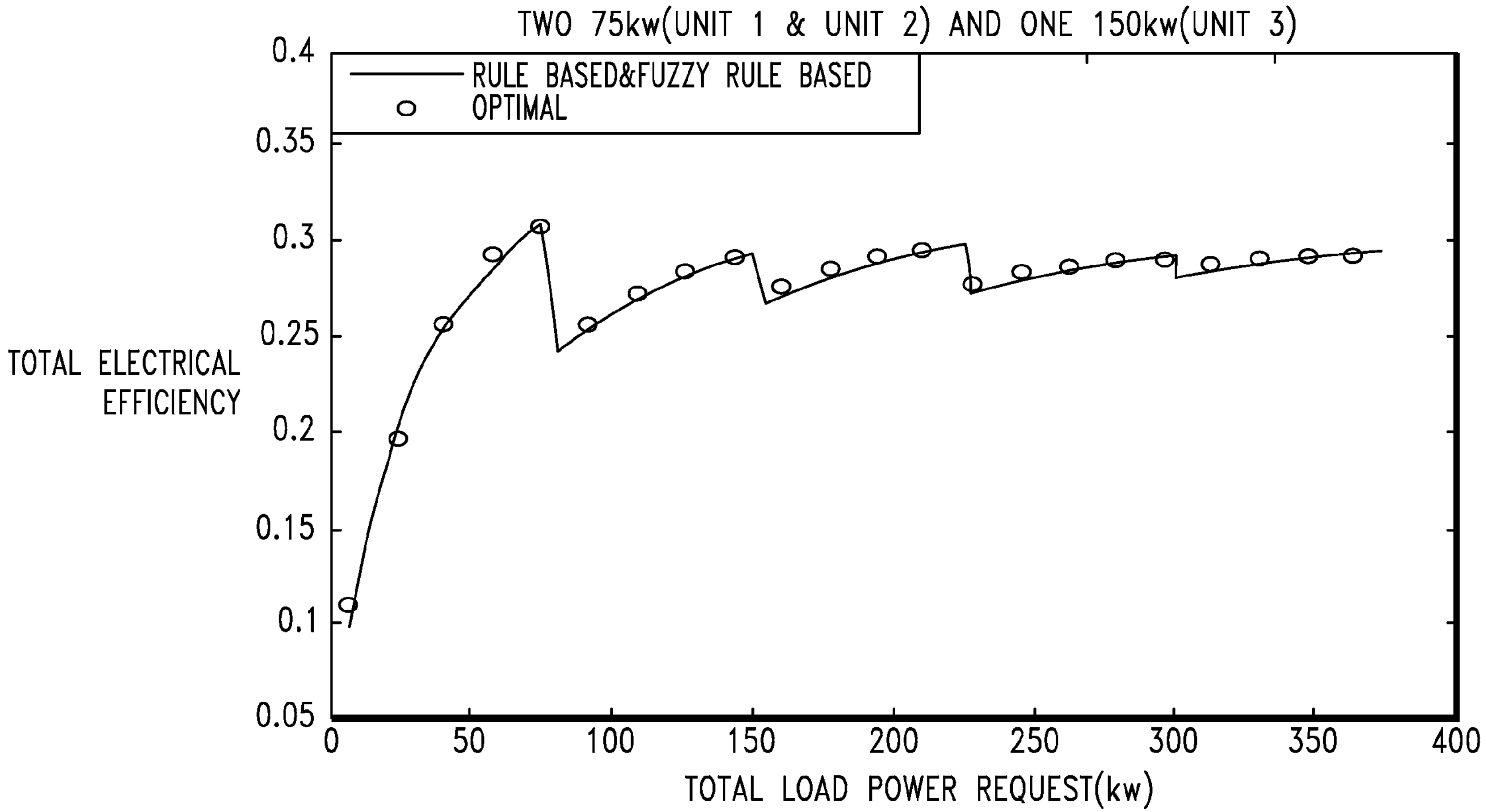

TWO 75kw(UNIT 1 & UNIT 2) AND ONE 150kw(UNIT 3)
RULE BASED&FUZZY RULE BASED
OPTIMAL
TOTAL ELECTRICAL EFFICIENCY
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
50
100
150
200
250
300
350
400
TOTAL LOAD POWER REQUEST(kw)

Fig. 45

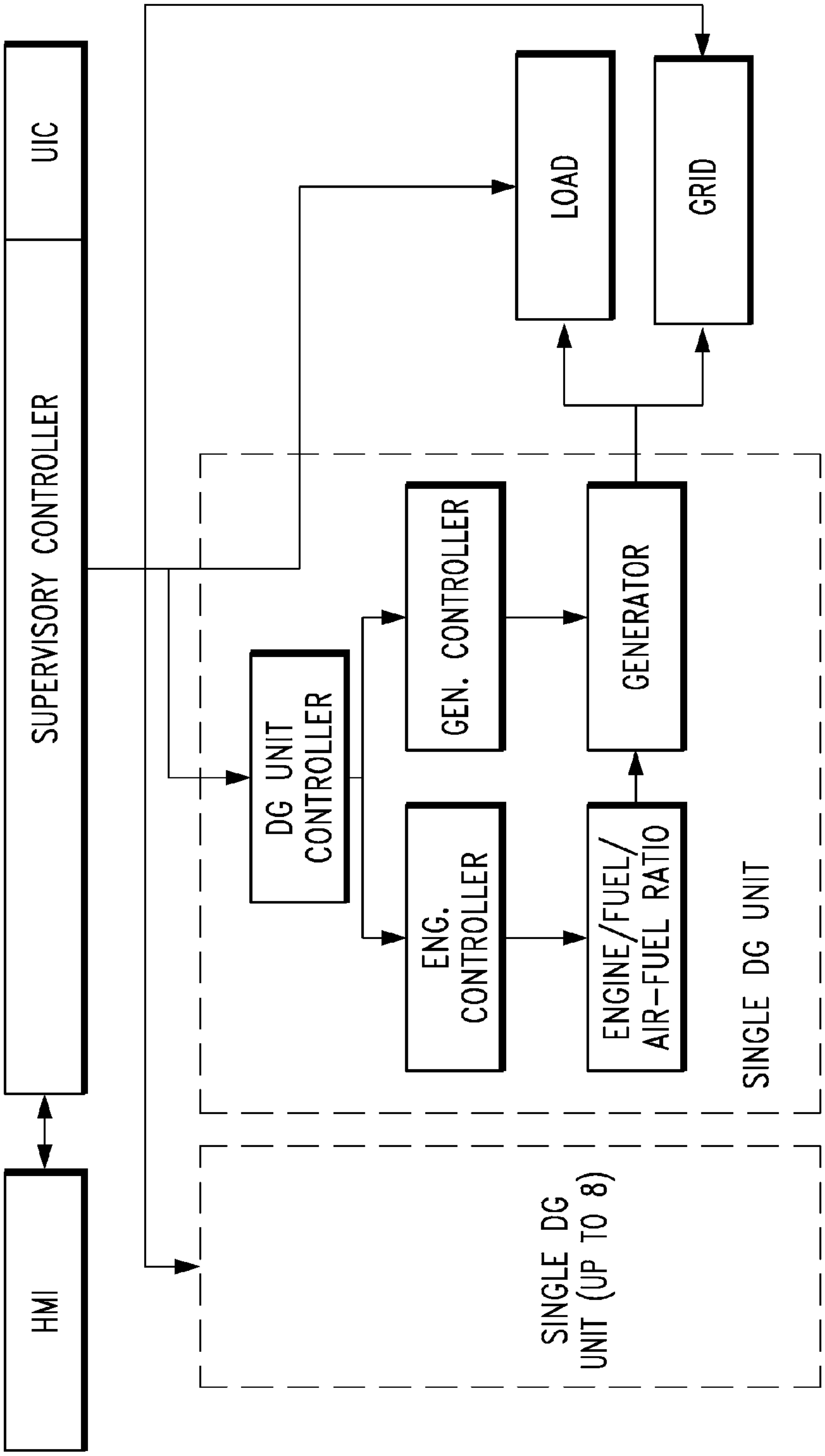

SUPERVISORY CONTROLLER
UIC
HMI
LOAD
GRID
DG UNIT CONTROLLER
GEN. CONTROLLER
ENG. CONTROLLER
GENERATOR
ENGINE/FUEL/ AIR-FUEL RATIO
SINGLE DG UNIT
SINGLE DG UNIT (UP TO 8)

Fig. 46

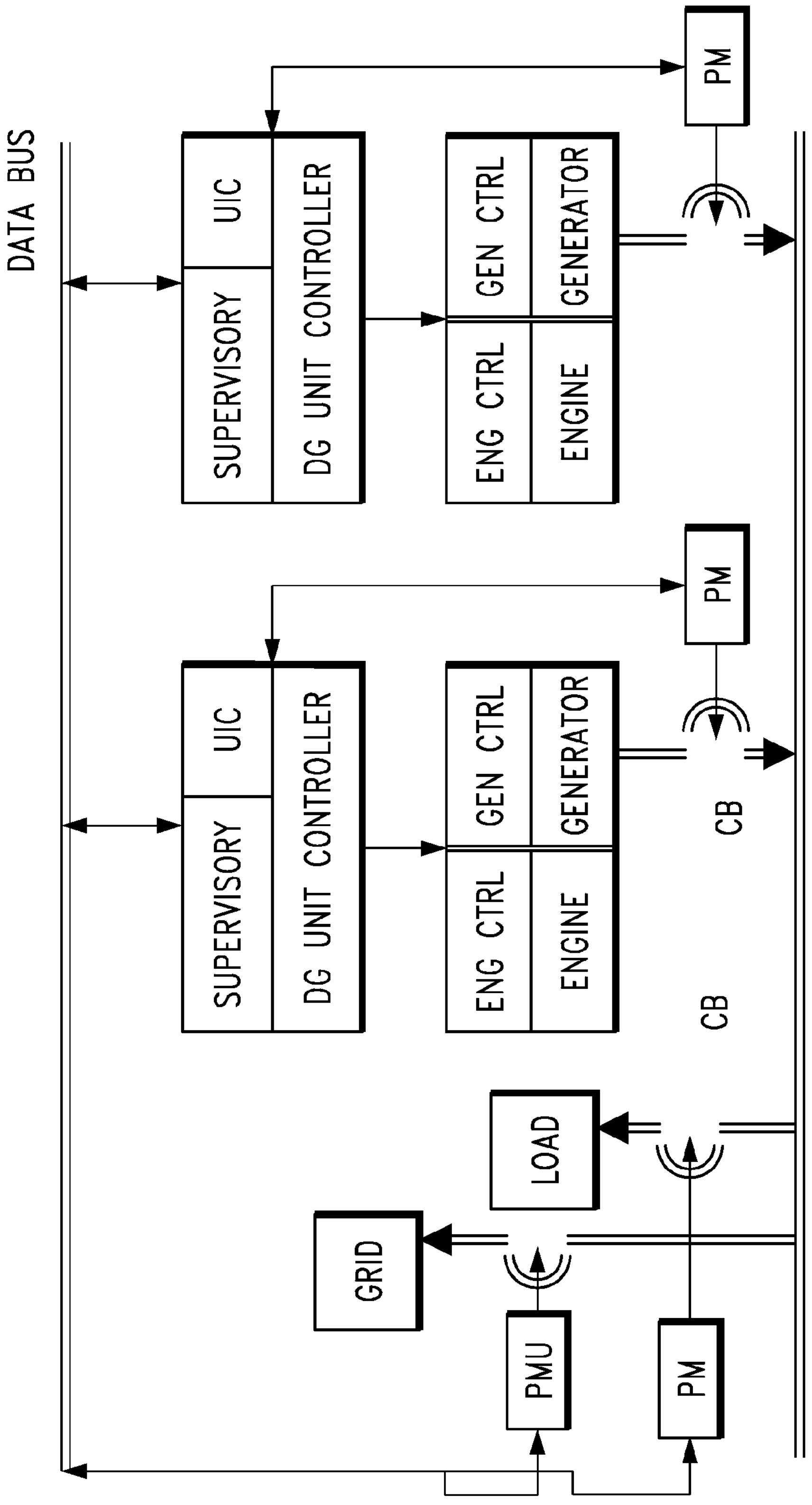

DATA BUS
SUPERVISORY
UIC
DG UNIT CONTROLLER
ENG CTRL
GEN CTRL
ENGINE
GENERATOR
PM
SUPERVISORY
UIC
DG UNIT CONTROLLER
ENG CTRL
GEN CTRL
ENGINE
GENERATOR
PM
CB
CB
LOAD
GRID
PMU
PM

Fig. 47

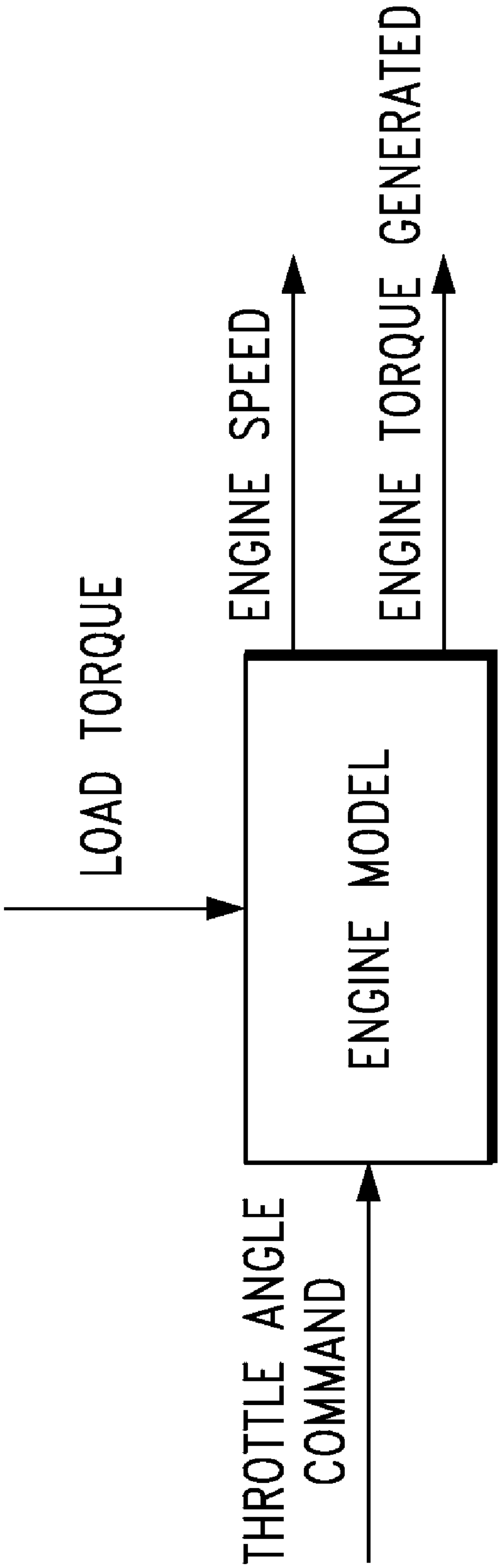

LOAD TORQUE
ENGINE SPEED
ENGINE TORQUE GENERATED
ENGINE MODEL
THROTTLE ANGLE COMMAND

Fig. 48

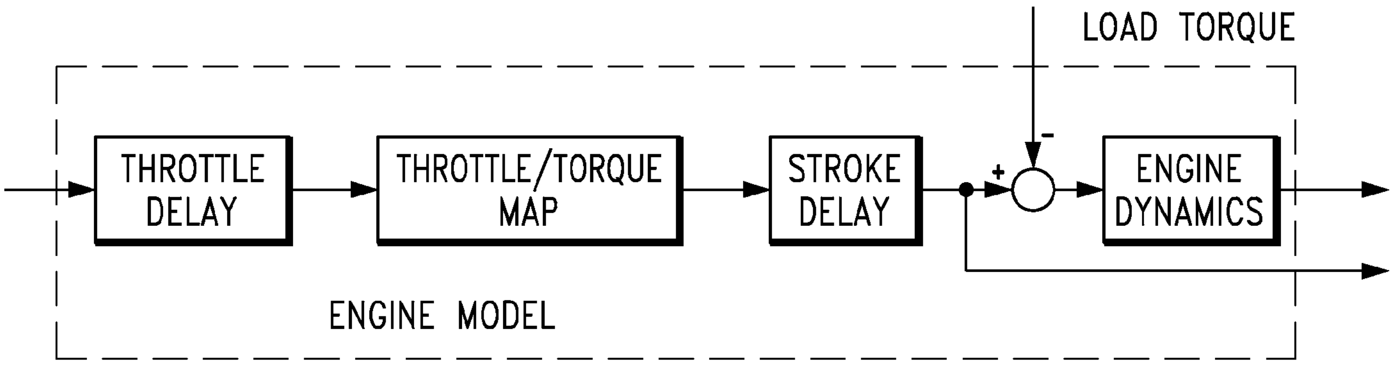

LOAD TORQUE
THROTTLE DELAY
THROTTLE/TORQUE MAP
STROKE DELAY
+
-
ENGINE DYNAMICS
ENGINE MODEL

Fig. 49

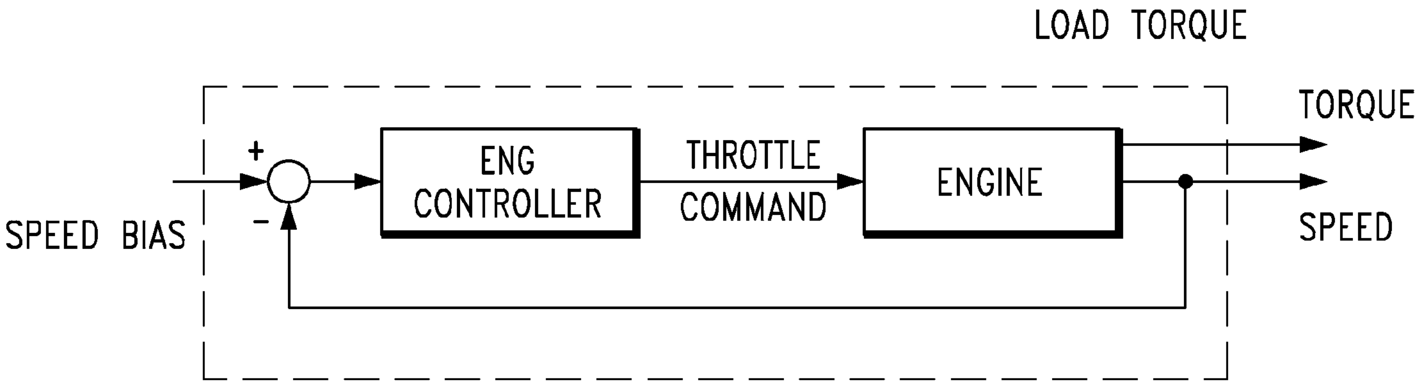

LOAD TORQUE
TORQUE
SPEED BIAS
+
-
ENG CONTROLLER
THROTTLE COMMAND
ENGINE
SPEED

Fig. 50

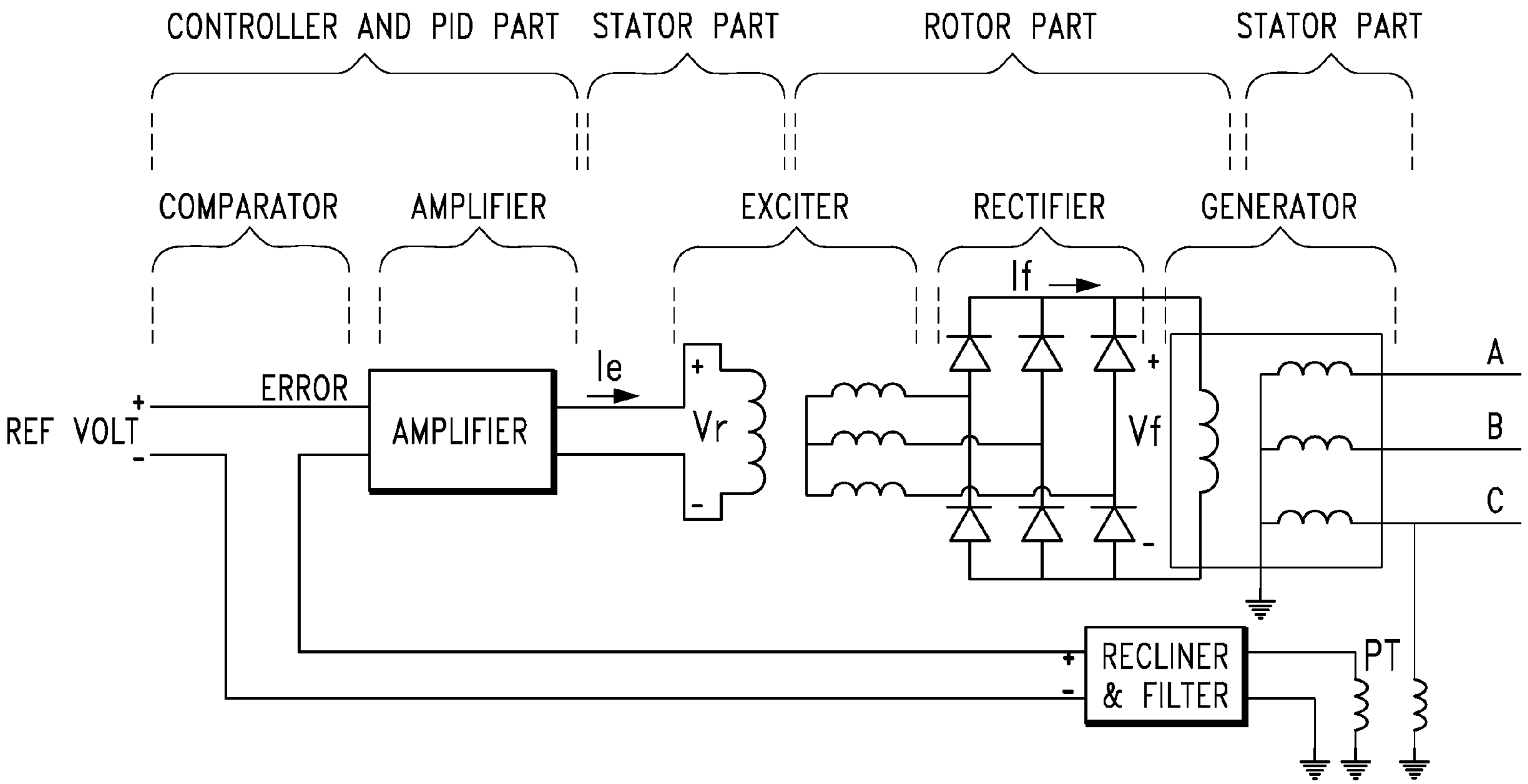


Fig. 51
(PRIOR ART)

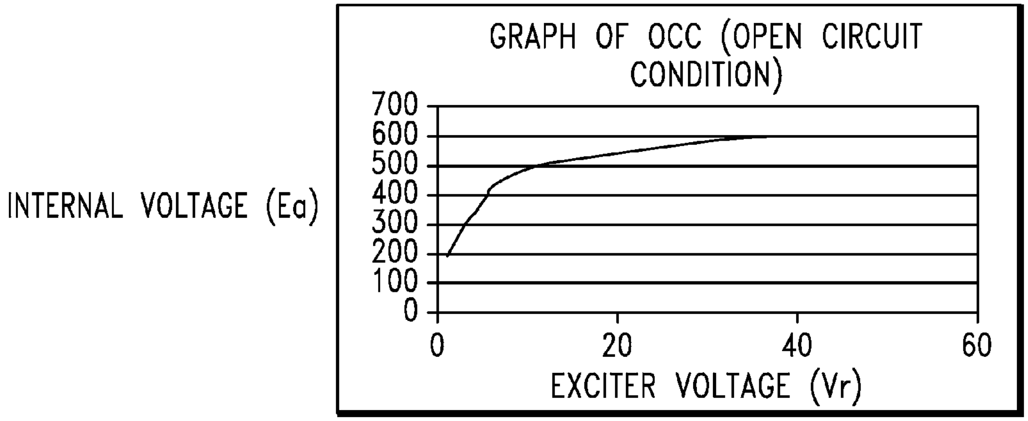

GRAPH OF OCC (OPEN CIRCUIT CONDITION)
INTERNAL VOLTAGE (Ea)
700
600
500
400
300
200
100
0
0
20
40
60
EXCITER VOLTAGE (Vr)

Fig. 52

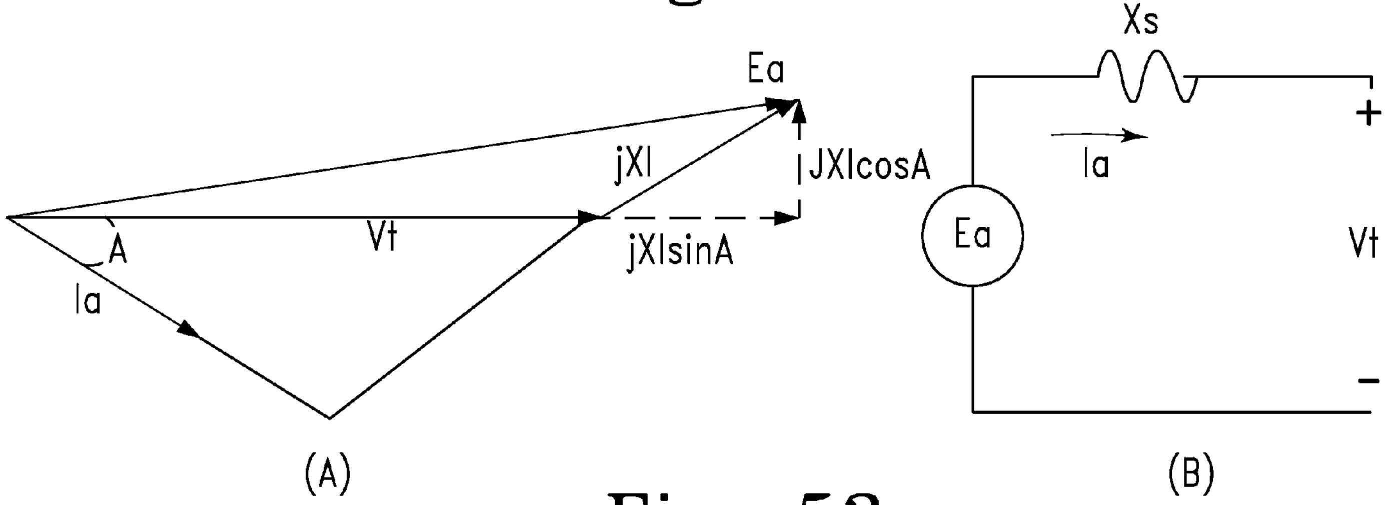

Ea
jXI
JXIcosA
A
Vt
jXIsinA
Ia
(A)
Xs
+
Ia
Ea
Vt
−
(B)

Fig. 53

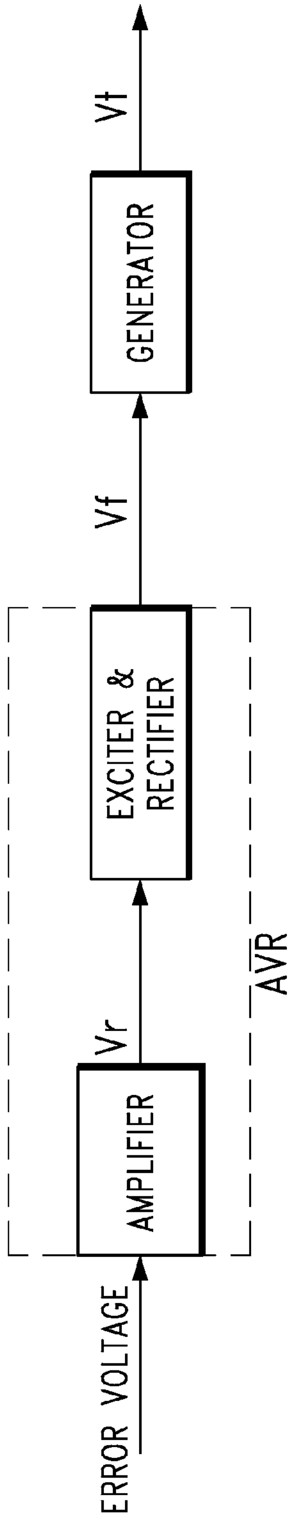

ERROR VOLTAGE
AMPLIFIER
Vr
EXCITER & RECTIFIER
AVR
Vf
GENERATOR
Vt

Fig. 54

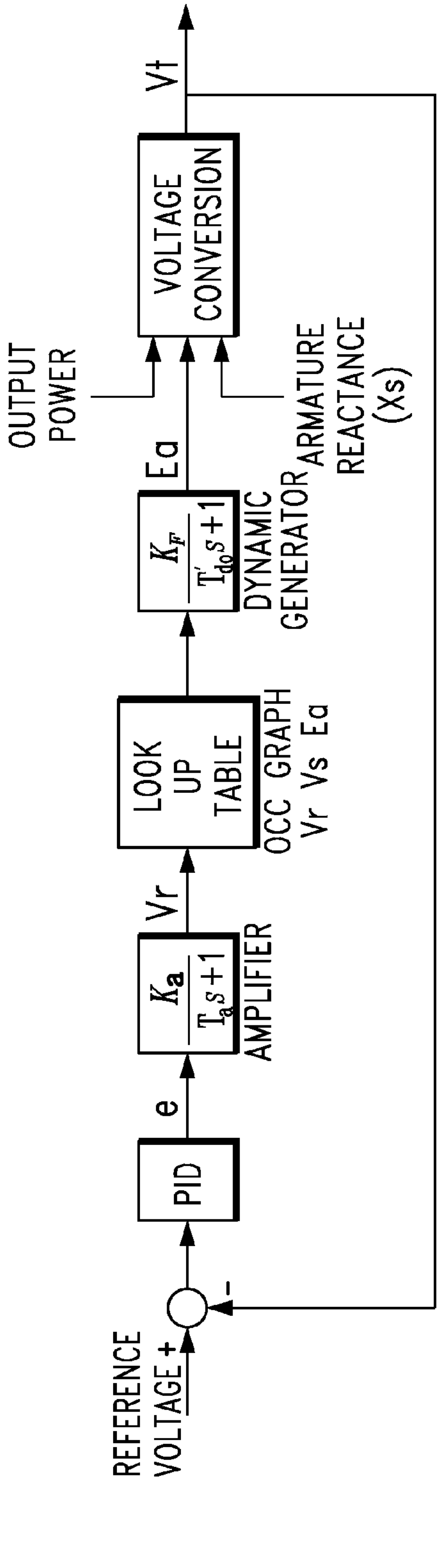

REFERENCE VOLTAGE
+
-
PID
e
$\frac{K_a}{T_a s+1}$
AMPLIFIER
Vr
LOOK UP TABLE
OCC GRAPH Vr Vs Ea
$\frac{K_F}{T'_{do} s+1}$
DYNAMIC GENERATOR
Ea
OUTPUT POWER
ARMATURE REACTANCE (Xs)
VOLTAGE CONVERSION
Vt

Fig. 55

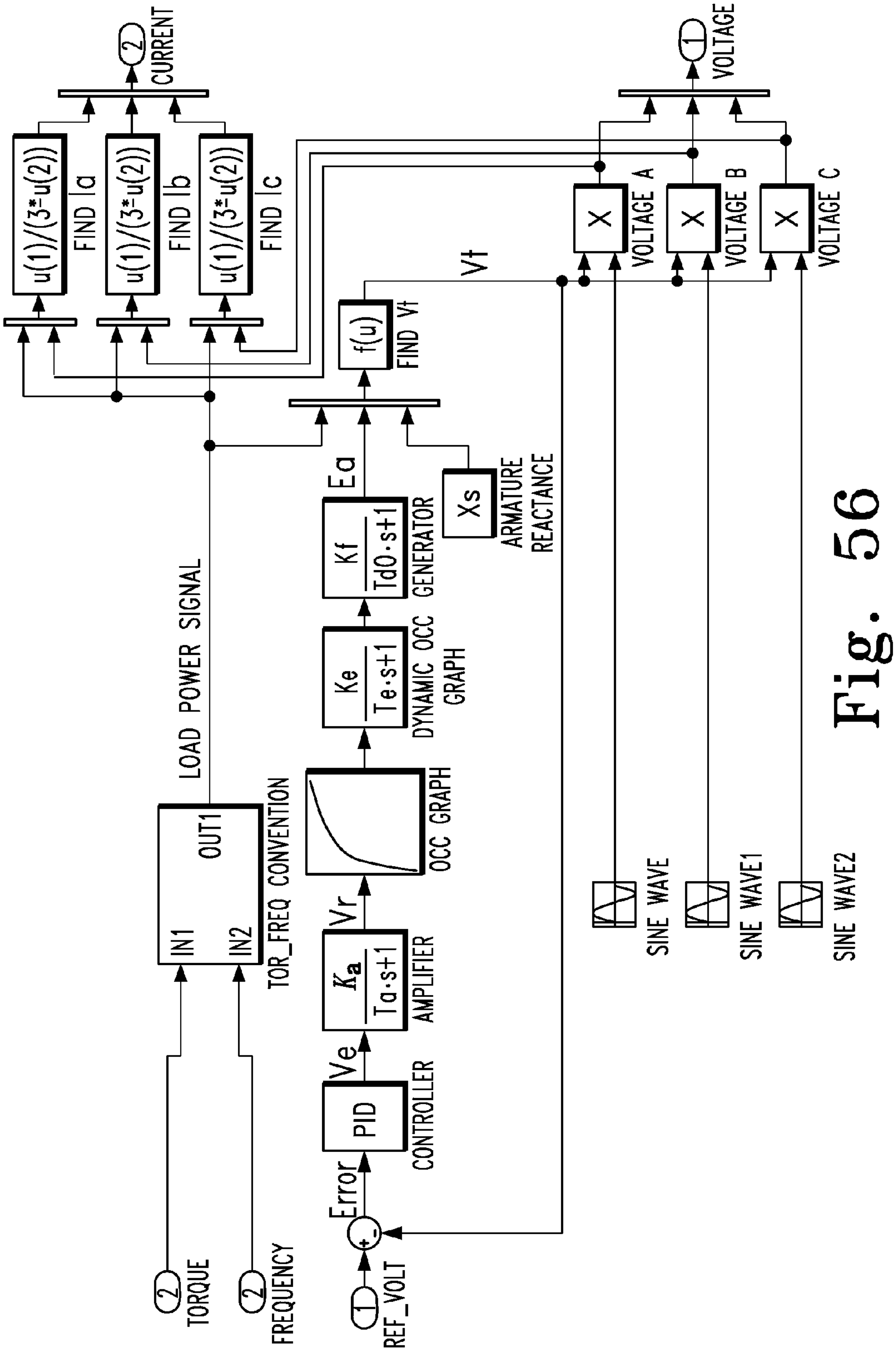

2
TORQUE
2
FREQUENCY
1
REF_VOLT
+
-
Error
PID
CONTROLLER
Ve
$\frac{K_a}{Ta \cdot s+1}$
AMPLIFIER
Vr
OCC GRAPH
$\frac{Ke}{Te \cdot s+1}$
DYNAMIC OCC GRAPH
$\frac{Kf}{Td0 \cdot s+1}$
GENERATOR
Ea
Xs
ARMATURE REACTANCE
IN1
IN2
OUT1
TOR_FREQ CONVENTION
LOAD POWER SIGNAL
f(u)
FIND Vt
Vt
u(1)/(3*u(2))
FIND Ia
u(1)/(3*u(2))
FIND Ib
u(1)/(3*u(2))
FIND Ic
2
CURRENT
X
VOLTAGE A
X
VOLTAGE B
X
VOLTAGE C
1
VOLTAGE
SINE WAVE
SINE WAVE1
SINE WAVE2

Fig. 56

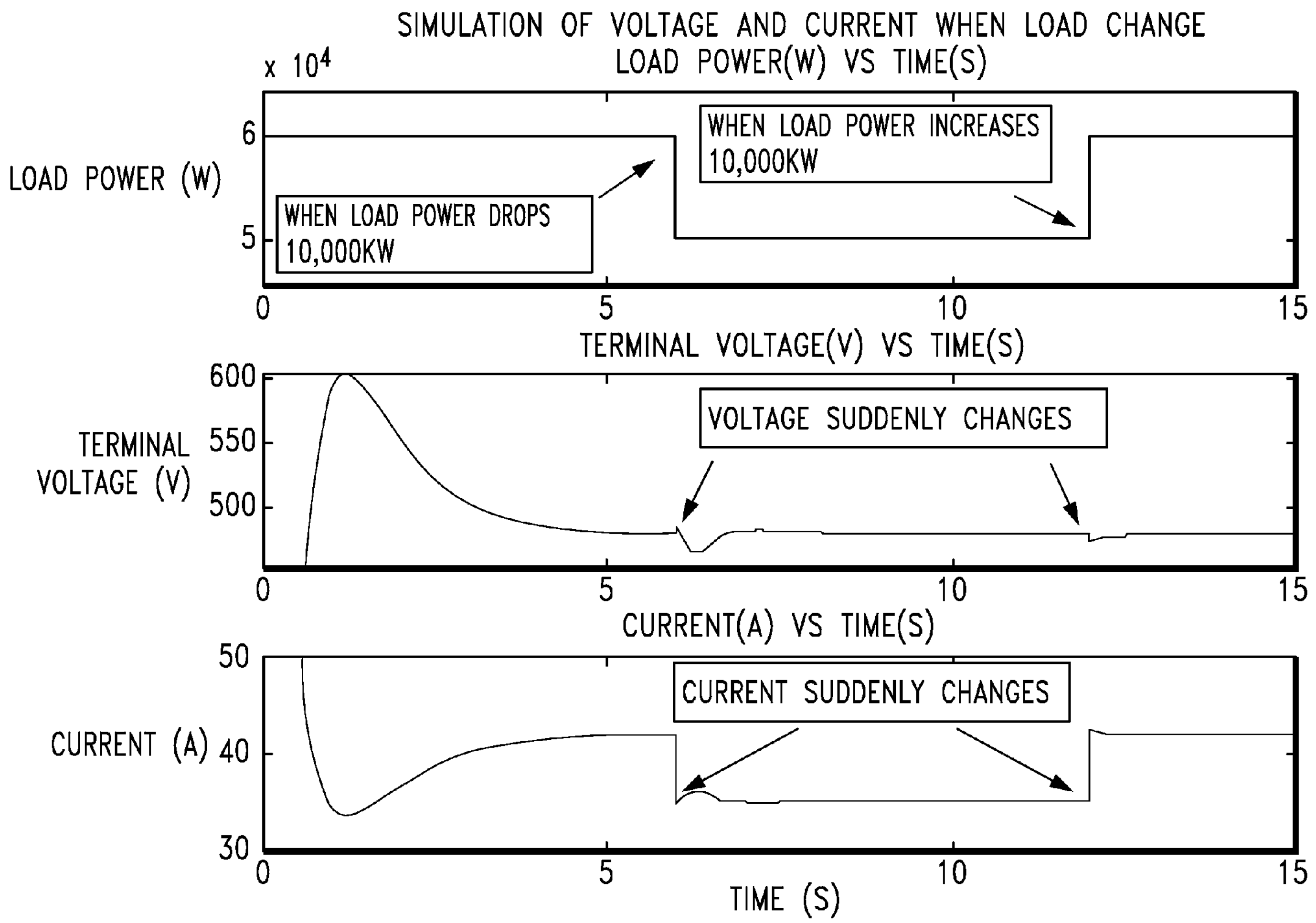

SIMULATION OF VOLTAGE AND CURRENT WHEN LOAD CHANGE
LOAD POWER(W) VS TIME(S)
x 10^4
LOAD POWER (W)
6
5
WHEN LOAD POWER DROPS 10,000KW
WHEN LOAD POWER INCREASES 10,000KW
0
5
10
15
TERMINAL VOLTAGE(V) VS TIME(S)
TERMINAL VOLTAGE (V)
600
550
500
VOLTAGE SUDDENLY CHANGES
0
5
10
15
CURRENT(A) VS TIME(S)
CURRENT (A)
50
40
30
CURRENT SUDDENLY CHANGES
0
5
10
15
TIME (S)

Fig. 57

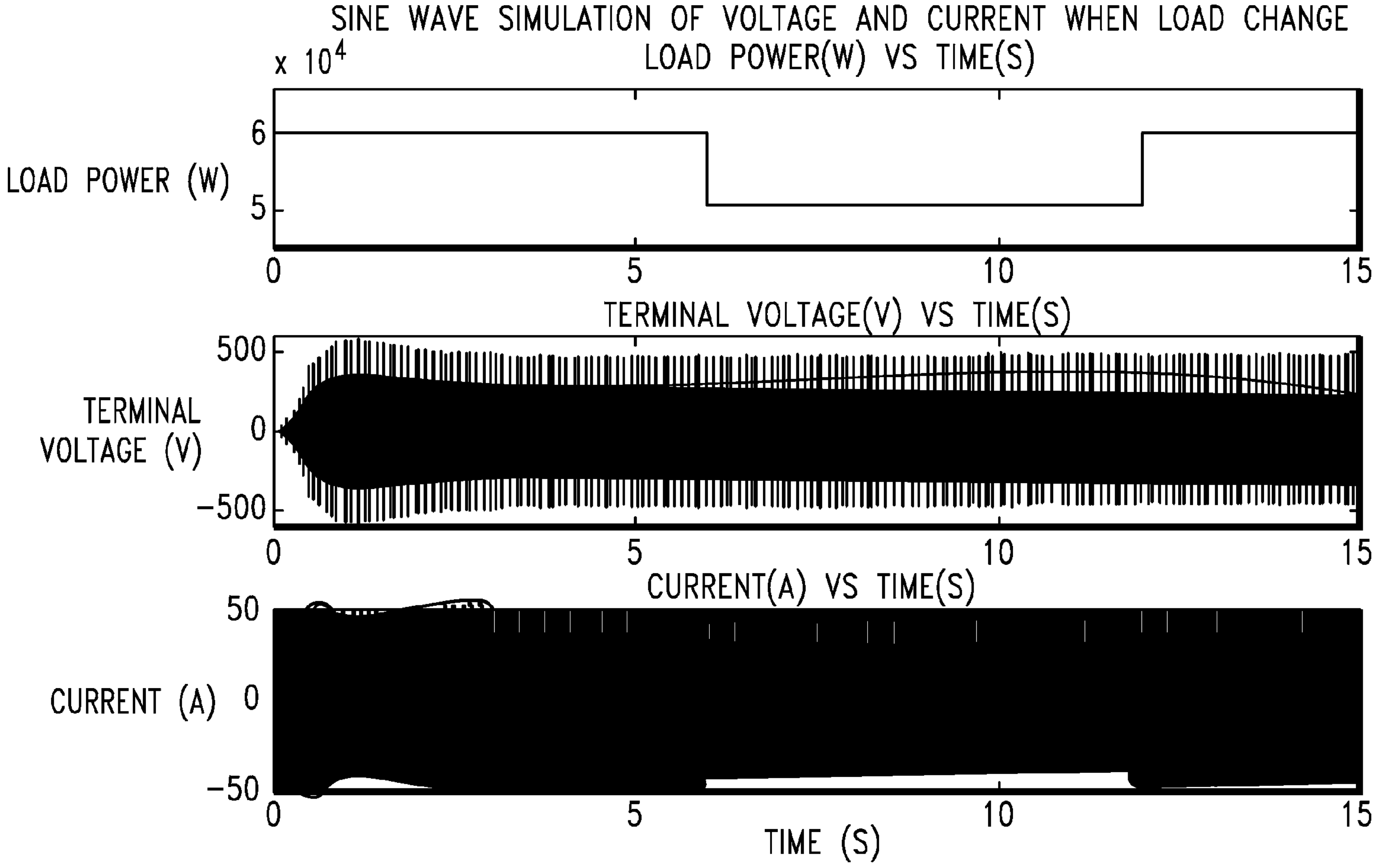

SINE WAVE SIMULATION OF VOLTAGE AND CURRENT WHEN LOAD CHANGE
LOAD POWER(W) VS TIME(S)
x 10^4
LOAD POWER (W)
6
5
0
5
10
15
TERMINAL VOLTAGE(V) VS TIME(S)
TERMINAL VOLTAGE (V)
500
0
−500
0
5
10
15
CURRENT(A) VS TIME(S)
CURRENT (A)
50
0
−50
0
5
10
15
TIME (S)

Fig. 58

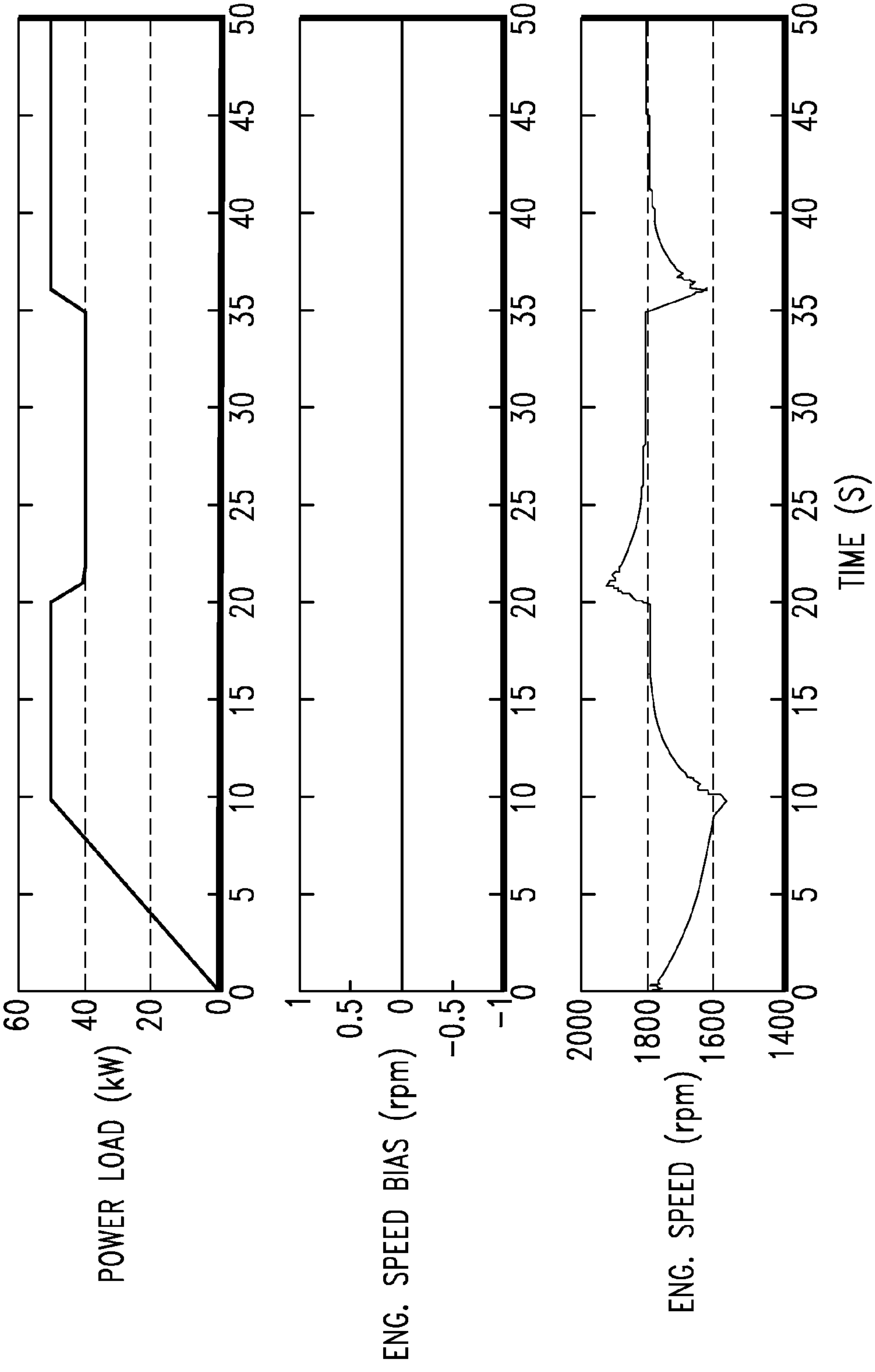

POWER LOAD (kW)
60
40
20
0
ENG. SPEED BIAS (rpm)
1
0.5
0
−0.5
−1
ENG. SPEED (rpm)
2000
1800
1600
1400
0
5
10
15
20
25
30
35
40
45
50
TIME (S)

Fig. 59

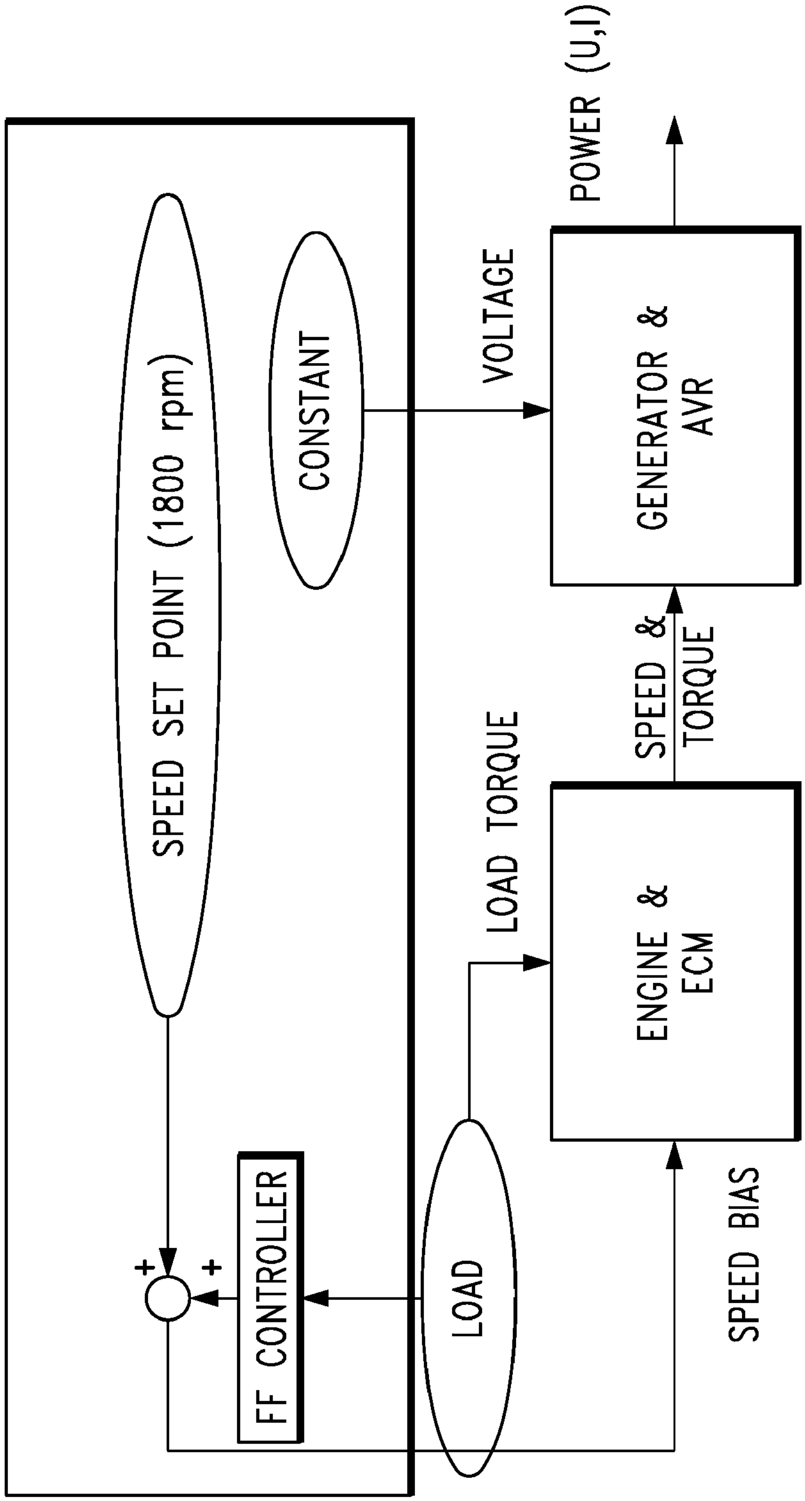

SPEED SET POINT (1800 rpm)
CONSTANT
VOLTAGE
POWER (U,I)
GENERATOR & AVR
SPEED & TORQUE
LOAD TORQUE
ENGINE & ECM
+
+
FF CONTROLLER
LOAD
SPEED BIAS

Fig. 60

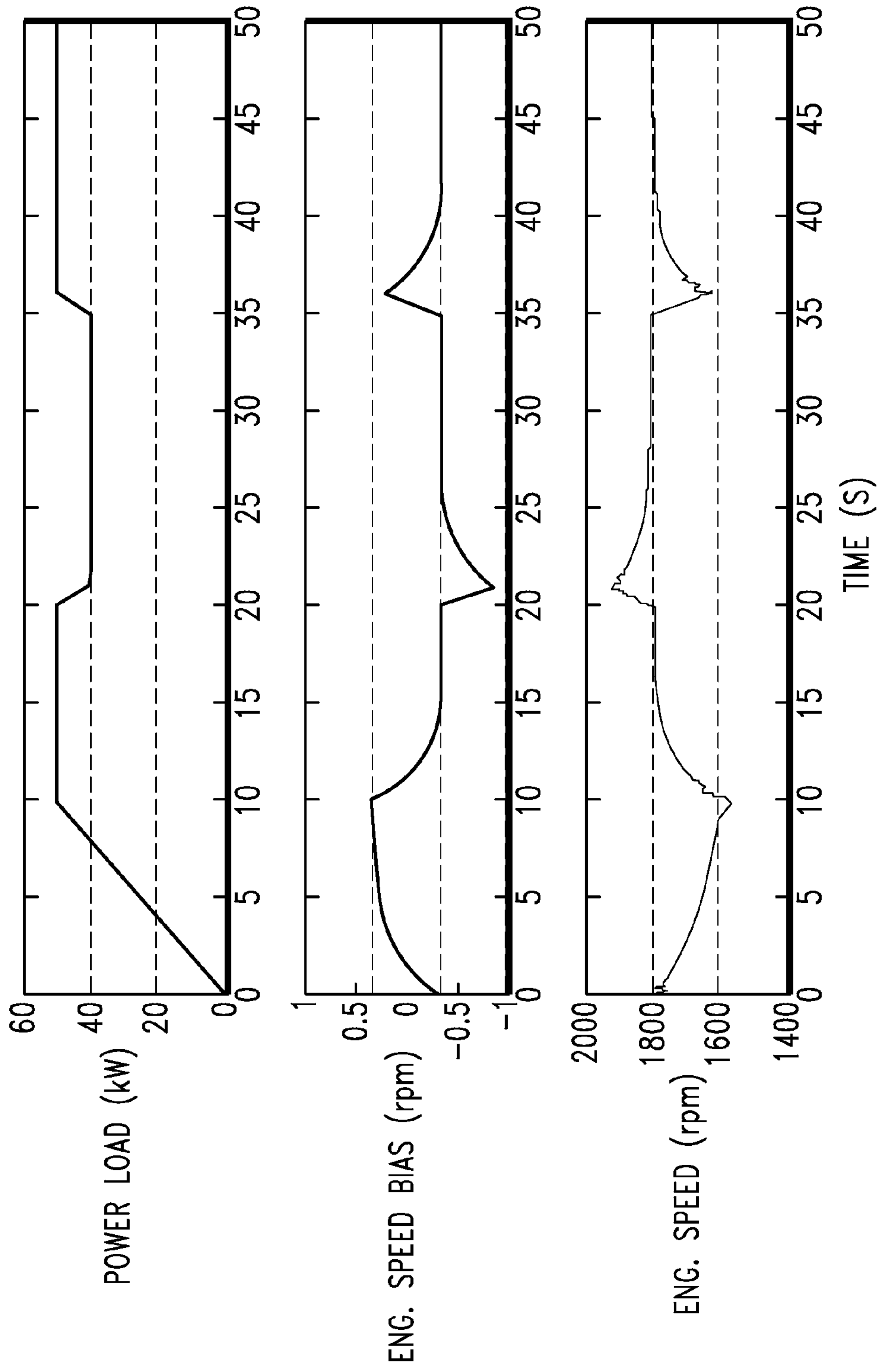

POWER LOAD (kW)
60
40
20
0
ENG. SPEED BIAS (rpm)
1
0.5
0
−0.5
−1
ENG. SPEED (rpm)
2000
1800
1600
1400
0
5
10
15
20
25
30
35
40
45
50
TIME (S)

Fig. 61

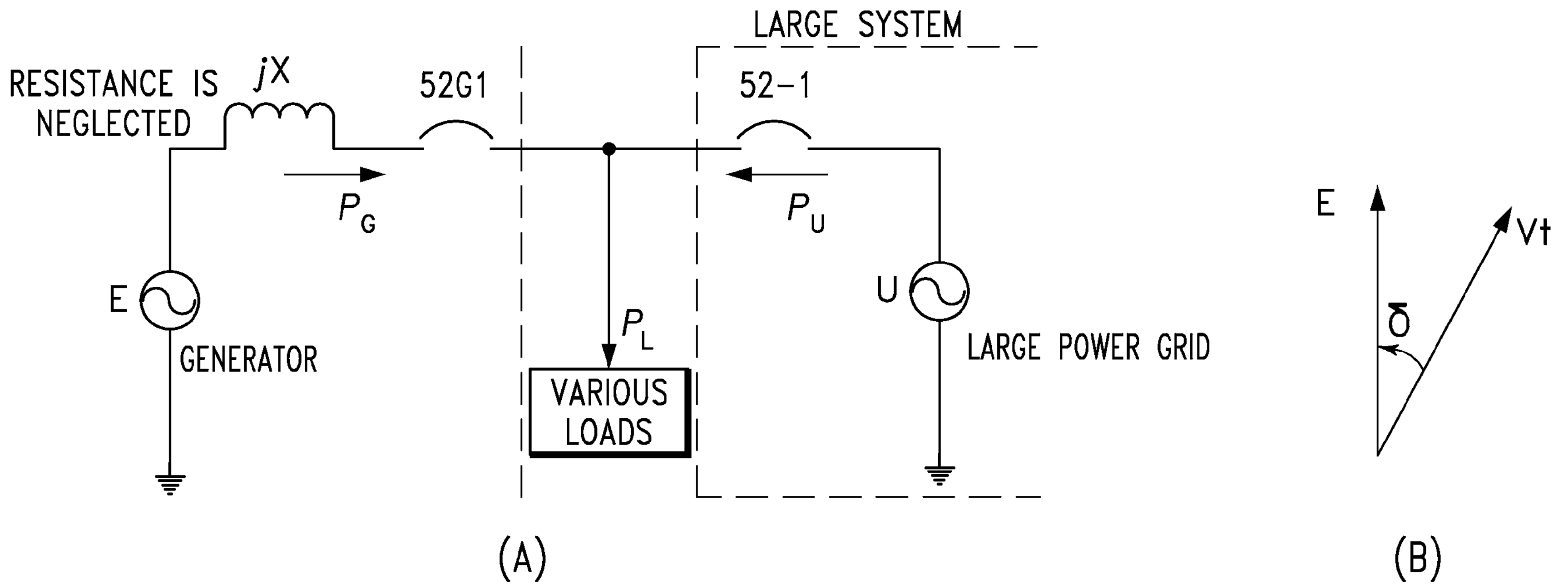

RESISTANCE IS NEGLECTED
jX
52G1
LARGE SYSTEM
52-1
P_G
P_U
E
GENERATOR
P_L
VARIOUS LOADS
U
LARGE POWER GRID
(A)
E
Vt
δ
(B)

Fig. 62

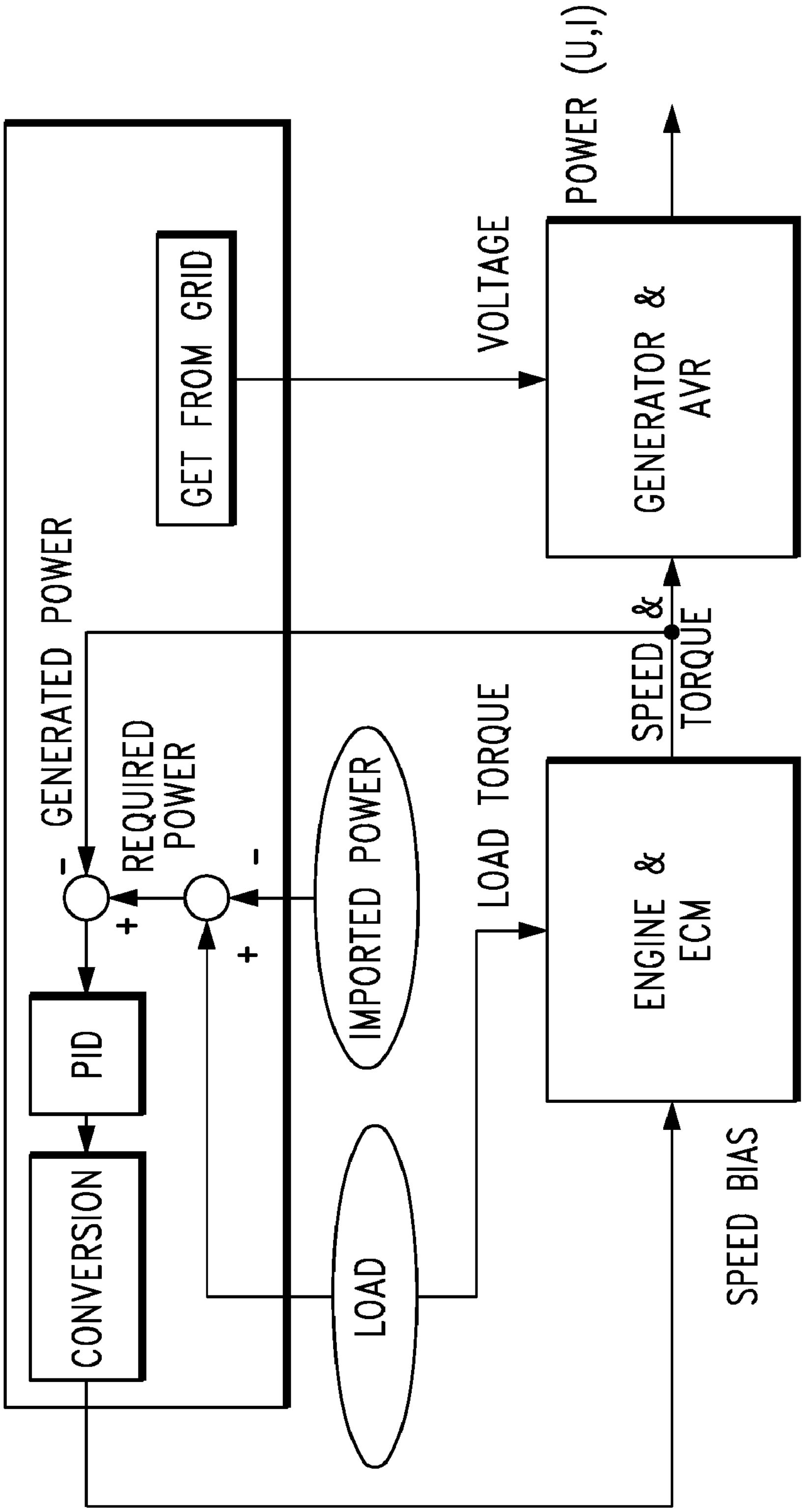

GENERATED POWER
-
+
REQUIRED POWER
-
+
IMPORTED POWER
PID
CONVERSION
LOAD
GET FROM GRID
VOLTAGE
POWER (U,I)
GENERATOR & AVR
SPEED & TORQUE
LOAD TORQUE
ENGINE & ECM
SPEED BIAS

Fig. 63

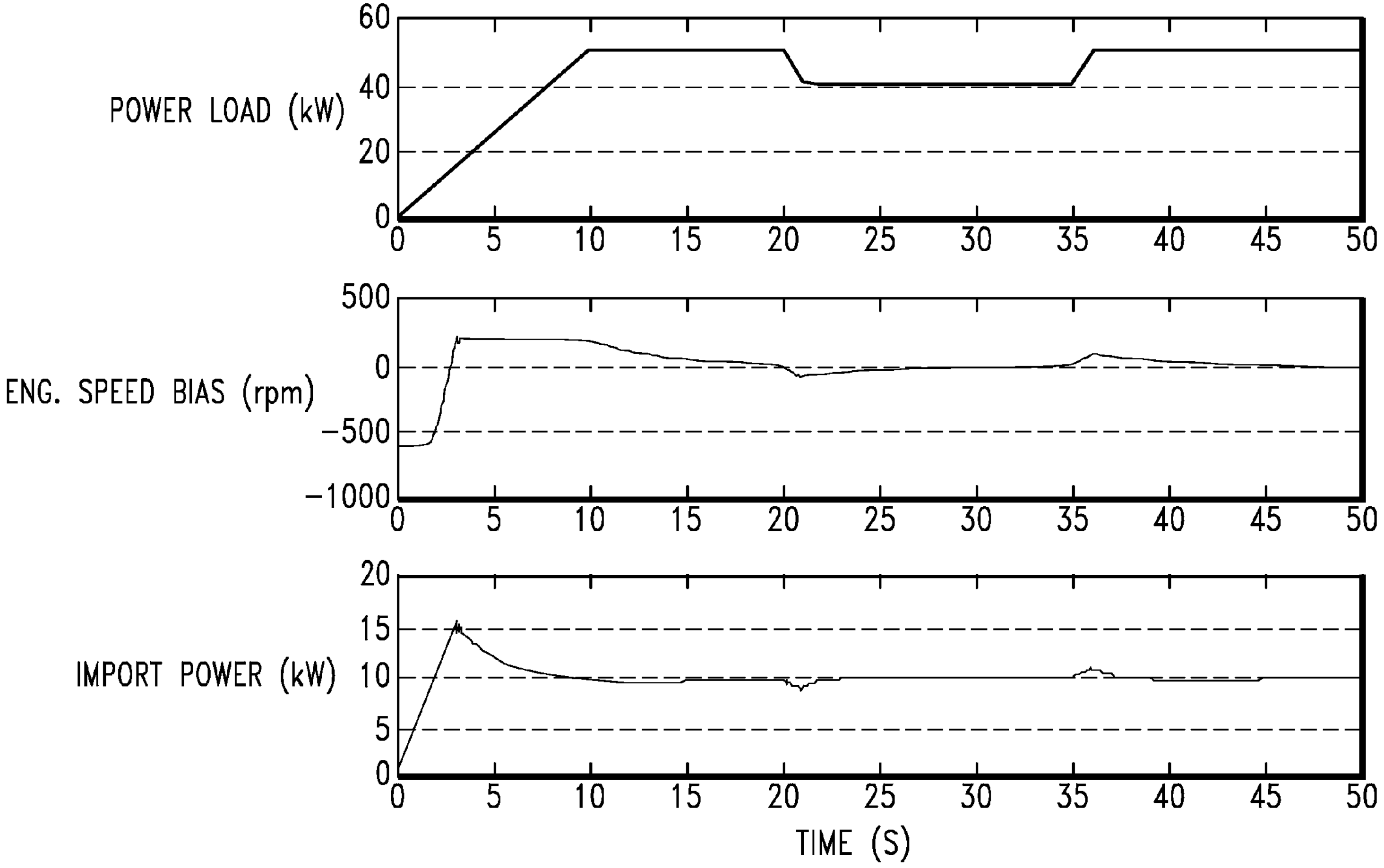

POWER LOAD (kW)
60
40
20
0
0
5
10
15
20
25
30
35
40
45
50
ENG. SPEED BIAS (rpm)
500
0
−500
−1000
0
5
10
15
20
25
30
35
40
45
50
IMPORT POWER (kW)
20
15
10
5
0
0
5
10
15
20
25
30
35
40
45
50
TIME (S)

Fig. 64

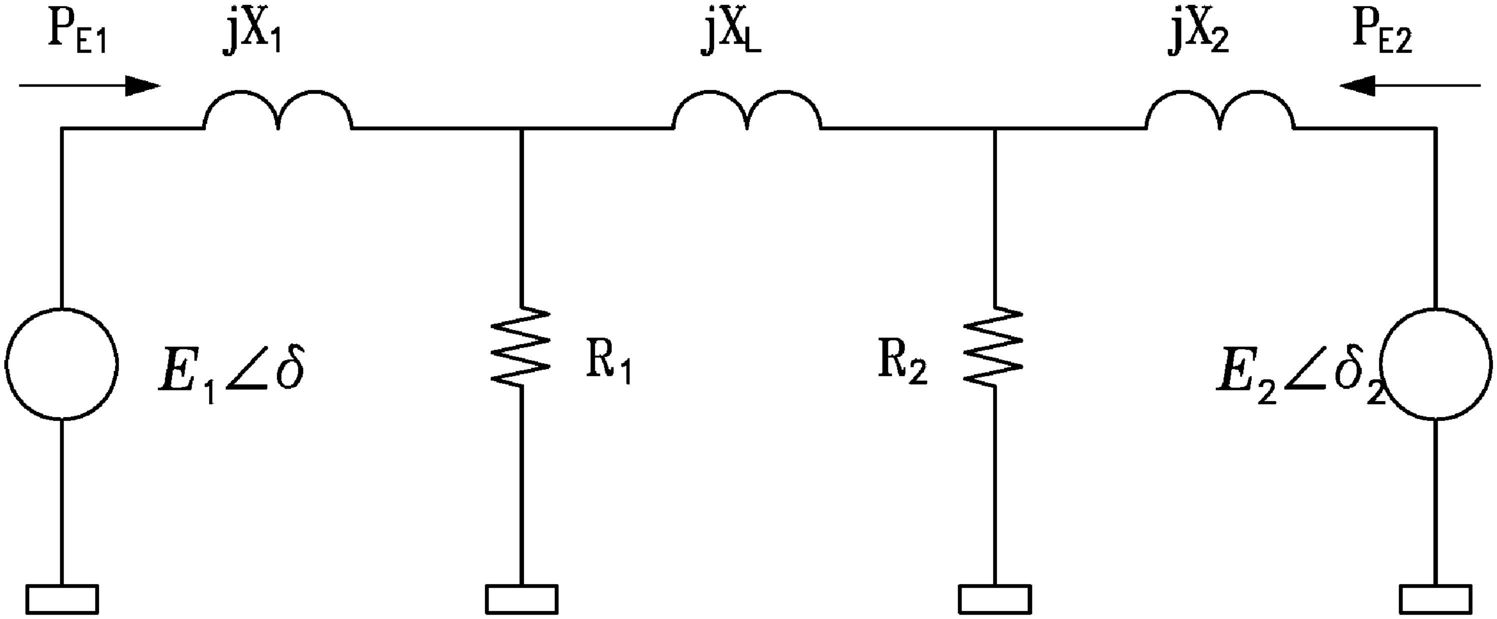


Fig. 65
(PRIOR ART)

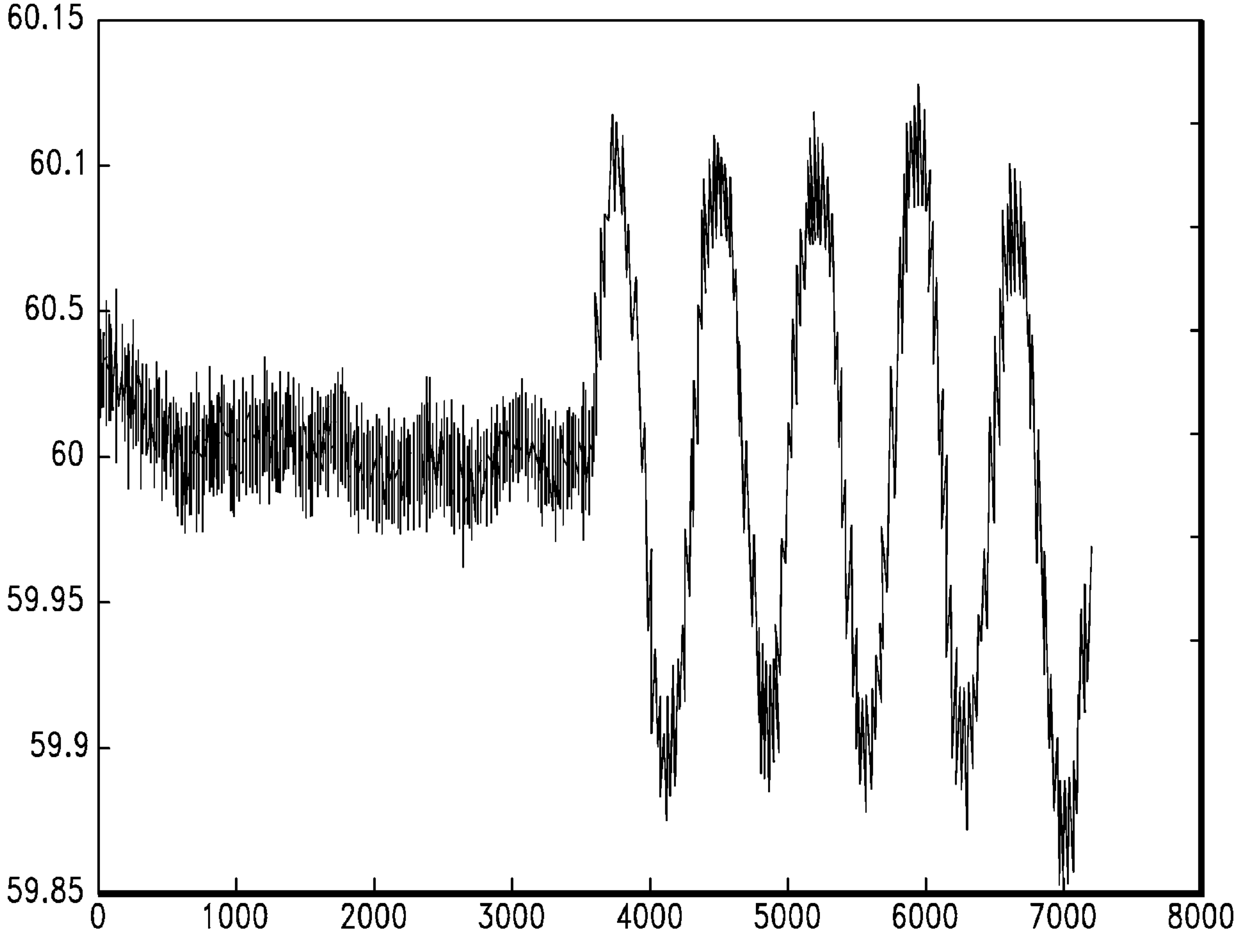

60.15
60.1
60.5
60
59.95
59.9
59.85
0
1000
2000
3000
4000
5000
6000
7000
8000

Fig. 66

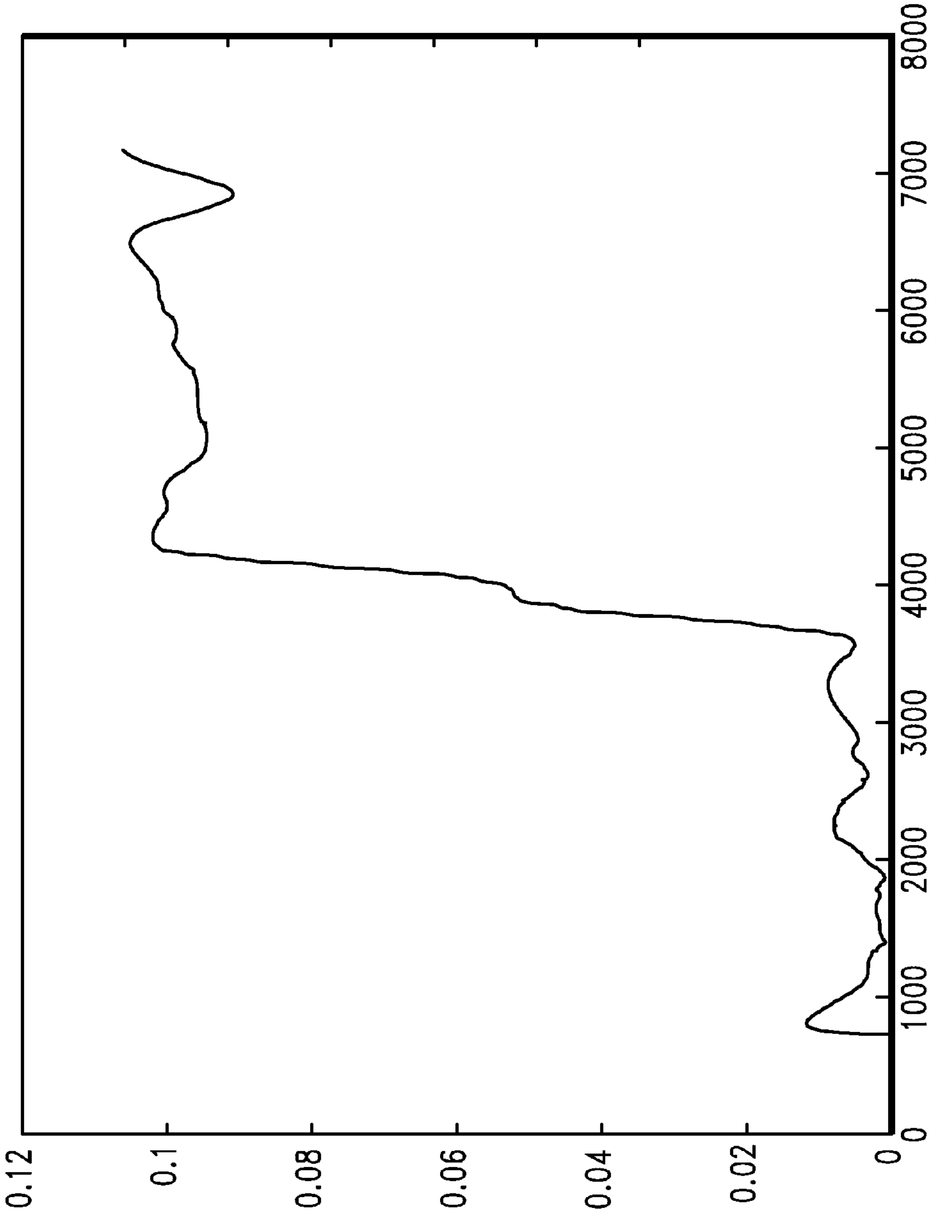

0.12
0.1
0.08
0.06
0.04
0.02
0
0
1000
2000
3000
4000
5000
6000
7000
8000

Fig. 67

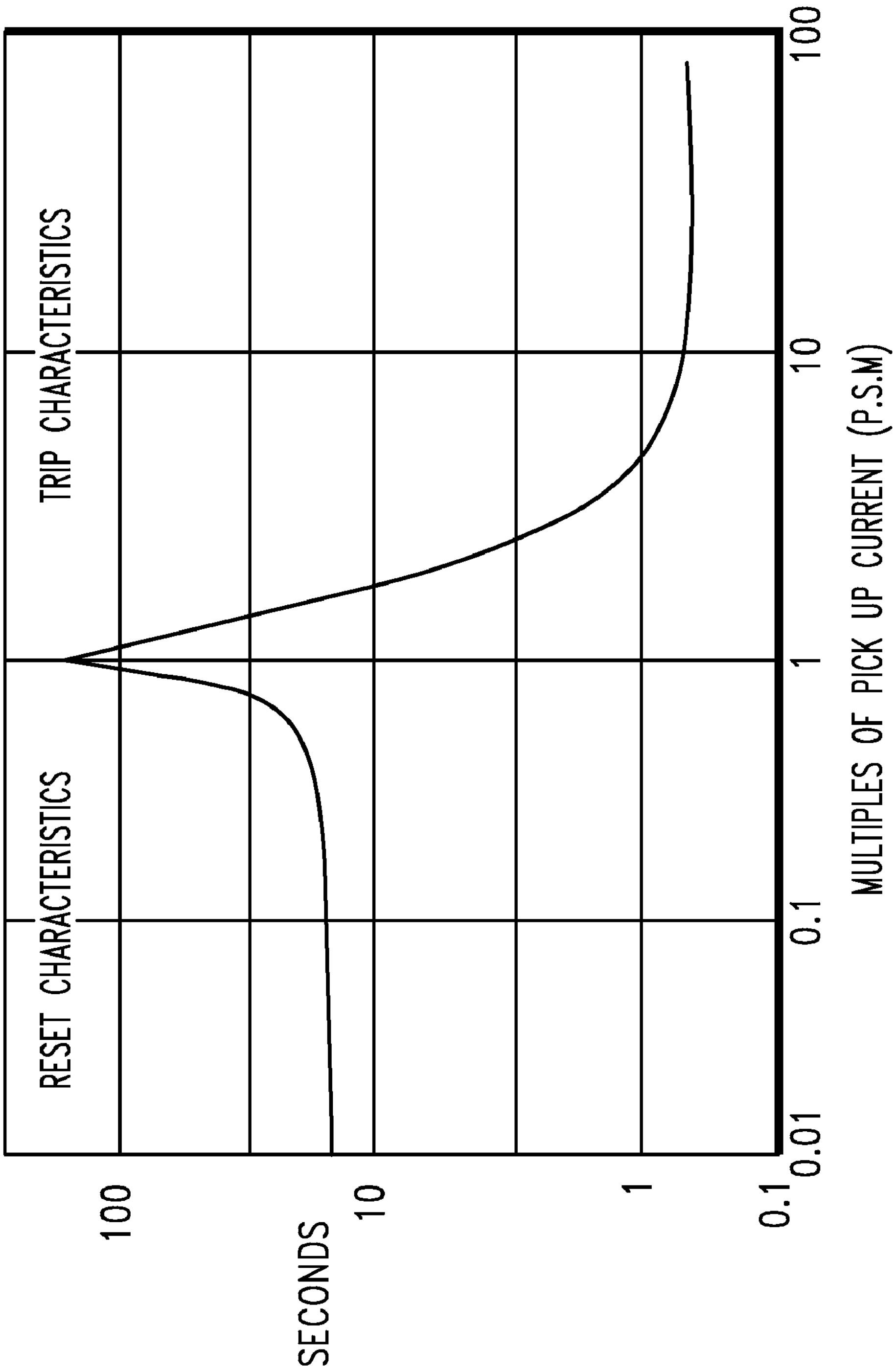

RESET CHARACTERISTICS
TRIP CHARACTERISTICS
SECONDS
100
10
1
0.1
0.01
0.1
1
10
100
MULTIPLES OF PICK UP CURRENT (P.S.M)

Fig. 68

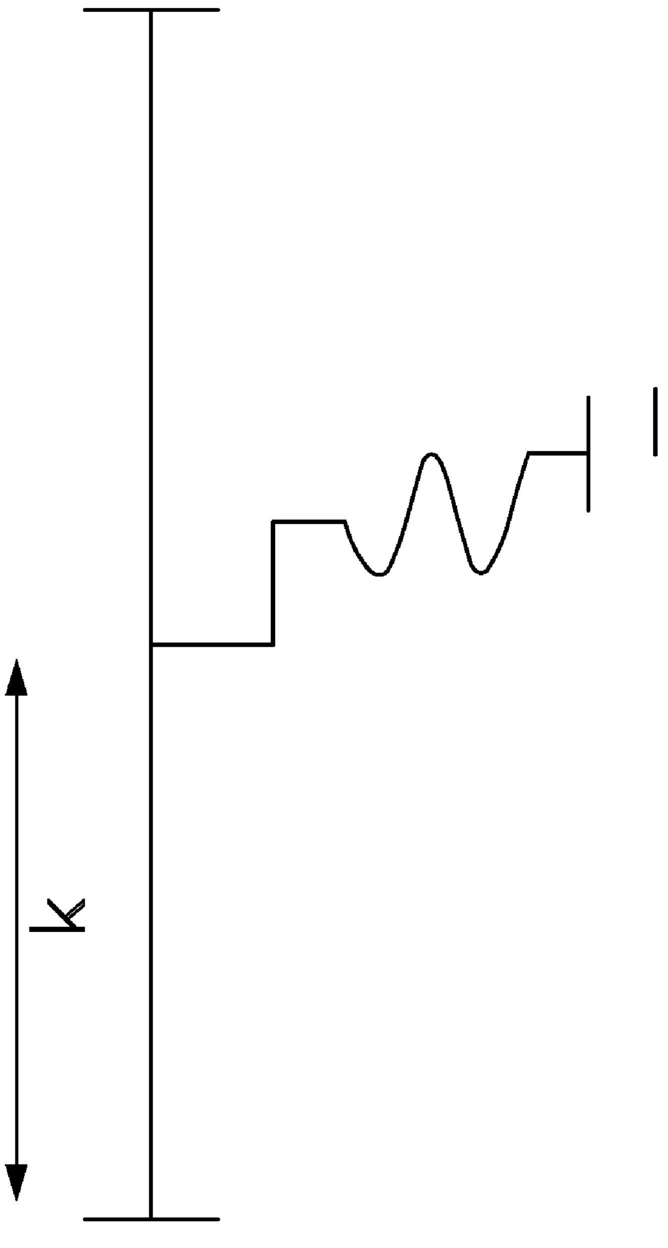


Fig. 69
(PRIOR ART)

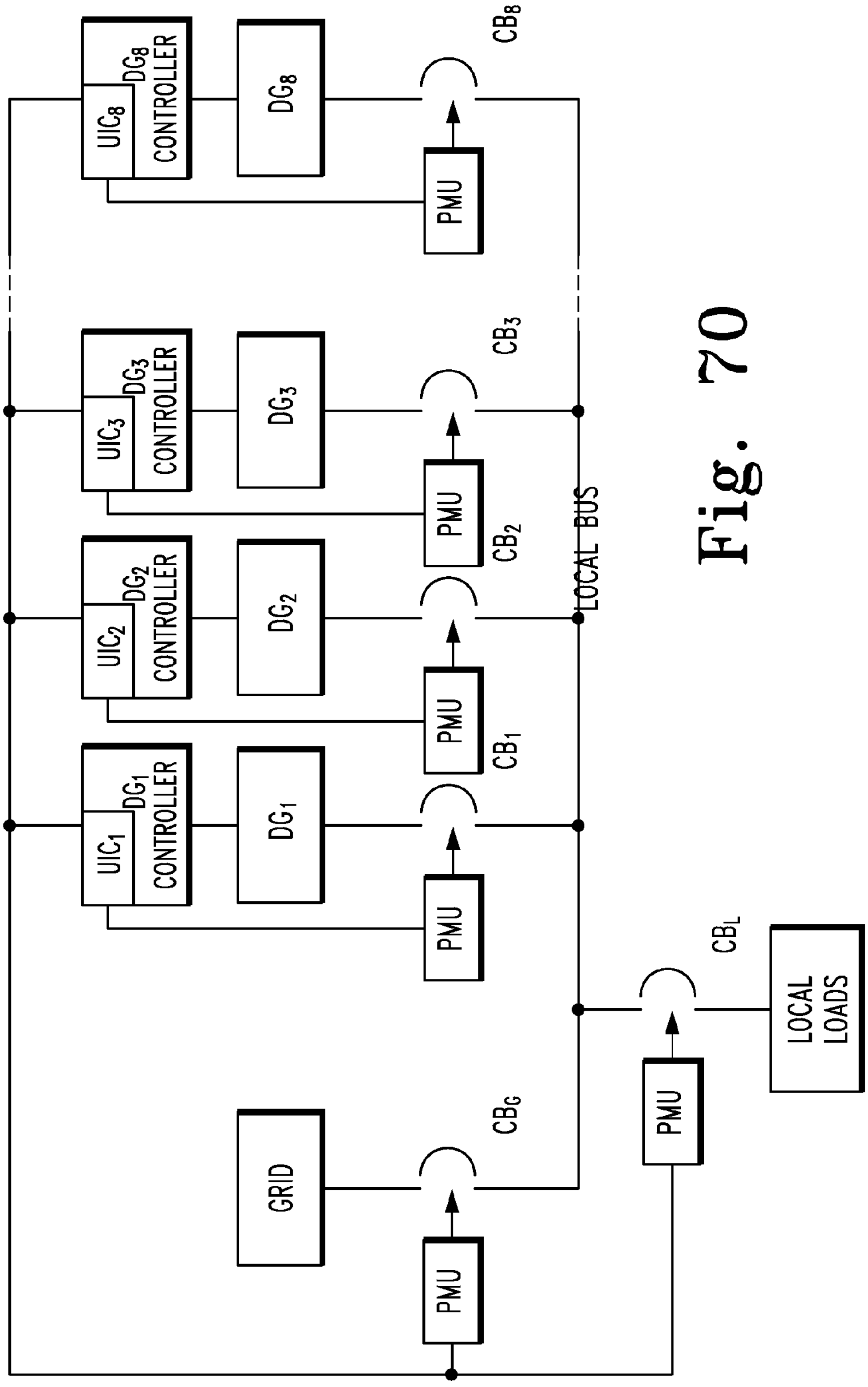

UIC$_8$
DG$_8$ CONTROLLER
DG$_8$
PMU
CB$_8$
UIC$_3$
DG$_3$ CONTROLLER
DG$_3$
PMU
CB$_3$
UIC$_2$
DG$_2$ CONTROLLER
DG$_2$
PMU
CB$_2$
LOCAL BUS
UIC$_1$
DG$_1$ CONTROLLER
DG$_1$
PMU
CB$_1$
CB$_L$
LOCAL LOADS
PMU
GRID
PMU
CB$_G$

Fig. 70

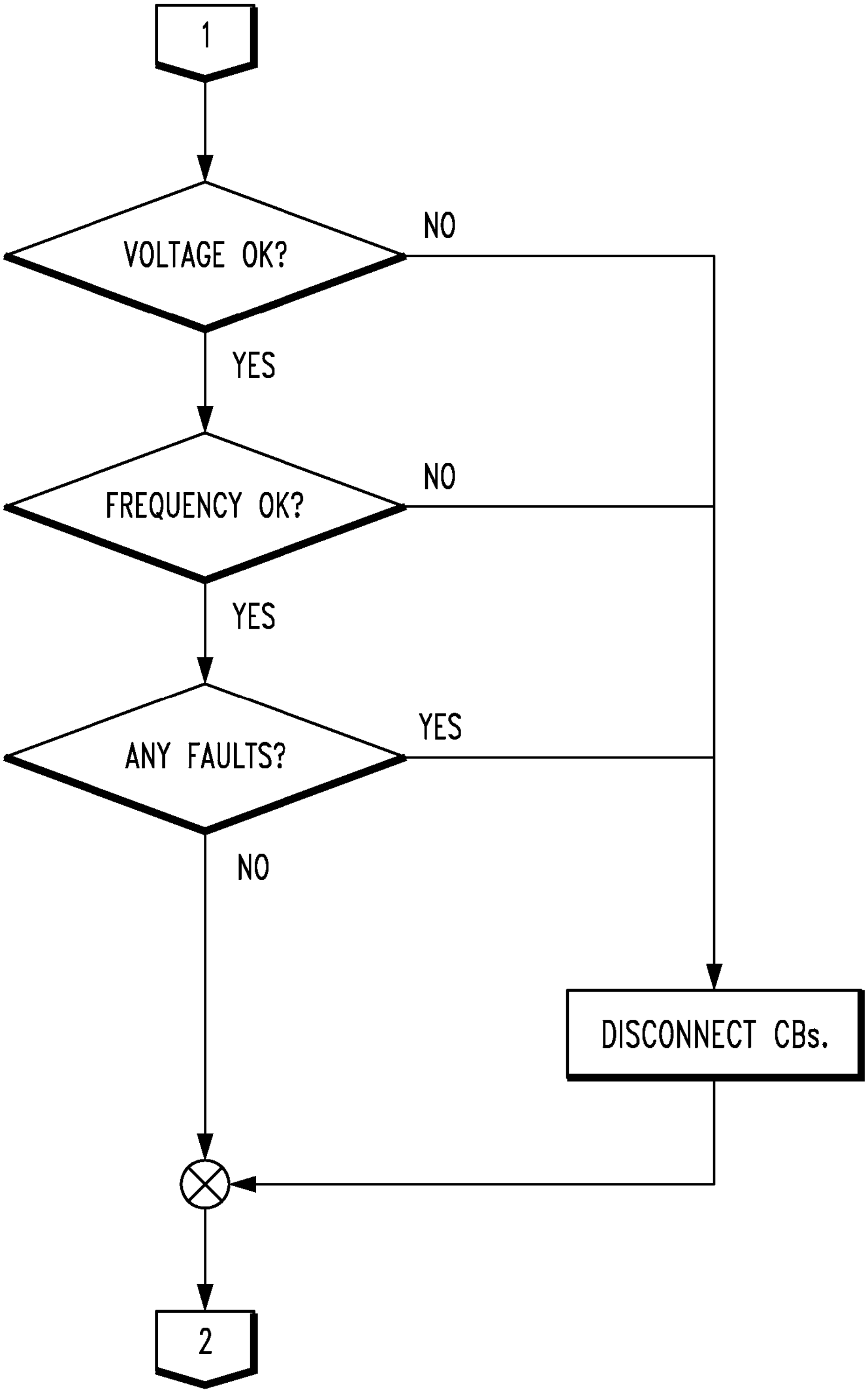

1
VOLTAGE OK?
NO
YES
FREQUENCY OK?
NO
YES
ANY FAULTS?
YES
NO
DISCONNECT CBs.
2

Fig. 71

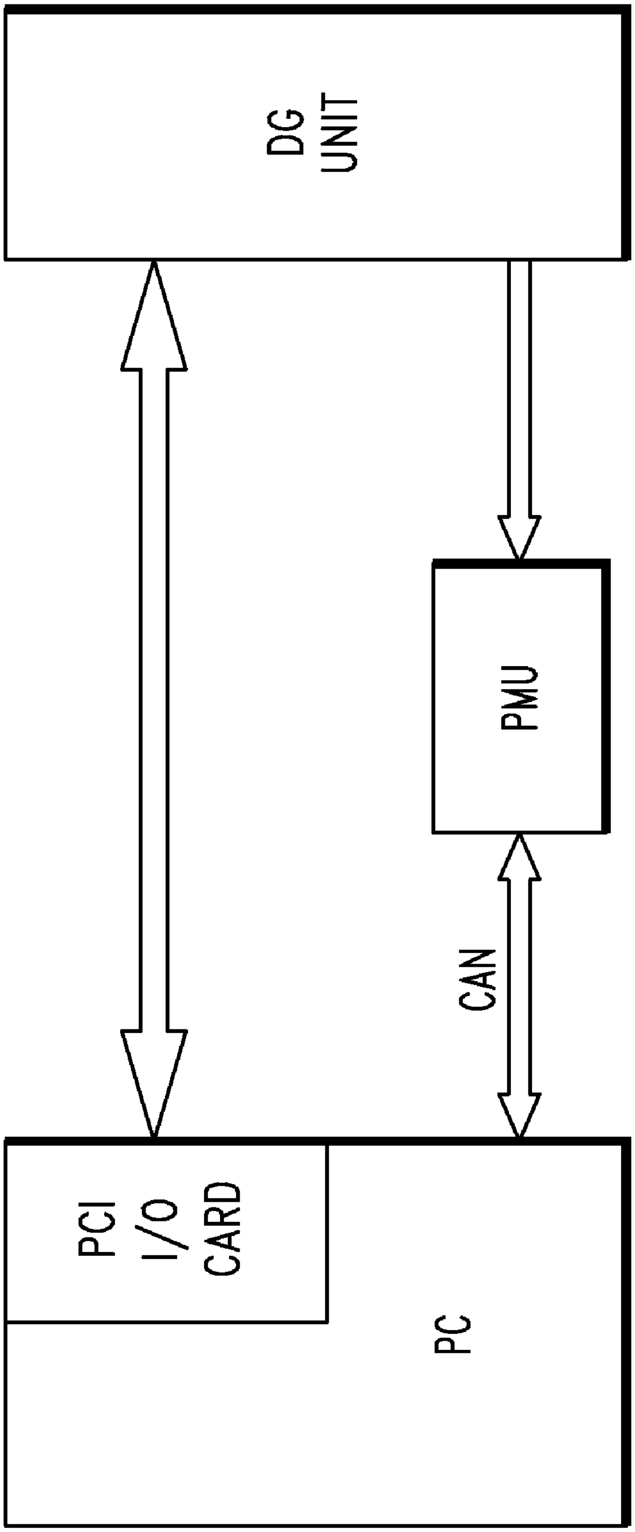


Fig. 72

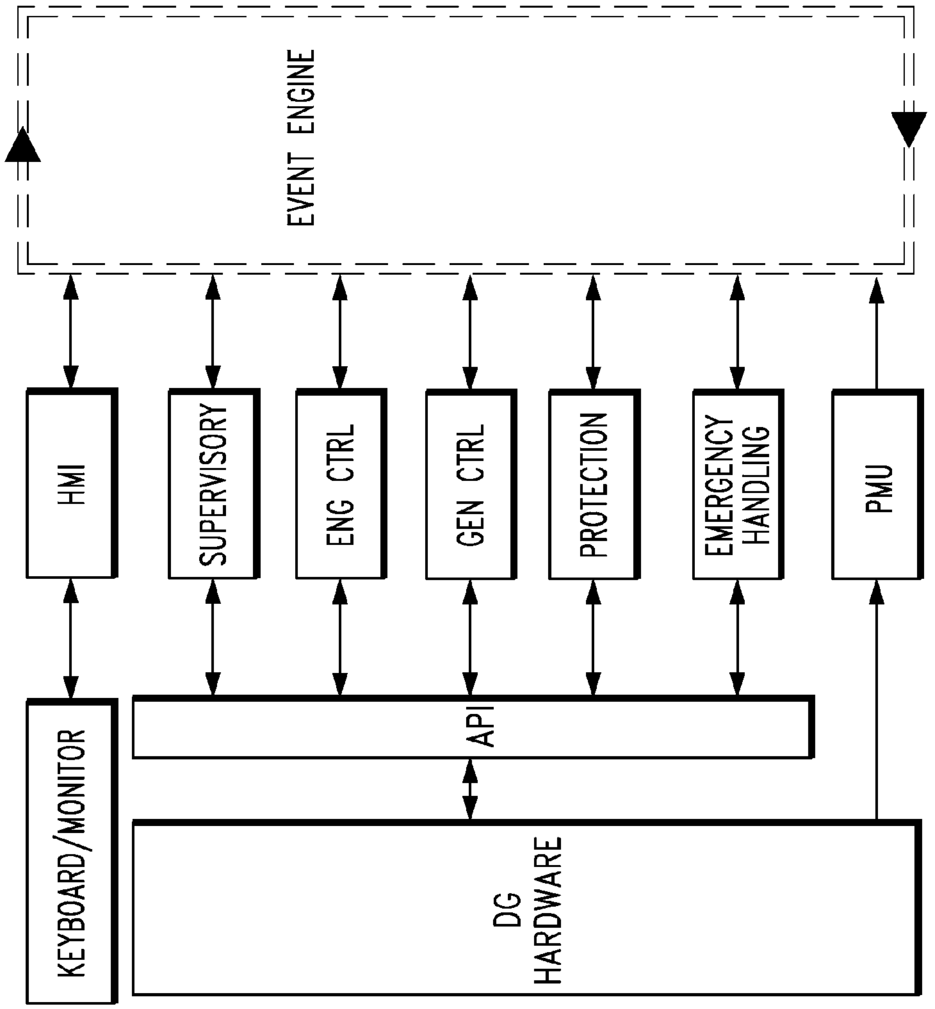

EVENT ENGINE
HMI
SUPERVISORY
ENG CTRL
GEN CTRL
PROTECTION
EMERGENCY HANDLING
PMU
KEYBOARD/MONITOR
API
DG HARDWARE

Fig. 73

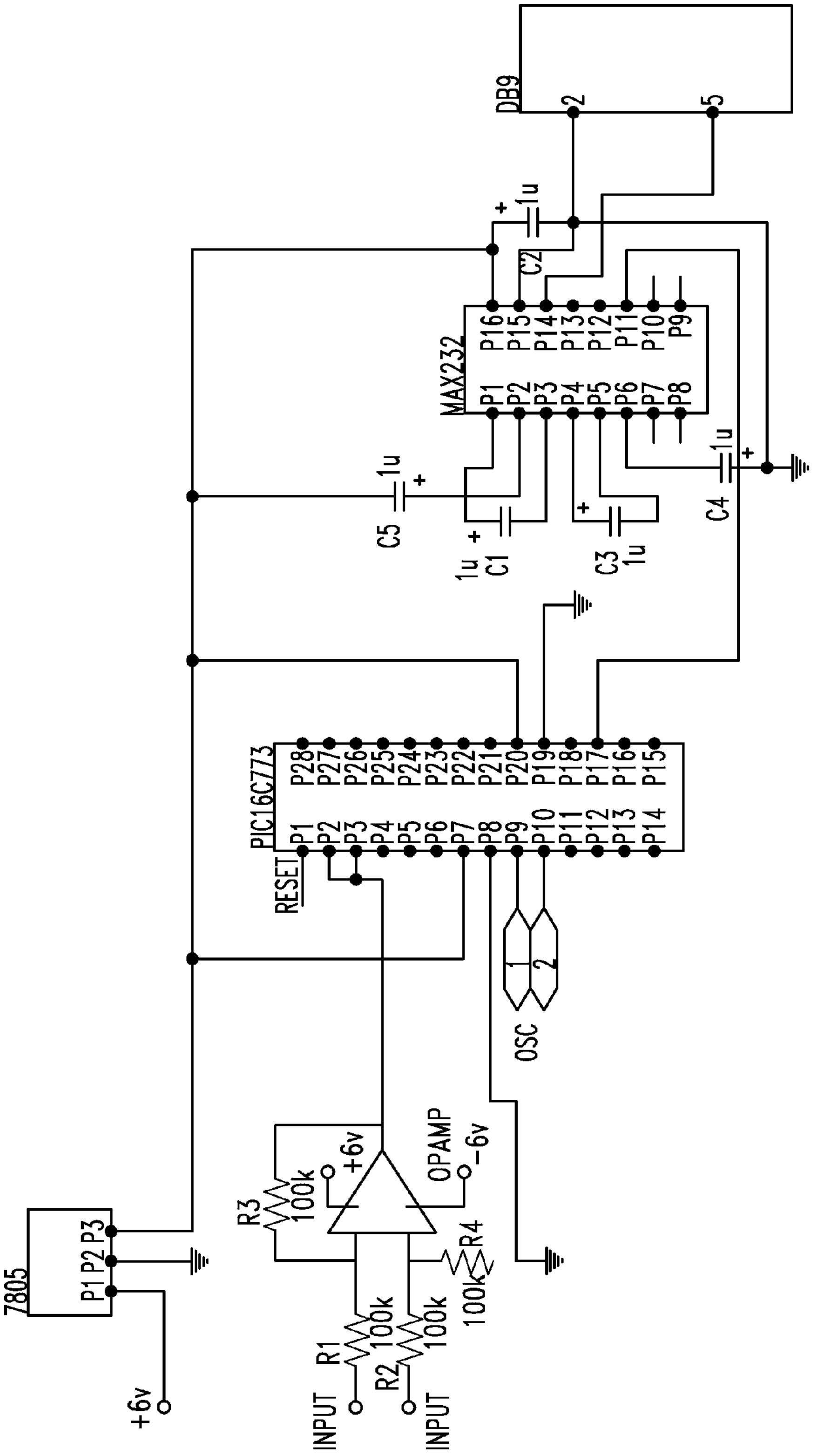

DB9
2
5
1u
+
C2
MAX232
P1
P2
P3
P4
P5
P6
P7
P8
P16
P15
P14
P13
P12
P11
P10
P9
1u
+
C5
+
1u
C1
+
C3
1u
+
1u
C4
PIC16C773
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
P13
P14
P28
P27
P26
P25
P24
P23
P22
P21
P20
P19
P18
P17
P16
P15
RESET
1
2
OSC
7805
P1
P2
P3
+6v
R3
100k
+6v
OPAMP
-6v
R4
100k
R1
100k
R2
100k
INPUT
INPUT

Fig. 74

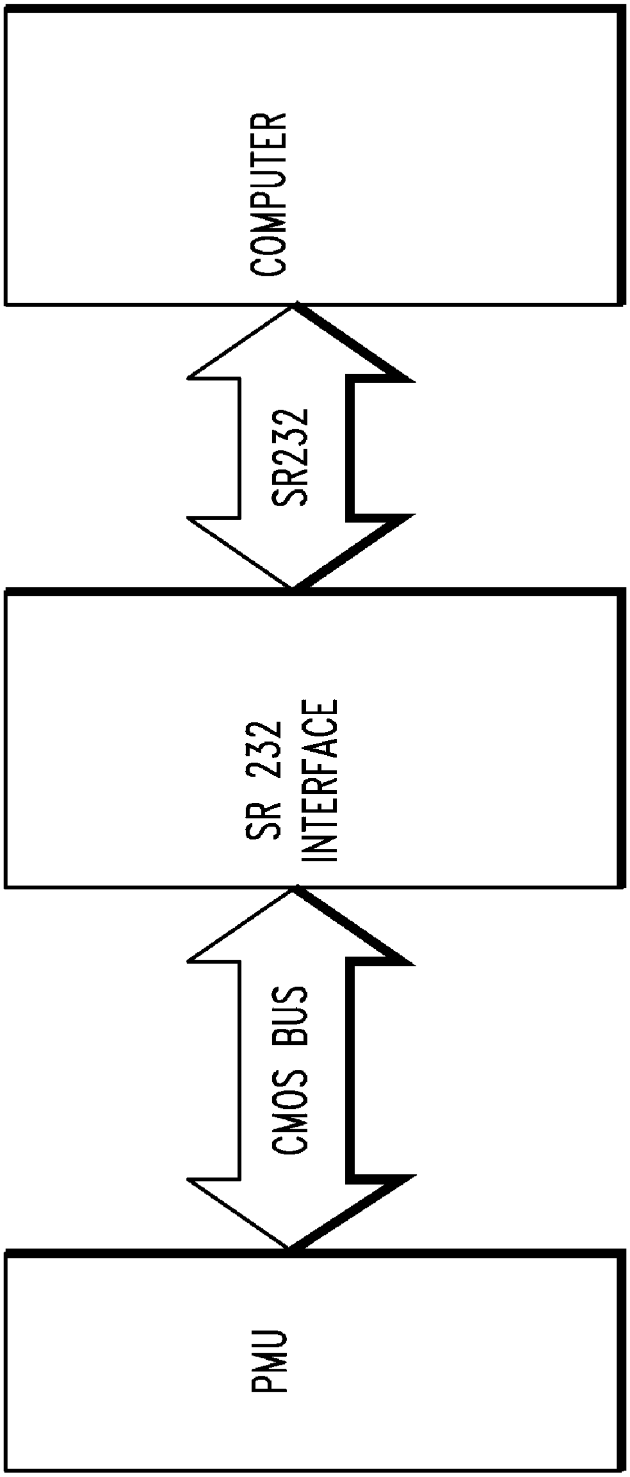


Fig. 75

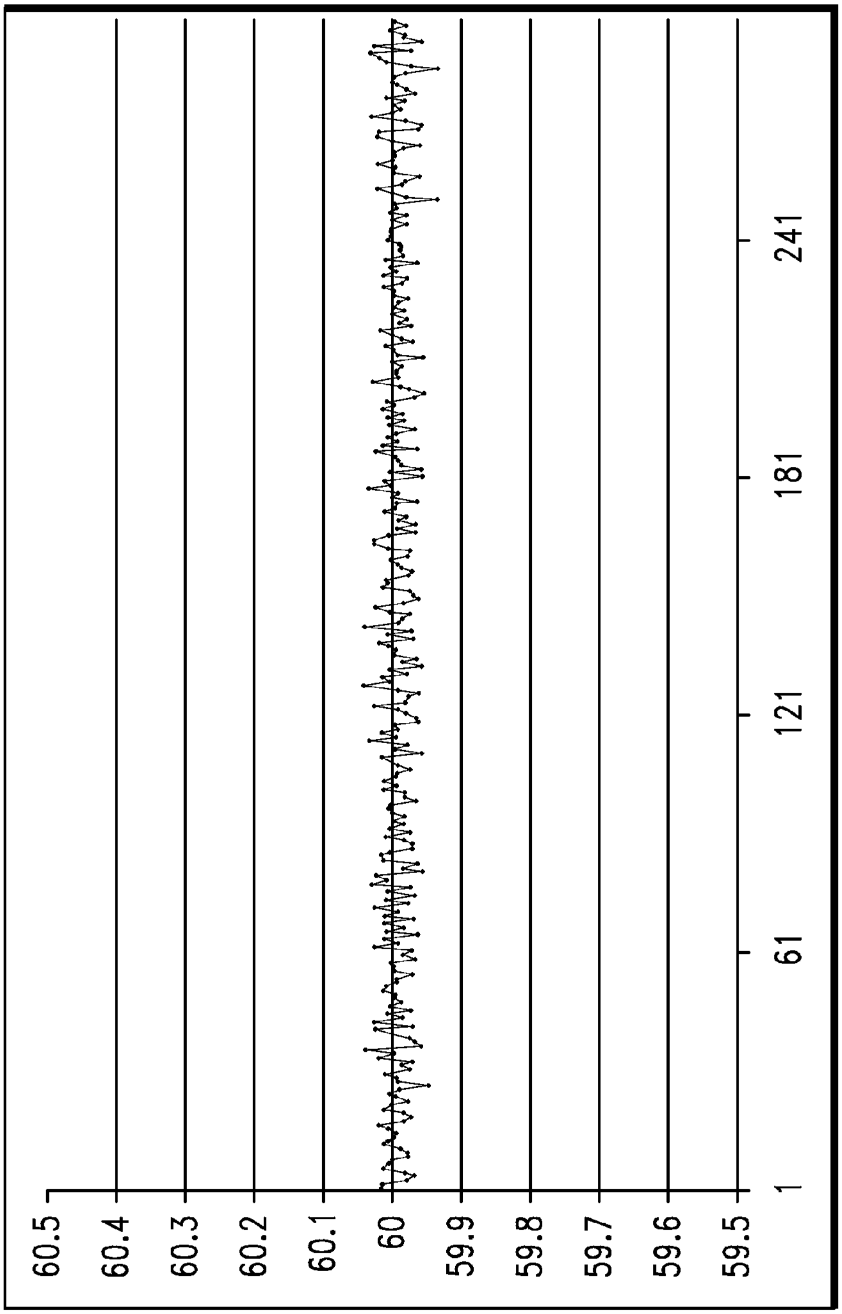


Fig. 76

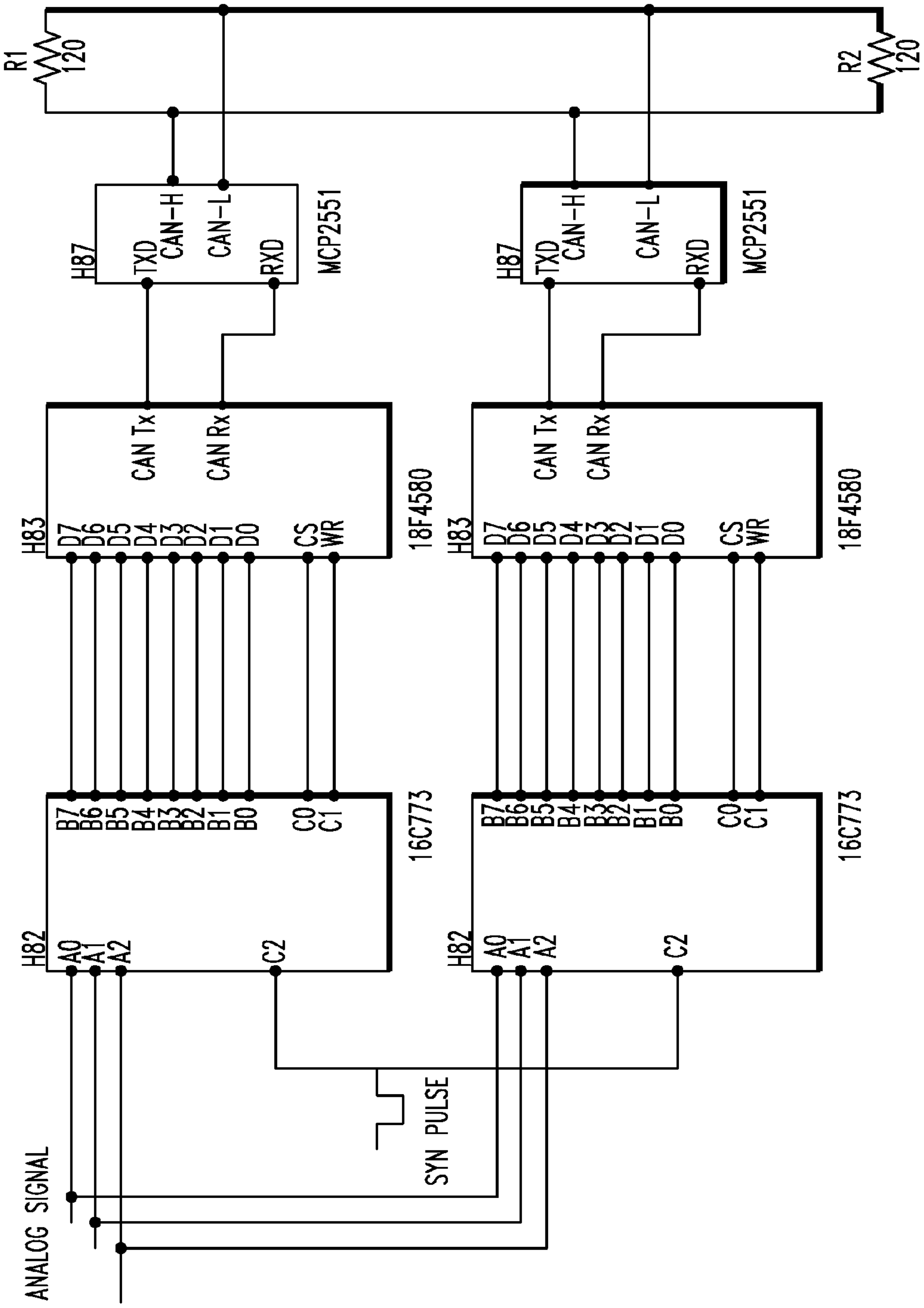

R1
120
R2
120
H87
TXD
CAN-H
CAN-L
RXD
MCP2551
H87
TXD
CAN-H
CAN-L
RXD
MCP2551
H83
D7
D6
D5
D4
D3
D2
D1
D0
CS
WR
CAN Tx
CAN Rx
18F4580
H83
D7
D6
D5
D4
D3
D2
D1
D0
CS
WR
CAN Tx
CAN Rx
18F4580
H82
A0
A1
A2
C2
B7
B6
B5
B4
B3
B2
B1
B0
C0
C1
16C773
H82
A0
A1
A2
C2
B7
B6
B5
B4
B3
B2
B1
B0
C0
C1
16C773
SYN PULSE
ANALOG SIGNAL

Fig. 77

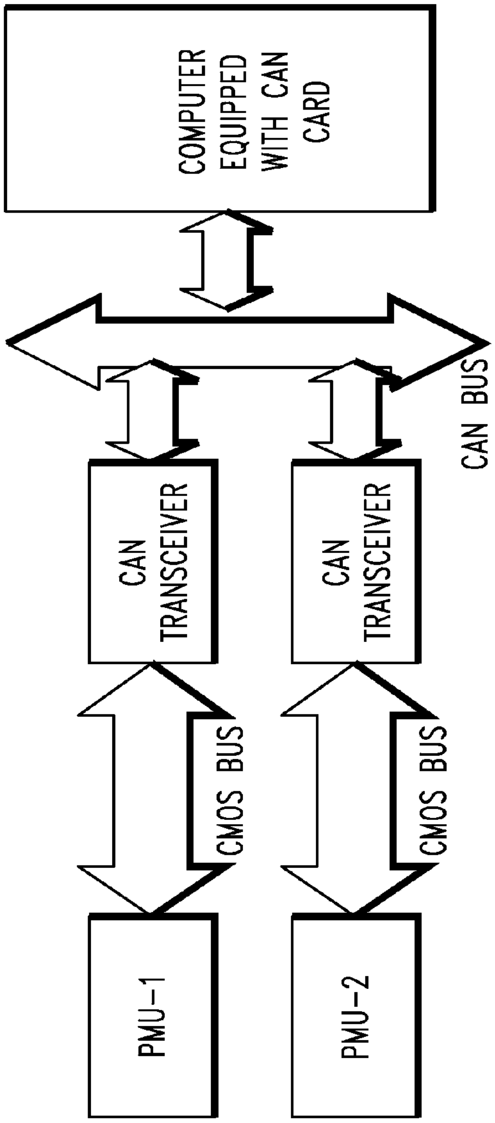

COMPUTER EQUIPPED WITH CAN CARD
CAN BUS
CAN TRANSCEIVER
CAN TRANSCEIVER
CMOS BUS
CMOS BUS
PMU-1
PMU-2

Fig. 78

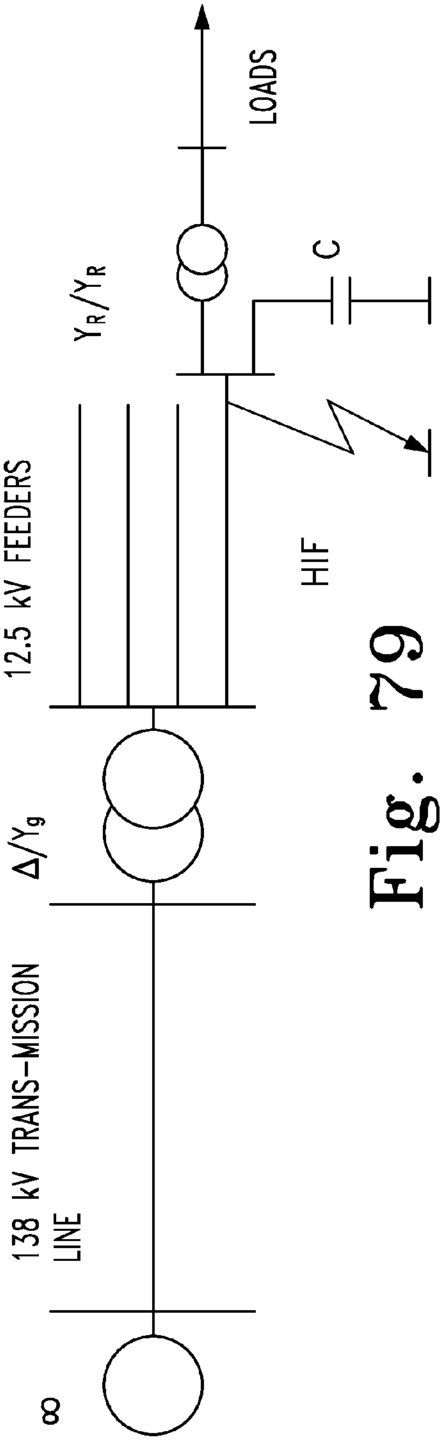

∞
138 kV TRANS-MISSION LINE
Δ/Yg
12.5 kV FEEDERS
YR/YR
C
HIF
LOADS

Fig. 79

INTEGRATED AND OPTIMIZED DISTRIBUTED GENERATION AND INTERCONNECT SYSTEM CONTROLLER

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application is a national stage application filed under 35 USC § 371 based on PCT/US07/73900, having an international filing date of Jul. 19, 2007, which claims priority to U.S. Provisional Patent Application Ser. No. 60/807,786, filed Jul. 19, 2006.

TECHNICAL FIELD

[0002] The present invention relates to distributed power generation systems. More specifically, the present invention relates to systems that control or include a combination of distributed generation resources, one or more loads, and utility grids.

BRIEF DESCRIPTION OF THE DRAWING

[0003] FIG. **1** is a block diagram of the functional components of a DG system controller described herein.

[0004] FIG. **2** is a block diagram of the hardware components of a DG system controller described herein.

[0005] FIG. **3** is a block diagram of a power management system according to the present disclosure.

[0006] FIG. **4** is a graph of system efficiency versus load power for optimal, rule-based, and load-following power distribution in a system having two 75 kw units.

[0007] FIG. **5** is a graph of load power output in a system having two 75 kw units.

[0008] FIG. **6** is a graph of system efficiency versus requested load power for optimal, equal-split and load-following power distribution strategies in a system having two 75 kw units.

[0009] FIG. **7** is a graph of unit load power versus total requested load power for a system of three 75 kw units.

[0010] FIG. **8** is a graph of system efficiency versus requested load power for optimal, equal-split and load-following power distribution strategies in a system having one 75 kw unit and one 150 kw unit.

[0011] FIG. **9** is a graph of unit load power versus total requested load power for a system of one 75 kw unit and one 150 kw unit.

[0012] FIG. **10** is a graph of total system efficiency versus total requested load power for a system of two 75 kw units and one 150 kw unit.

[0013] FIG. **11** is a graph of unit load power versus total requested load power for a system of two 75 kw units and one 150 kw unit.

[0014] FIG. **12** is a graph of total system efficiency versus total requested load power for a system of two 75 kw units and one 150 kw unit.

[0015] FIG. **13** is a graph of unit load power versus total requested load power for a system of one 75 kw unit and two 150 kw units.

[0016] FIG. **14** is a block diagram of a fuzzy controller for use in connection with any DG system.

[0017] FIG. **15** is a Matlab/Simulink model of a rule-based power distribution implementation for two 75 kw units.

[0018] FIGS. **16** (A) and (B) are Matlab/Simulink models of the "IF Action Subsystem" and "ELSE Action Subsystem," respectively, for use in the model of FIG. **15**, respectively.

[0019] FIG. **17** is a Matlab/Simulink model of a rule-based power distribution implementation for three 75 kw units.

[0020] FIGS. **18** (A), (B) and (C) are Matlab/Simulink models of subsystems for use in the "IF Action Subsystem," "ELSEIF Action Subsystem," and "ELSE Action Subsystem" of FIG. **17**, respectively.

[0021] FIG. **19** is a Matlab/Simulink model of a rule-based and fuzzy rule-based power distribution implementation for a system having one 75 kw unit and one 150 kw unit.

[0022] FIGS. **20** (A), (B) and (C) are Matlab/Simulink models of subsystems for use in the "IF Action Subsystem," "ELSEIF Action Subsystem," and "ELSE Action Subsystem" of FIG. **19**, respectively.

[0023] FIG. **21** is a fuzzy logic controller FIS structure for an "ELSE subsystem" in a system having two different distributed generator units.

[0024] FIG. **22** is a graph of fuzzy set membership functions for an "ELSE subsystem" in a system having two different distributed generator units.

[0025] FIG. **23** is a Matlab/Simulink model of a hybrid fuzzy rule-based power distribution implementation for controlling a system having two 75 kw units and one 150 kw unit.

[0026] FIGS. **24** (*a*) and (*b*) are Matlab/Simulink models of the "2ELSEIF Action Subsystem" and "ELSE Action Subsystem" for use in the model of FIG. **23**.

[0027] FIG. **25** is a fuzzy logic controller FIS structure for an "ELSE subsystem" in a system having two different distributed generator units as illustrated in FIG. **23**.

[0028] FIG. **26** is a graph of fuzzy set membership functions for an "ELSE subsystem" in a system having two different distributed generator units as illustrated in FIG. **23**.

[0029] FIG. **27** is a Matlab/Simulink model of a hybrid fuzzy rule-based power distribution implementation for controlling a system with one 75 kw units and two 150 kw unit.

[0030] FIGS. **28** (A) and (B) are Matlab/Simulink models of the "2ELSEIF Action Subsystem" and "ELSE Action Subsystem" blocks for use in the model of FIG. **27**.

[0031] FIG. **29** is a fuzzy logic controller FIS structure for an "ELSE subsystem" in a system having two different distributed generator units as illustrated in FIG. **27**.

[0032] FIG. **30** is a graph of fuzzy set membership functions for an "ELSE subsystem" of two different distributed generator units as illustrated in FIG. **27**.

[0033] FIG. **31** is a graph of a rule-based controller implementation for managing power distribution from two 75 kw units.

[0034] FIG. **32** is a graph of system efficiency of a rule-based implementation according to FIG. **31**.

[0035] FIG. **33** is a graph of system efficiency for rule-based implementation and optimal power distributions in a system having two 75 kw units.

[0036] FIG. **34** is a graph of power output distribution in a rule-based controller implementation in a system having three 75 kw units.

[0037] FIG. **35** is a graph of system efficiency of a rule-based controller implementation for use in a system having three 75 kw units.

[0038] FIG. **36** is a graph of system efficiency in a rule-based controller implementation and an optimal power distribution scenario in a system with three 75 kw units.

[0039] FIG. **37** is a graph of unit power output in a hybrid fuzzy rule-based controller implementation in a system having one 75 kw unit and one 150 kw unit.

[0040] FIG. **38** is a graph of system efficiency for a hybrid fuzzy rule-based controller implementation in a system having one 75 kw unit and one 150 kw unit.

[0041] FIG. **39** is a graph of system efficiency under a hybrid fuzzy rule-based implementation and under an optimal power distribution scenario for a system having one 75 kw unit and one 150 kw unit.

[0042] FIG. **40** is a graph of unit power output under a hybrid fuzzy rule-based controller implementation a system having two 75 kw units and one 150 kw unit.

[0043] FIG. **41** is a graph of system efficiency in a hybrid fuzzy rule-based controller implementation in a system having two 75 kw units and one 150 kw unit.

[0044] FIG. **42** is a graph of system efficiency under a hybrid fuzzy rule-based controller implementation and the optimal power distribution for two 75 kw units and one 150 kw unit.

[0045] FIG. **43** is a graph of unit power output under a hybrid fuzzy rule-based controller implementation in a system having one 75 kw unit and two 150 kw units.

[0046] FIG. **44** is a graph of system efficiency under a hybrid fuzzy rule-based controller implementation in a system having one 75 kw unit and two 150 kw units.

[0047] FIG. **45** is a graph of system efficiency under a hybrid fuzzy rule-based controller implementation and optimal power distribution in a system having one 75 kw unit and two 150 kw units.

[0048] FIG. **46** is a block diagram of the functional components of a distributed generator system controller in one embodiment of the present invention.

[0049] FIG. **47** is a block diagram of the hardware components of a distributed generator system controller in one embodiment of the present invention

[0050] FIG. **48** is a block diagram illustrating inputs and outputs to and from an engine in a distributed generator in one illustrated embodiment.

[0051] FIG. **49** is a block diagram of a quasi-static engine model.

[0052] FIG. **50** is a block diagram of an engine speed feedback control for use in various embodiments of the disclosed system.

[0053] FIG. **51** is a schematic diagram of a generator system.

[0054] FIG. **52** is a graph of internal voltage to excitor voltage under an open circuit condition (OCC).

[0055] FIGS. **53** (A) and (B) are a phasor and a schematic diagram of terminal voltage as used herein.

[0056] FIG. **54** is a block diagram of a generator system including certain control signals as used in various embodiments of the disclosed system.

[0057] FIG. **55** is a block diagram of an AVR and generator module for use some embodiments of the disclosed system.

[0058] FIG. **56** is a Simulink diagram of the AVR and generator module shown in FIG. **55**.

[0059] FIG. **57** is a graph of power, voltage, and current in a simulated system during a load change event.

[0060] FIG. **58** is a graph of power, voltage, and current in a sine wave simulation during a load change event.

[0061] FIG. **59** is a graph of power, engine speed bias, and engine speed in a simulation of an engine speed feedback control (GIM) of one example system.

[0062] FIG. **60** is a block diagram of the engine speed feed-forward plus feedback control (GIM) in some embodiments of the disclosed system.

[0063] FIG. **61** is a graph of power, engine speed bias, and engine speed signals in a simulation of the control illustrated in FIG. **60**.

[0064] FIGS. **62** (A) and (B) are a schematic diagram and phasor diagram, respectively, of power flow among a generator, loads, and tie-line in some embodiments of the disclosed system.

[0065] FIG. **63** is a block diagram of a feedback control system for maintaining a minimum power level from the grid.

[0066] FIG. **64** is a graph of simulation results from a controller in load-following mode (GPM).

[0067] FIG. **65** is a schematic diagram of a classical model of two generators with matching loads connected by a distribution line.

[0068] FIG. **66** is a graph of simulated frequency measurements from a DG unit before and after an islanding event.

[0069] FIG. **67** is a graph of the magnitude of the 1 Hz component of the DG signal shown in FIG. **66**.

[0070] FIG. **68** is a graph of the trip and reset characteristics of an over-current relay for use in the disclosed system.

[0071] FIG. **69** is a schematic diagram of a transmission line in a fault condition.

[0072] FIG. **70** is a block diagram showing GENSETs and UICs as implemented in one embodiment of the disclosed system.

[0073] FIG. **71** is a flowchart illustrating error-handling logic in the controller in certain embodiments of the disclosed system.

[0074] FIG. **72** is a block diagram of a PC-based controller implementation of the current system.

[0075] FIG. **73** is a block diagram of functional components in the system shown in FIG. **72**.

[0076] FIG. **74** is a circuit diagram of a PMU for use in the system shown in FIG. **72**.

[0077] FIG. **75** is a block diagram of communication channels in a subsystem of the system shown in FIG. **72**.

[0078] FIG. **76** is a graph of results obtained from a phasor calculation program for use in one embodiment of the disclosed system.

[0079] FIG. **77** is a schematic diagram of a phasor measurement unit (PMU) according to one embodiment of the disclosed system.

[0080] FIG. **78** is a block diagram showing communication between PMUs and the PC in the system illustrated in FIG. **72**.

[0081] FIG. **79** is a one-line system diagram used to simulate high-impedance fault (HIF) detection techniques implemented in some embodiments of the disclosed system.

DESCRIPTION

[0082] For the purpose of promoting an understanding of the principles of the present invention, reference will now be made to the embodiment illustrated in the drawings and specific language will be used to describe the same. It will, nevertheless, be understood that no limitation of the scope of the invention is thereby intended; any alterations and further modifications of the described or illustrated embodiments, and any further applications of the principles of the invention as illustrated therein are contemplated as would normally occur to one skilled in the art to which the invention relates.

[0083] Generally, one embodiment of the present system implements par distribution strategies to improve overall system efficiency. Another embodiment provides an integrated and optimized distributed generation system control (DGSC) module that improves overall power generation performance and lowers the capital cost of distributed generation site installation in many implementations.

[0084] IEEE Standard 1547 ("the Standard") provides a nationally recognized set of rules for connecting distributed power generation resources to a utility grid. This standard specifies requirements for voltage regulation, grounding, synchronization, inadvertent energizing, monitoring, protected relaying, power quality, islanding, and verification of compliance with itself. The subsequent IEEE Standard 1547.1 specifies test procedures for establishing and verifying compliance with the requirements of IEEE 1547. Some aspects of these standards will be discussed in further detail below.

[0085] Synchronous generators can regulate voltage and supply power to a local load even when they are not connected to the utility grid. DG units operating independently in grid-isolated mode (GIM) can provide primary or backup power when utility service is not available. However, DG facilities with synchronous machines also have the potential to supply voltage and power to a small piece of the utility grid that has been disconnected from the rest of the grid. This phenomenon is called islanding and it should generally be avoided for safety and other reasons. The IEEE 1547 standard requires that the DG detect islanded operation and cease to energize the utility grid within two seconds. The standard allows this requirement to be met using any of various island-detection schemes, as long as the selected method passes a test designed to create the most difficult circumstances for island-detection. That test requires an island to be detected when the DG and a matching load are simultaneously disconnected from the grid. Matching the load and DG output causes minimal frequency change when the grid is disconnected. Some systems currently on the market presently use the simplest technique to satisfy the islanding test, but this approach has an unfortunate drawback. The method simply disconnects the DG from the utility when the power imported from the utility falls below a preset threshold. The unfortunate drawback of this method is that it never allows export of power to the grid. As mentioned above, being able to export power to the grid is important for achieving maximum CHP efficiency. One aspect of the present system includes an island detection scheme that allows power export to the grid.

[0086] Proposed anti-islanding methods can be divided into two categories, namely, passive schemes and active schemes. Anti-islanding relays based on passive schemes include rate of change of frequency (ROCOF) relays and vector surge (VS) relays. Passive methods tend to perform poorly when the active power generated by the DG roughly balances the local load. Active methods overcome this limitation by introducing a small perturbation signal into the DG system. When the utility remains connected, the small disturbance caused by the intentional perturbation is not sufficient to trip island-detection relays. However, if the connection to the grid is lost, the response to the perturbation becomes large enough to activate island-detection relays. The challenge of active methods is to keep the perturbation small enough to avoid stability and power quality problems but large enough to cause a rapidly detectable response once the grid is lost.

[0087] Some forms of the present system include an optimized controller that allows a system to satisfy the requirements if IEEE 1547 and still export power to the grid with a system that could be used in many different jurisdictions. Built-in regulation features and anti-islanding capability in some embodiments provide operational benefits including more efficient power generation. In addition, the system's ability to export unneeded power to the grid allows distributed generation capacity to be sized by thermal load requirements for more efficient operation. Still further, some embodiments of the system provides significant capital- and installation-cost savings. Some embodiments incorporate relay components into a turn-key controller that eliminates the need for a trained technician to travel to an installation site to program the controller. Factory testing of some embodiments eliminates costly on-site installation delays. Still further, the supervisory control scheme in several embodiments allows users to add distributed power generation modules over time without undue additional engineering effort.

[0088] The present disclosure is presented in two parts. The first is directed primarily to a control system that optimizes distributed generation of power by homogeneous and heterogeneous collections of generators. The second part discusses additional aspects of the distributed power generation system.

Part 1—Optimized Controller

[0089] The power management system plays a key role in improving the overall system efficiency for multiple distributed generation units. The system performance, such as fuel economy, depends in part on the power management system strategy. The two conventional rule-based power distribution methods used in one existing design are quite simple, and only have been applied to two units. One method is to split the power equally between two identical units, and the other method is that the master unit always run full load, and the slave unit takes the rest of the load. These two conventional methods are not the optimal power distribution strategy.

[0090] This Part describes optimal power distribution strategies to improve overall system efficiency. A sequential quadratic programming method will be applied to find the optimal power distribution among multiple units. The optimal power management system is an important control function of the system control. This Part not only describes the optimal strategies for two identical units, but also describes strategies for three identical units, two different units, and three different units. Simulation results show that the disclosed optimal power management system can improve the system efficiency over the certain rule-based power management system.

[0091] A challenge to the full deployment of distributed energy is the fact that the electric power grid system is under the authority of hundreds of utilities and regulatory commissions. Developing a secure distributed generation (DG) system of energy production is hindered by the wide variety of technologies to produce power and the various grid suppliers' requirements for power acceptance.

[0092] A purpose of the present disclosure is to describe an integrated and optimized DG system controller module that improves the overall power generation performance attributes, and lowers the capital cost of a distributed generation site installation. The proposed system is intended to leverage the newly released IEEE 1547 *Standard for Interconnecting Distributed Resources with Electric Power Systems* that has defined a nationally recognized standard for grid interconnect for DG resources.

[0093] The overall DG system controller functions units include a supervisory controller, a DG unit controller, utility interface controller (UIC), and a DG unit protection functions and multiple unit configurations. Based on the operation of the system, the control system can be characterized as a hierarchical control structure with three distinct levels: supervision, coordination, and execution. The supervisory controller, which includes the UIC, is in the supervision level controller. The DG unit controller is in the coordination level. The execution level includes engine, engine control module (ECM), generator, and automatic voltage regulator (AVR) components.

[0094] This Part describes optimal power management strategies for multiple DG units, which is a significant function of the system controller. One objective of the optimal power management system (PMS) is to find an optimal way to distribute load power among multiple units to improve system efficiency. In this Part the optimal PMS of the DG system is described. The optimal rule-based approach, a hybrid fuzzy rule-based PMS will distribute power among multiple DG units, which can optimize the entire system efficiently.

[0095] Certain existing power distribution system are only used for power distribution between two units, and power distribution methods are quite simple. Two conventional methods in the existing art to distribute power between two DG units in industry involve either splitting the power equally between these two identical units, or a load-following method (the master unit always runs full load, and the slave unit takes the rest of the load). These two conventional methods are not optimal power distribution strategies. In this Part, the optimal power distribution strategies are shown to improve the overall system efficiency. The sequential quadratic programming (SQP) method is applied to find the optimal power distribution between multiple units. The theoretical derivation is presented for the problem. Before introducing the optimization, the overall control structure of the DG system is presented. The optimization and implementation of the optimal PMS will be one layer of the system control. This Part develops the optimal strategies for two identical units, for three identical units, for two different units and for three different units.

[0096] Optimal rule-based systems and hybrid fuzzy rule-based systems are designed in this disclosure for different configurations. In this Part, the fuzzy logic rules are tuned according to the optimal solution data. The controller for the DG unit is implemented and tested in hardware and software.

[0097] 1.2. Main Subjects of this Part

[0098] Find optimal power distribution among multiple units by using SQP.

[0099] Compare optimal PMS with existing power distribution methods.

[0100] Show design of rule-based PMS for identical units.

[0101] Show design of Hybrid fuzzy rule-based PMS for different units.

[0102] 1.3. Organization of the Part

[0103] In this Part, the preceding issues are organized as follows: Chapter 2 discusses distributed generation systems, system controller architecture, and simulation model development for the DG unit. Chapter 3 describes optimal power management and control strategies, finds optimal load distribution solutions among multiple DG units, and compares optimal strategies with the two conventional methods. Chapter 4 describes two implementation methods, which are rule-based optimal power distribution and hybrid fuzzy rule-based optimal power distribution. In Chapter 4, the implementation results also are analyzed. Finally, Chapter 5 summarizes the teachings of this disclosure.

2. DG System and System Controller Architecture

[0104] 2.1. Principles of DG

[0105] The DG unit in many embodiments can be used as a backup power supply for industry, and it also can sell electricity to the power grid. In one embodiment, a DG unit is an internal combustion engine generating system with a synchronous generator. In other embodiments DG engines are gasoline engines, diesel engines, and natural gas engines. The DG unit discussed in this Part is a natural gas fueled engine driving a synchronous generator that produces 60 Hz three-phase electrical power at 277/480 VAC under ISO standard conditions. Two types of DG units will be discussed in this Part. Their rated output powers are 75 kw and 150 kw respectively.

[0106] Other embodiments will use other energy sources and have other output ratings or other characteristics. In these embodiments, the DG unit can work in grid isolated mode (GIM) or grid parallel mode (GPM) mode. DG units are connected to the local utility power grid in GPM or to the local site distribution bus in GIM.

[0107] 2.2. DG System Control Architecture

[0108] The overall structure of the DG multiple units system is a hierarchical system, which is illustrated in FIG. **1**. The DG system controller coordinates the control of one or more DG units as shown in FIG. **1**. The system controller is capable of controlling up to 8 parallel DG units that are connected to the local utility grid in GPM or to the local site distribution bus in GIM. In this Part, controlling 2-3 parallel units will be discussed. It can be seen from FIG. **1** that the DG subsystem modules communicate with a supervisory controller. The supervisory controller performs the UIC functions.

[0109] There are two types of controllers in this embodiment: the DG unit controllers and the supervisory controller. A DG unit controller controls a single DG unit, while the supervisory controller coordinates the operation of multiple units in a single location. The functions of the supervisory controller are:

[0110] Grid synchronization and protection of the DG unit

[0111] Human/automatic switch between GPM/GIM

[0112] Control of human machine interface (HMI)

[0113] Communication among multiple units and external DG gateway

[0114] This Part will discuss the supervisory controller. There are two major functions of the supervisory controller that will be mentioned. The first function is controlling the GIM/GPM mode switch. The second is distributing load power among multiple units to maximize the system efficiencies by using a built-in power management system. The power management system will be the focus of this Part and will be discussed in the remaining chapters. Table 2.1 shows the supervisory control functions for the GIM/GPM mode switch. The switch is a logic control.

[0115] The DG system can run in GIM or GPM. Each DG unit will be running at either speed control mode or torque control mode according to the situation. The supervisory controller will determine the control mode of each DG unit. Table 2.1 shows the basic rules.

TABLE 2.1

Supervisory control functions for GIM/GPM mode switch (logic control)

Configuration	GIM	GPM
Single DG unit	Voltage-regulating speed control	Voltage-following torque control

TABLE 2.1-continued

Supervisory control functions for GIM/GPM mode switch (logic control)

Configuration	GIM	GPM
Multiple DG units	Determine master unit. One master unit uses Voltage-regulating speed control, other units use Voltage-following torque control.	All units use Voltage-following torque control.

[0116] FIG. **2** shows hardware-oriented block diagram for this embodiment of a DG system. It can be seen from FIG. **2** that each system controller includes a supervisory controller. Then supervisory controller communicates with other units to determine the master unit and slave units, and then determinates the load power percentage to each unit when there are multiple units. A Phasor Measurement Unit (PMU) was used to measure the phasor of load power and then control the circuit break.

[0117] Additionally, the system controller includes a GIM/GPM switch control, load-following mode control, and UIC and DG unit controller. The GIM/GPM control is a digital logical control used to switch modes between GIM and GPM. The DG supervisory controller and UIC are preferably housed in a single enclosure (being in some cases a unitary housing, and in others a plurality of housings connected detachably or non-detachably by conduit or other substantially sealed passage), though in some embodiments they are in unconnected cases.

[0118] A PMS was developed for the DG system controller. The PMS is at the supervisory control level. An objective of the PMS is to manage the power distribution and coordinate among multiple units to improve the system efficiency. The execution level includes the DG unit controller, the engine, and the generator used to perform the specific tasks according to the desired commands generated by the PMS. The PMS accepts the signals from the supervisory controller and determines the splitting power demand to each DG unit controller using predefined strategies. The PMS also converts the load power demand to torque commands, which are sent to the DG unit controller as a load disturbance. There are several ways to distribute power among multiple units. Equal-split load power and load-following methods are used in some systems because they are easy to implement. However, these two methods are not the optimal power distribution methods. In this Part, an optimal power distribution system is described to optimize the entire system efficiency when multiple DG units are used. Chapter 3 characterizes the system efficiency problem and solves it mathematically using the Matlab optimization toolbox. In Chapter 4, the implementations of the optimal power management strategies are shown in Matlab/Simulink. The implementations are rule-based and hybrid fuzzy/rule-based power distribution.

3. Optimal Load Power Distribution Among Multiple Units

[0119] 3.1. Power Management System Reviews

[0120] Power management is one of the functions in the supervisory controller. The system performance, such as the fuel economy, depends on the power management strategy.

[0121] The PMS is at the supervisory control level. The primary objective of the PMS is to manage the power distribution and coordinate among multiple units to improve system efficiency.

[0122] 3.1.1. Inputs and Outputs

[0123] The inputs and outputs of the PMS module in this embodiment are illustrated in FIG. **3**. Its input signal reflects the total required load power. Its outputs are the required load power values for each unit. For example, if there are three units, then the PMS has three output signals. The PMS accepts the signals from the supervisory controller and determines the assignment of power generation to each DG unit controller using predefined strategies. The PMS then converts the load power demand to torque commands, which will be sent to the DG engine controller.

[0124] 3.1.3. Optimal PMS

[0125] Two very simple rule-based methods are used for power distribution among multiple DG units in some existing systems. These two methods were developed on the basis of simple analysis of component efficiencies and these two modes can be switched manually. One method is load-following by a single unit while all the other units are supplying no load or full load. The other method is to split power equally between multiple units.

[0126] These rule-based PMSs, although easy to implement, yield far from the optimal system efficiency. Moreover, they cannot determine whether the system efficiency is optimal or not.

[0127] There should in fact be an optimal power distribution among multiple DG units that maximizes the system efficiency. An optimal PMS can optimize the system-wide fuel efficiency for multiple units.

[0128] There are several approaches to designing an optimal PMS. Math-optimal PMS (such as SQP or dynamic programming (DP)) can achieve the exact optimal efficiency. However, this is often too time-consuming for online calculation. Rule-based PMS is easy to use, but it is practically limited to simple optimization rules; otherwise it will be difficult to obtain the rules, and will be time consuming to apply. Fuzzy PMS is not as precise as math-model based PMS (SQP or DP); however, it is convenient and fast for online implementation. If a look-up table is used, it can more precise than a fuzzy system if there is sufficient optimal data in the look up table. A comparison among these five PMS design approaches is given in Table 3.1.

TABLE 3.1

Comparison of five PMS design approaches

Method	Advantages	Disadvantages
Rule-based	Simple, easy to implement	Intuitive, difficult to obtain exact rules
SQP	Well developed nonlinear optimization method	Static optimization
DP	Global optimization	Time consuming, the future load profile is needed
Fuzzy rule-based	Practical, robust, easy to implement online	Difficult to obtain expert data to design fuzzy rule
Look up table	Practical, easy to implement online	Not robust against environmental variations

[0129] 3.2. Optimal PMS Design Procedure for DG System

[0130] The design procedure preferably begins with formulating the problem, defining a cost function, and then selecting a reliable and state-of-the-art optimization algorithm. The algorithm is preferably applied to minimize the cost function and optimize power distribution among multiple units. Finally, the feasible control rules, summarized from the statistical analysis of the optimal solution, are employed to develop a PMS for online power distribution implementation. The performance of the power management strategy is validated in the DG unit model, described herein. Finding the optimal power distribution will be described in this chapter.

[0131] 3.3. Introduction to SQP

[0132] A cost function was developed that included a set of constraints. The cost function of system efficiency includes distributed load power for each unit, and an efficiency map of each unit. The power distribution was formulated as a constrained nonlinear optimization problem. In the meantime, the system efficiency should be maximized and a set of constraints should be satisfied.

[0133] SQP techniques are applied to solve the optimization problem. The optimizer can make well-informed decisions regarding directions of search and step length. The SQP subroutine in the Matlab optimization toolbox performs these steps:

[0134] 1. Initialize;

[0135] 2. Calculate the gradient of the cost function given the constraints;

[0136] 3. Define a positive definite Hessian matrix using the BFGS formula based on the cost function and constraints;

[0137] 4. Solve the quadratic programming (QP) problem using a modified simplex method;

[0138] 5. Use linear search and merit function method to determine the new search direction and step size;

[0139] 6. Check whether the termination criterion is satisfied; if not, then go to step 2.

[0140] It can be seen that in this embodiment, at each iteration of the SQP method, a QP sub-problem is solved, and an estimate of the Hessian of the Lagrangian is updated at each iteration using the Broyden-Fletcher-Goldfarb-Shanno (BFGS) Formula. Therefore, the SQP method converts the original optimization problem to a QP problem using the quadratic approximation of the objective function and the constraints. The SQP method closely mimics Newton's method for constrained optimization just as is done for unconstrained optimization. At each major iteration, an approximation is made of the Hessian of the Lagrangian function using a quasi-Newton updating method. This is then used to generate a QP sub-problem whose solution is used to form a search direction for a line search procedure.

[0141] 3.4. How to Optimize the Problem in Matlab

[0142] In the Matlab optimization toolbox, the functions fmincon, fminimax, fgoalattain, and fseminf each use SQP. Among them, fmincon attempts to find a minimum of a constrained nonlinear multivariable function starting at an initial estimate. This is generally referred to as constrained nonlinear optimization or nonlinear programming.

[0143] 3.5. Initial Value xo

[0144] The fmincon function does work properly, but the function fmincon will usually return a local minimizer in the vicinity of the starting point that the user supplies. If the model has more than one local minimizer, then different answers may be obtained for different starting points because the solver may converge to different solutions depending on the starting point. The user, using knowledge about the problem, provides the initial start point. Therefore, one should choose the initial vector xo to be in the feasible set.

[0145] If the problem has more than one local minimizer, a common approach is to sample the problem domain and optimize from different initial guesses. Therefore, all the local minimizers should be found and then the global minimum should be searched from those local minimizers.

[0146] A second option to increase the chances of finding the global minimizer for a nonlinear function is to run a genetic algorithm (GA), like the GA solver in the Genetic Algorithm and Direct Search Toolbox. However, this algorithm does not admit nonlinear constraints. A third option is to run PATTERNSEARCH solver, which admits nonlinear constraints.

[0147] A fourth option is to obtain all the local optimizers of different starting points xo by sampling the domain of the variables. Because the optimal minimizer is not only one point, but rather, a sequence of data points, the problem was divided into several regions. In each region, the problem domain was sampled from different initial guess xo. The global minimizer was then obtained.

[0148] 3.6. Problem Formulations and Solutions

[0149] The optimization problem can be formulated as follows.

To maximize system efficiency f=f (x_i); is the same as minimizing F=1/f(x_i);

Subject to Constraints:

[0150]

$$x_1+x_2+\ldots+x=P_{out};$$

$$0 \leqq x_1 \leqq c_1;$$

$$0 \leqq x_2 \leqq c_2;$$

$$0 \leqq x_3 \leqq c_3;$$

$$\ldots$$

$$0 \leqq x_i \leqq c_i$$

[0151] In the experiments described in this Part, two types of DG units were used. One type is a 75 kw unit; the other is a 150 kw unit. Five different configurations are discussed:

[0152] Two identical 75 kw units.

[0153] Three identical 75 kw units.

[0154] Two different units with one 75 kw and one 150 kw.

[0155] Three different units with two 75 kw and one 150 kw unit.

[0156] Three different units with one 75 kw unit and two 150 kw units.

[0157] The system efficiency optimization for each of these five scenarios will be analyzed herein. Each configuration will be formulated with a certain cost function. With each of these cost functions, the SQP method was used to obtain the optimize solutions by using the Matlab optimization toolbox. Two identical 75 kw units will be evaluated first.

[0158] 3.6.1. Two Identical 75 kw Units

[0159] 3.6.1.1. Problem Formulation

[0160] The total system efficiency is derived as follows:

$$\begin{aligned}
\text{System efficiency} &= \frac{\text{Total output power}}{\text{Total input power}} \\
&= \frac{\text{Total load power request}}{(\text{unit 1 input power} + \text{unit 2 input power})} \\
&= \frac{\text{Total load power request}}{\left(\dfrac{\text{unit 1 output power}}{\text{efficiency}} + \dfrac{\text{unit 2 output power}}{\text{efficiency}}\right)} \\
&= \frac{P_{total}}{(x_1/\eta(x_1) + x_2/\eta(x_2))};
\end{aligned}$$

[0161] Where, P_{total} is the total load power request, x_1 and x_2 are the distributed load power for unit 1 and unit 2 respectively; η is the efficiency map, which is a curve representing efficiency vs. output power. Original efficiency data was obtained from a company, and the efficiency function q was obtained by curve fitting. For a given P_{total}, the above efficiency maximization problem is the same as minimizing the following cost function:

[0162] Minimize:

$$F=x_1/\eta(x_1)+x_2/\eta(x_2);$$

$$x_1+x_2=P_{total};$$

[0163] subject to:

$$0\leqq x_1\leqq 75\text{ kw};$$

$$0\leqq x_2\leqq 75\text{ kw}.$$

[0164] It is a nonlinear optimization problem with linear constraints. The SQP was used to solve this problem in Matlab. The next subsection shows the optimal efficiency solution.

[0165] 3.6.1.2. Optimal Solution Results for Two Identical 75 kw Units

[0166] The optimal solution results are depicted in FIG. **4**. The X-axis represents the total load power request, and the Y-axis represents system electrical efficiency. There are three power distribution methods underlying this graph. The first method is the optimal power distribution, which is the continuous line. The second method is the equal split power method, which is represented by the "+" curve, and the third method is the load following method, which is represented by the "o" curve. Load following in this case means one unit runs as the master unit and always takes the load power. When the master unit alone is not enough for the total load power request, then the second unit takes the remaining load power. This continues when there are more than two units. From the graph, we can see that the system efficiency of optimal power distribution has higher performance than that of equal split method or load following method when power is distributed between two 75 kw units.

[0167] We also can see that when the total load power request is less than or equal to 75 kw, the optimal efficiency is almost the same as load following mode. When the total load power request is greater than 75 kw, the optimal efficiency is almost the same as the equal split mode. FIG. **5** shows how optimal PMS distributes power between these two 75 kw units.

[0168] There is a pattern in the optimal solution. When the total load power request is less than or equal to 75 kw, only unit 1 runs and unit 2 does not run. When the total load power request is greater than 75 kw, the power is split equally between unit 1 and unit 2.

[0169] 3.6.2. Three Identical 75 kw Units

[0170] 3.6.2.1. Problem Formulation

[0171] The total system efficiency is derived as follows:

$$\text{System efficiency}=P_{total}/(x_1/\eta(x_1)+x_2/\eta(x_2)+x_3/\eta(x_3));$$

Where x_1, x_2, and x_3 are the power generation assignments for unit 1, unit 2, and unit 3 respectively. The above efficiency maximization problem for a given P_{tota1} is the same as minimizing the following cost function:

[0172] Minimize:

$$F=x_1/\eta(x_1)+x_2/\eta(x_2)+x_3/\eta(x_3);$$

$$x_1+x_2+x_3=P_{total};$$

[0173] Subject to:

$$0\leqq x_1\leqq 75\text{ kw};$$

$$0\leqq x_2\leqq 75\text{ kw};$$

$$0\leqq x_3\leqq 75\text{ kw};$$

This is a nonlinear optimization problem with linear constraints. The SQP was used to solve it in Matlab.

[0174] 3.6.2.2. Optimal Solution Result for Three Identical 75 kw Units

[0175] The optimal solution results are depicted in FIG. **6**. From the graph, we can see that the system efficiency of optimal power distribution has higher performance than that of the equal split method or the load following method for this configuration.

[0176] We also can see that when the total load power request is less than or equal to 75 kw, the optimal efficiency is almost the same as the load following mode. When the total load power request is greater than 150 kw, the optimal efficiency is almost the same as the equal split mode. FIG. **7** shows how optimal PMS distributes power among these three 75 kw units.

[0177] There is a logical pattern in the optimal solution. When the total load power request is in the interval of [0 75] kw, only unit 1 runs. When the total load power request is in the interval of (75 150] kw, the power is split equally between unit 1 and unit 2, and unit 3 does not run. When the total load power request is in the interval (150 225] kw, the power is split equally among three units, as in equal split mode. Two different units will be evaluated now. When identical units are not used, there is no obvious pattern in some regions.

[0178] 3.6.3. Two Different Units with One 75 kw Unit and One 150 kw Unit

[0179] 3.6.3.1. Problem Formulation

[0180] The total system efficiency is derived as follows:

$$\text{System efficiency}=P_{total}/x/\eta_1(x_1)+x_2/\eta_2(x_2));$$

Where, x_i is for 75 kw unit, x_2 is for 150 kw unit and η_2 is the efficiency map for 150 kw unit. In this configuration η_1 and η_2 are different functions.

[0181] The above efficiency maximization problem for a given P_{total} is the same as minimizing the following cost function:

[0182] Minimize:

$$F=x_1/\eta_1(x_1)+x_2/\eta_2(x_2);$$

[0183] Subject to:

$$x_1+x=P_{total};$$

$$0\leqq x_1\leqq 75\text{ kw};$$

$$0\leqq x_2\leqq 150\text{ kw}.$$

This is a nonlinear optimization problem with linear constraints. The SQP was used to solve the problem in Matlab.

[0184] 3.6.3.2. Optimal Solution for Two Different Units with One 75 Kw Unit and One 150 Kw Unit

[0185] The optimal solution results for this configuration are depicted in FIG. **8**. When there are different units, there is no equal split mode, and there are two different load following methods. One method uses the 75 kw unit as the master unit with a 150 kw unit following. The other method uses a 150 kw unit as the master unit with a 75 kw unit following. These two load following methods can result in different system efficiencies. From the graph, we can see that the system efficiency of optimal power distribution has higher performance than that of load following methods when power is distributed between two different units.

[0186] We also can see that when the total load power request is in the interval of [0 75] kw, the optimal efficiency is almost the same as the first load following mode (75 kw unit as master). When the total load power request is in the interval of (75 150] kw, the optimal efficiency is almost the same as the second load following mode (150 kw unit as master). When the total load power request is in the interval of (150 225] kw, the optimal efficiency is different from either load following mode. FIG. **9** shows how optimal PMS distributes power between these two different units.

[0187] There is a pattern in the optimal solution. When the total load power request is in the interval of [0 75] kw, only the 75 kw unit (unit 1) runs. When the total load power request is in the interval of (75 150] kw, only the 150 kw unit runs, and the 75 kw unit does not run. When the total load power request is in the interval of (150 225] kw, then the total load power request is distributed between the 75 kw unit and the 150 kw unit. There is no obvious pattern for P_{total}>150 kw. Because of that, a fuzzy logic controller for online implementation was used in this region.

[0188] 3.6.4. Three Different Units with Two 75 kw Units and One 150 kw Unit

[0189] 3.6.4.1. Problem Formulation

[0190] The total system efficiency is derived as follows:

$$\text{System efficiency}=P_{total}/(x_1/\eta_1(x_1)+x_2/\eta_1(x_2)+x_3/\eta_2(x_3));$$

Where, x_1, x_2, and x_3 are for one 75 kw unit, the other 75 kw unit, and a 150 kw unit respectively. The above efficiency maximization problem for a given P_{total} is the same as minimizing the following cost function:

[0191] Minimize

$$F=(x_1/\eta_1(x_1)+x_2/\eta_1(x_2)+x_3/\eta_2(x_3);$$

[0192] Subject to:

$$x_1+x_2+x_3=P_{total};$$

$$0\leqq x_1\leqq 75\text{ kw};$$

$$0\leqq x_2\leqq 75\text{ kw};$$

$$0\leqq x_3\leqq 150\text{ kw}.$$

[0193] This is a nonlinear optimization problem with linear constraints. The SQP was used to solve it in Matlab.

[0194] The optimal solution results are depicted in FIG. **10**. There are three different load following methods for this configuration. One method is S-S-B; "S" indicates the small unit, 75 kw, and "B" indicates the big unit, 150 kw. For S-S-B, 75 kw unit (unit 1) was the master, and the other 75 kw unit (unit 2) follows as the second unit with the 150 kw unit (unit 3) following as the last unit. The other two methods are S-B-S and B-S-S. These three different load following methods can result in different system efficiencies. From the graph, it can be seen that the system efficiency of optimal power distribution has higher performance than that of the three load following methods when load power is distributed among these three different units.

[0195] 3.6.4.2. Optimal Solution Result

[0196] It also can be seen from FIG. **10** that when the total load power request is less than or equal to 75 kw, the optimal system efficiency is almost the same as the system efficiency of the S-S-B and S-B-S load following methods. When the total load power request is in the interval of (75 300] kw, the optimal system efficiency is better than the system efficiency of all load following modes. FIG. **11** shows how optimal PMS distributes power among these three different units.

[0197] There is a logical pattern in the optimal solution. When the total load power request is in the interval of [0 75] kw, only the 75 kw unit (unit 1) runs. When the total load power request is in the interval of (75 150] kw, two 75 kw units (unit 1 and unit 2) split the total load power equally and the 150 kw unit (unit 3) does not run. When the total load power request is in the interval of (150 225] kw, one 75 kw unit (unit 1) does not run. The load power is distributed between the other 75 kw unit (unit 2) and the 150 kw unit (unit 3). There is no obvious pattern in this case. Because of that, a fuzzy logic controller for online implementation was used in this region. When the total load power request is in the interval of (225 300] kw, the load power is distributed between two 75 kw units and one 150 kw units. There is no obvious pattern in this case except that the two 75 kw units run at the same load power. Because of that, a fuzzy logic controller for online implementation was used in this region to distribute power between two 75 kw units and one 150 kw unit, and a mathematical rule is used at the same time to allow the two 75 kw units to provide identical output.

[0198] 3.6.5. Three Different Units with One 75 Kw Unit and Two 150 Kw Units

[0199] 3.6.5.1. Problem Formulation

[0200] The total system efficiency for this configuration is derived as follows:

$$\text{System efficiency}=P_{total}/(x_1/\eta_1(x_1)+x_2/\eta_1(x_2)+x_3/\eta_2(x_3));$$

where x_i is for the 75 kw unit, both x_2 and x_3 are for the 150 kw units. For a given P_{total}, the above efficiency maximization problem is the same as minimizing the following cost function:

[0201] Minimize:

$$F=(x_1/\eta_1(x_1)+x_2/\eta_2(x_2)+x_3/\eta_2(x_3));$$

$$x_1+x_2+x_3=P_{total};$$

[0202] Subject to:

$$0\leqq x_1\leqq 75\text{ kw};$$

$$0\leqq x_2\leqq 150\text{ kw};$$

$$0\leqq x_3\leqq 150\text{ kw}.$$

[0203] This is a nonlinear optimization problem with linear constraints. The SQP was used to solve the problem in Matlab.

[0204] 3.6.5.2. Optimal Solution Results

[0205] The optimal solution results for this configuration are depicted in FIG. **12**. There are three different load following methods for this configuration. For the S-B-B method, 75 kw unit (unit 1) was master and the 150 kw unit (unit 2) follows as the second with 150 kw unit (unit 3) following as the last unit. The other two methods are B-B-S and B-S-B. These three different load following methods can result in different system efficiencies. From the graph, it can be seen that the system efficiency of optimal power distribution has higher performance than that of three load following methods when load power is distributed among these three different units.

[0206] It also can be seen that when the total load power request is less than or equal to 75 kw, the optimal system efficiency is almost the same as the system efficiency of the SB-B load following method. When the total load power request is in the interval of (75 150) kw, the optimal system efficiency is the same as the system efficiency of B-S-B and B-B-S load following methods. When the total load power request is in the interval of (150 375] kw, the optimal system efficiency is better than that of all load following methods. FIG. **13** shows how optimal PMS distributes power among these three different units.

[0207] There is also a pattern in the optimal solution. When the total load power request is in the interval of [0 75] kw, only the 75 kw unit (unit 1) runs. When the total load power request is in the interval of (75 150] kw, only one of the 150 kw units (unit 3) runs and the other units do not run. When the total load power request is in the interval of (150 225] kw, one of the 150 kw units (unit 2) does not run and the load power is distributed between the 75 kw unit (unit 1) and another 150 kw unit (unit 3). There is no obvious pattern in this case. Because of that, the fuzzy logic controller illustrated in FIG. **14** was used in this region for online implementation. When the total load power request is in the interval of (225 300] kw, the 75 kw unit (unit 1) does not run, and the two 150 kw units (unit 2 and unit 3) equally split the total load power. When the total load power request is in the interval of (300 375] kw, the load power is distributed between one 75 kw unit and two 150 kw units, and there is no obvious pattern in this case except that the two 150 kw units run at the same load power. Because of that, a fuzzy logic controller for online implementation was used in this region to distribute power between one 75 kw unit and two 150 kw units, and a mathematical rule is used at the same time to allow the two 150 kw units to provide identical output.

[0208] 3.7. Result Analysis and Comparison Between Different Configurations

[0209] 3.7.1. Optimal Results Analysis

[0210] The simulation results of the above five configurations demonstrated that the proposed optimal PMS improved the system efficiency significantly compared to the current rule-based PMS. The next step is to implement the optimal PMS algorithm in real time.

[0211] 3.7.2. Optimal PMS Implementation

[0212] The optimal PMS have several drawbacks:

[0213] 1. The SQP algorithm has a complex structure; therefore, more computing power would make the PMS online implementation more feasible.

[0214] 2. The optimal PMS design is not robust enough against the environmental variations and external disturbances. The efficiency map used in the optimization algorithm was based on a specific temperature and altitude. Therefore, more efficiency and robust PMS should be developed to distribute power among multiple DG units.

[0215] In this Part, two methods are developed to implement optimal PMS. They are rule-based PMS and hybrid fuzzy rule-based (rule-based and fuzzy rule-based) PMS. The hybrid fuzzy rule-based PMS combined both crisp logic and fuzzy logic. Table 3.2 shows the control rules summarized from the statistical analysis of the optimal solution and the implementation methods for each configuration.

TABLE 3.2

Comparison of the optimal power distributions and their implementations for five different configurations

		Control rules summarized from the statistical analysis of the optimal solutions				
#	Configuration	P_{total} (kw)	Unit 1	Unit 2	Unit 3	Implementation
1	Two identical 75 kw units	[0 75]	P_{total}	0	N/A	Rule-based
	Unit 1-75 kw	(75 150]	/2	/2	N/A	
	Unit 2-75 kw					
2	Three identical 75 kw units	[0 75]		0	0	Rule-based
	Unit 1-75 kw	(75 150]	/2	/2	0	
	Unit 2-75 kw	(150 225]	/3	/3	/3	
	Unit 3-75 kw					
3	Two different units with	[0 75]		0	N/A	Hybrid fuzzy
	one 75 kw unit and one	(75 150]	0		N/A	rule-based
	150 kw unit	(150 225]	x1	−x1	N/A	
	Unit 1-75 kw					
	Unit 2-150 kw					
4	Three different units with	[0 75]		0	0	Hybrid fuzzy
	two 75 kw units and one	(75 150]	/2	/2	0	rule-based
	150 kw unit	(150 225]	0	x2	−x2	
	Unit 1-75 kw	(225 300]	(−x3)/2	(−x3)/2	x3	
	Unit 2-75 kw					
	Unit 3-150 kw					
5	Three different units with	[0 75]		0	0	Hybrid fuzzy
	one 75 kw unit and two	(75 150]	0	0		rule-based
	150 kw units	(150 225]	x1	−x1	0	
	Unit 1-75 kw	(225 300]	0	/2	/2	
	Unit 2-150 kw	(300 375]	x1	(−x1)/2	(−x1)/2	
	Unit 3-150 kw					

[0216] The rule-based PMS is constructed according to the math-optimal result; therefore, the resulting rule-based PMS is exactly the same as the math-optimal PMS. However, the rule-based PMS can only be used for simple optimization rules; otherwise it will be difficult to find exact rules and also will be very time consuming. The fuzzy PMS is not as precise as the rule-based PMS in this case; however, it approximates the math-optimal PMS and is convenient and fast for online implementation.

[0217] Based on the analysis of the optimal solution, the rule-based PMS was developed for identical units because there are some obvious patterns in their optimal solutions. And the hybrid fuzzy rule-based PMS were developed for different units because in some regions there are no obvious patterns for different units to split load power. Therefore, fuzzy logic was used in those regions for online implementation.

[0218] The implementation result will now be compared with optimal solution and those obtained from current PMS.

4. Implementation of Optimal Power Management System

[0219] 4.1. Optimal Power Distribution Implementation

[0220] In Chapter 3, we used SQP to statically optimize the power distribution. The remaining problem is how to implement the PMS online. There are several methods to implement the optimal power distribution online. According to the optimal results, rule-based and fuzzy rule-based implementations are chosen in this Part. The rule-based system is simple and easy to implement. Generally speaking, it is intuitive, but difficult to obtain exact rules, and they apply well only to certain simple systems. Fuzzy rule-based systems, generally speaking, are practical, robust, and easy to implement online, but it is difficult to obtain expert data from which to design fuzzy rules. In this Part, fuzzy rules will be designed according to the optimal results that were obtained in Chapter 3.

[0221] 4.1.1. Rule-based and Fuzzy Rule-based Implementation for Power Distribution

[0222] According to the optimization results, when identical units were used, certain patterns were apparent, and rules can be derived based on these results. The rule-based PMS configurations were thus designed for implementation. However, when different units are used, rule-based PMS is not so easily applied for different ranges of P total in certain regions. The optimal solutions can be divided into several regions. In some regions, the rule-based implementation was designed. In other regions, fuzzy logic was used to implement the power distribution. This hybrid power distribution was used for different units. The power distribution uses crisp logic and a fuzzy logic controller at the same time.

[0223] 4.1.2. Introduction to Fuzzy Logic Controller

[0224] 4.1.2.1. Fuzzy Logic System and Fuzzy Logic Controller Scheme

[0225] Fuzzy logic provides a general description and measurement of a system similar human reasoning. In fuzzy logic, the truth of any statement is a matter of degree. A membership function is the curve that describes how true a given statement is for a given input value. Fuzzy logic does not use crisp values to describe a system; instead, human reasoning is used. For example, when describing the temperature, crisp values were not used. Instead, we use a "high", "medium", "low", etc. fuzzy logic set. Different temperature values may belong to different fuzzy logic sets with a certain percentage membership. Fuzzy set membership occurs by degree over the range [0, 1]. For example, 100° F. might belong to a set of "high" temperatures, and its membership to the "high" fuzzy set is 0.7, and it may also belong to "medium" at the same time with a membership of 0.2.

[0226] Fuzzy logic systems can be used to make decisions or control systems by encoding human reasoning. Therefore a fuzzy logic system can easily be understood and built based on human understanding. Fuzzy logic comprises fuzzy sets, fuzzy set membership functions, fuzzy rules, and fuzzy inference. Fuzzy sets represent non-statistical uncertainty. A membership function is used to represent the fuzzy set membership. Triangle, trapezoid, Gaussian and sigmoid functions are commonly used as fuzzy set membership functions. Fuzzy rules approximate human reasoning, which used to make inferences in fuzzy logic. All the fuzzy rules in a fuzzy system are applied in parallel. The design procedure of the fuzzy logic controller in this Part works as follows.

[0227] Define input and output variables of a fuzzy logic controller.

[0228] Fuzzify the input and output variables. Define fuzzy sets and fuzzy membership functions for each fuzzy variable.

[0229] Determine fuzzy rule set.

[0230] Defuzzify the fuzzy outputs to obtain the crisp values for each output variable.

[0231] In this Part, the input signal to the fuzzy controller is the total load power request. The output signal is the load power to each single unit (75 kw or 150 kw unit). The diagram below shows the fuzzy logic controller scheme.

[0232] 4.1.2.2. Design Fuzzy System Using Matlab/Fuzzy Logic Toolbox

[0233] In this work, the fuzzy logic toolbox in Matlab was used. Its graphic user interface (GUI) is easy to use to design fuzzy inference system (FIS) structure, fuzzy sets, fuzzy set membership functions, and fuzzy rule set. As long as the fuzzy system was properly designed, the fuzzy system can be represented as a FIS file in the fuzzy toolbox. When designing a fuzzy controller in Matlab/Simulink, the FIS file name must be provided in the fuzzy controller block parameter window, and when running the Simulink model, Simulink recalls the fuzzy system.

[0234] 4.2. Rule-based and Hybrid Fuzzy Rule-based Implementation for Power Distribution in Matlab/Simulink for Five Different Scenarios

[0235] According to the optimal power distribution data, rule-based and fuzzy rule-based power distribution implementations are designed in a Matlab/Simulink environment. Simulation results also will be discussed in this section. In this section, the design of the rule-based and fuzzy rule-based power distribution Simulink models of each configuration will be discussed.

[0236] The models for every configuration are built according to the optimal power distribution results. The distributed load power signals were sent to each unit through multiport switches. Inside each unit there is a subsystem used to calculate input power and fuel consumption. Finally, the system electrical efficiency was calculated.

[0237] 4.3. Two Identical 75 Kw Units

[0238] FIG. **15** shows the Simulink model of the rule-based power distribution implementation for two 75 kw units. Conventional logic is used in this model to distribute power between two 75 kw units (unit 1 and unit 2). FIG. **16** shows the

IF and ELSE subsystems, which are built according to the rules obtained from the analysis of optimal power distribution data.

[0239] 4.4. Three Identical 75 kw Units

[0240] FIG. **17** shows the Simulink model of the rule-based power distribution implementation for three 75 kw units. Crisp logic is used in this model to distribute power among three 75 kw units (unit 1, unit 2, and unit 3). FIG. **18** shows the IF, ELSEIF, and ELSE subsystems, which are built according to the rules obtained from the analysis of optimal power distribution data.

[0241] 4.5. Two Different Units with One 75 Kw and One 150 Kw Unit

[0242] FIG. **19** shows the rule-based and fuzzy rule-based power distribution implementation for two different units. Both crisp logic and fuzzy logic are used in this model to distribute power between a 75 kw unit (unit 1) and a 150 kw unit (unit 2).

[0243] FIG. **20** shows the IF, ELSEIF, and ELSE subsystems, which are built according to the rules obtained from the analysis of optimal power distribution data. Both IF and ELSE subsystems are rule-based subsystems, but the ELSEIF subsystem is a fuzzy rule-based system. Inside this block is a fuzzy controller. The inputs of the fuzzy controller are the total load power request and heat request. The output signal is the load power for the 75 kw unit (unit 1). The load power for the 150 kw unit (unit 2) is equal to the total load power minus the load power of the 75 kw unit (unit 1). The fuzzy controller in this embodiment can only have an output for either the 75 kw unit (unit 1) or the 150 kw unit (unit 2), and cannot have outputs for each of these two units. If there are two outputs for the fuzzy controller, and when these two outputs are added together, the sum may not equal the total load power request. Because the controller used here applies fuzzy logic, the fuzzy controller may not provide the output values whose sum is exactly equal to the input total load power value.

[0244] 4.5.1. Fuzzy Logic Controller Design for ELSE Subsystem

[0245] 4.5.1.1. FIS Structure

[0246] When designing a fuzzy system, first define the inputs and outputs and their membership functions. Then construct fuzzy rules. The FIS structure represented by the fuzzy toolbox GUI is shown in FIG. **21**.

[0247] There are one input, one output, and three rules in this FIS structure. Each input and output signal of the fuzzy system is a fuzzy variable, which has several fuzzy sets and their own fuzzy membership functions. The output signal in this case is the load power to the 75 kw unit.

[0248] 4.5.1.2. Fuzzy Sets and Fuzzy Membership Functions

[0249] The fuzzy controllers in this Part have one input and one output as described in the fuzzy controller scheme. Each input and output has a different number of fuzzy sets; for example, for two different units, the fuzzy sets for input and output are:

[0250] Total load power request: Three fuzzy sets—H (high), M (medium), L (low)

[0251] Unit 1 load power: Four fuzzy sets—H (high), M (medium), OK, L (low)

[0252] In fuzzy logic, a value can partially belong to a fuzzy set with corresponding membership between 0 and 1. In this Part, trapezoid functions (include left-trapezoid function and right-trapezoid function) and triangle functions are selected as the membership functions. The entire rule set is applied in parallel, and fuzzification and defuzzification methods are used to obtain the output signal from the inputs. FIG. **22** shows the fuzzy sets and membership functions for the configuration of two different units.

[0253] 4.5.1.3. Fuzzy Rule Set

[0254] The fuzzy rule set includes a set of fuzzy if-then rules, which define the fuzzy inference system from the input to output based on the optimal data.

[0255] The fuzzy control rule set shown in Table 4.1 is built based on human analysis of the optimal solution in Chapter 3.

TABLE 4.1

Fuzzy rule set
1. If (Total_Required_Power is L) then (Output_Power_1 is OK) (1)
2. If (Total_Required_Power is M) then (Output_Power_1 is M) (1)
3. If (Total_Required_Power is H) then (Output_Power_1 is H) (1)

[0256] 4.5.1.4. Representation of Fuzzy System (FIS File)

[0257] In Matlab/fuzzy logic toolbox, the following method is used to represent a fuzzy system as shown in Table 4.2. This is also the FIS file.

TABLE 4.2

The representation of a fuzzy system in Matlab

```
[System]
Name='PMS_5_2DIFF'
Type='mamdani'
Version=2.0
NumInputs=1
NumOutputs=1
NumRules=3
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='centroid'
[Input1]
Name='Total_Required_Power'
Range=[150 225]
NumMFs=3
MF1='L':'trapmf',[150.9 154.5 181.8 186]
MF2='M':'trapmf',[174.5 182.7 209.1 215.2]
MF3='H':'trapmf',[210 214.6 226.4 239 1]
[Output]
Name='Output_Power_1'
Range=[0 75]
NumMFs=4
MF1='L':'trapmf',[0.5036 17.78 38.14 52.69]
MF2='OK':'trapmf',[44.55 50.5 60.45 66.82]
MF3='H':'trapmf',[71.49 72.67 77.49 80.22]
MF4='M':'trapmf',[61.94 66.12 72.12 75.12]
[Rules]
1, 2 (1) : 1
2, 4 (1) : 1
3, 3 (1) : 1
```

[0258] The first section shows general information about the system, which includes number of inputs, outputs, and rules. The first section also includes fuzzification and defuzzification methods. As the above table shows, there are one input, one output, and three rules. The remaining sections show the information about each input and output fuzzy variable, which include the name, fuzzy sets and their membership functions, and fuzzy rules of each input and output variable. For the membership functions, information about the name, type, and range of each membership function is indicated. For the fuzzy rule set, the above FIS file lists three rules. Each line was encoded from a rule provided in Table 4.1. The first integer represents the input and the second integer represents the output. "1" means membership function 1, which is "L" (low), and "2" means membership function 2, which is "M" (medium) and so on. The number in parentheses is the weight of the rule. The last number is the rule connection. For example, the first line shows: "1, 2 (1): 1," which line represents the first rule listed in Table 4.1. This rule is as follows:

[0259] If (Total_Required_Power is L) then (Output Power 1 is OK) (weight 1)

[0260] 4.6. Three Different Units with Two 75 Kw Units and One 150 Kw Unit

[0261] FIG. **23** shows the hybrid fuzzy rule-based power distribution implementation for three different units with two 75 kw units and one 150 kw unit. Both crisp logic and fuzzy logic are used in this model to distribute power among two 75 kw units (unit 1 and unit 2) and a 150 kw unit (unit 3).

[0262] FIG. **24** shows the IF, ELSELF, and ELSE subsystems, which are built according to the rules in Table 4.3. For the ELSEIF2 subsystem, the output signal of the fuzzy controller is the load power for the 75 kw unit (unit 1). The load power for the 150 kw unit (unit 3) is equal to the total load power request minus the load power of the 75 kw unit (unit 1). The other 75 kw unit (unit 2) does not run. For the ELSE subsystem, the output signal of the fuzzy controller is the load power for the 150 kw unit (unit 3). The two 75 kw units (unit 1 and unit 2) equally share the remaining load power.

[0263] The fuzzy logic controller in the ELSEIF2 subsystem in FIG. **24** is the same controller as used for two different units, but the fuzzy logic controller in the ELSE subsystem is different.

[0264] 4.6.1. Fuzzy Logic Controller Design for the ELSE Subsystem

[0265] 4.6.1.1. FIS Structure

[0266] There are one input, one output, and four rules in the FIS structure illustrated in FIG. **25**. The output signal in this case is the load power to the 150 kw unit (unit 3). The output is different from that of the previous fuzzy controller.

[0267] 4.6.1.2. Fuzzy Sets and Fuzzy Membership Functions

[0268] The fuzzy sets for input and output are as illustrated in FIG. **26**:

[0269] Total load power request: four fuzzy sets—H (high), M (medium), OK, L (low)

[0270] Unit 3 load power: four fuzzy sets—H (high), M (medium), OK, L (low)

[0271] 4.6.1.3. Fuzzy Rule Set

[0272] The fuzzy rules for this fuzzy logic controller are shown in Table 4.3.

TABLE 4.3

Fuzzy rule set
1. If (Total__Required__Power is L) then (unit__3 is L) (1)
2. If (Total__Required__Power is M) then (unit__3 is M) (1)
3. If (Total__Required__Power is H) then (unit__3 is H) (1)
4. If (Total__Required__Power is OK) then (unit__3 is OK) (1)

[0273] 4.7. Three Different Units with One 75 kw Unit and Two 150 kw Units

[0274] FIG. **27** shows the hybrid fuzzy rule-based power distribution implementation for three different units with one 75 kw unit and two 150 kw units. Both crisp logic and fuzzy logic are used in this model to distribute power among a 75 kw unit (unit 1) and two 150 kw units (unit 2 and unit 3).

[0275] FIG. **28** shows the IF, ELSEIF, and ELSE subsystems, which are built according to the rule obtained from the analysis of optimal power distribution data. For the ELSEIF2 subsystem, the load power for the 150 kw unit (unit 2) is equal to the total load power minus the load power of the 75 kw unit (unit 1). The other 150 kw unit (unit 3) does not run. For the ELSE subsystem, both 150 kw units (unit 2 and unit 3) share the remaining load power request equally.

[0276] The fuzzy logic controller in the ELSEIF2 subsystem is the same controller as used for two different units, but the fuzzy logic controller in the ELSE subsystem is different from previous fuzzy controllers. The fuzzy sets, membership functions, and rule set are totally different.

[0277] 4.7.1. Fuzzy Logic Controller Design for the ELSE Subsystem

[0278] 4.7.1.1. FIS Structure

[0279] FIG. **29** shows the FIS structure of the fuzzy controller. There are one input, one output, and 5 rules in this FIS structure. Output signal in this case is the load power request to the 75 kw unit (unit 1).

[0280] 4.7.1.2. Fuzzy Sets and Fuzzy Membership Functions

[0281] The fuzzy sets for inputs and outputs are as illustrated in FIG. **30**:

[0282] Total load power request: five fuzzy sets—VH (very high), H (high), M (medium), OK, L (low)

[0283] Unit 1 load power: five fuzzy sets—VH (very high), H (high), M (medium), OK, L (low)

[0284] 4.7.1.3. Fuzzy Rule Set

[0285] The fuzzy rules for this fuzzy logic controller are shown in Table 4.4.

TABLE 4.4

Fuzzy rule set
1. If (Total__Required__Power is L) then (unit__1 is L) (1)
2. If (Total__Required__Power is H) than (unit__1 is H) (1)
3. If (Total__Required__Power is M) than (unit__1 is M) (1)
4. If (Total__Required__Power is OK) than (unit__1 is OK) (1)
5. If (Total__Required__Power is VH) then (unit__1 is VH) (1)

[0286] 4.8. Simulation Results of Rule-based and Hybrid Fuzzy Rule-based Implementations for Power Distribution with Five Different Configurations.

[0287] 4.8.1. Two Identical 75 kw Units

[0288] For two identical 75 kw units, FIG. **31** shows the rule-based power distribution implementation design using Matlab/Simulink according to the optimal power distribution result FIG. **31** shows how the PMS splits power between two units.

[0289] The first row shows the total load power request, which increases with time as a ramp signal. The second row shows the load power distributed to a 75 kw unit (unit 1). The third row shows the load power distributed to the other 75 kw unit (unit 2). It can be seen that when the total load power request is less than 75 kw, only unit 1 runs and unit 2 does not run. When the total load power request is greater than 75 kw, unit 1 and unit 2 split the load power equally.

[0290] FIG. **32** shows the total electrical efficiency by using this rule-based power management system. The first row shows the total load power request, which increases with time as a ramp signal. The second row shows the total system electrical efficiency. It can be seen that when the total load power request reaches 75 kw, unit 1 ran full load, and the system electrical efficiency achieved its highest point. It is approximately 31%. When the total load power request is 150 kw, both units run full load, and the system electrical efficiency also achieve its highest point.

[0291] This system electrical efficiency is satisfactory if compared with the optimal efficiency. This can be seen from FIG. **33**. The continuous line indicates the system total efficiency of rule-based implementation for two 75 kw units. The "o" curve indicates the system electrical efficiency of optimal power distribution. It can be seen that the system electrical efficiency of the rule-based implementation is almost exactly the same as that of the optimal power distribution. It is noted that the optimal efficiency is slightly lower than that obtained in the rule-based implementation at a few points due to numerical round off error.

[0292] 4.8.2. Three Identical 75 kw Units

[0293] For three identical 75 kw units, FIG. **35** shows the rule-based performance in a static simulation.

[0294] FIG. **34** shows how the PMS splits power among three 75 kw units.

[0295] It can be seen that when the total load power request is less than 75 kw, only unit 1 runs, and neither unit 2 nor unit 3 run. When the total load power request is greater than 75 kw and less than 150 kw, unit 1 and unit 2 split the load power equally. When the total load power request is greater than 150 kw, three units split power equally.

[0296] FIG. **35** shows the total electrical efficiency by using this rule-based power management system. It can be seen that when the total load power request reaches 75 kw, unit 1 ran full load, and the total electrical efficiency achieve its highest point. It is approximately 31%. When the total load power request reaches 150 kw, both unit 1 and unit 2 ran full load, and the system electrical efficiency also achieve its highest point. When the total load power request reaches 225 kw, three units ran full load and the system electrical efficiency also achieve its highest point.

[0297] This system electrical efficiency is satisfactory if compared with the optimal efficiency. This can be seen from FIG. **36**.

[0298] It can be seen that the system electrical efficiency of rule-based implementation is almost the same as that of the optimal power distribution. This is the power distribution for three 75 kw units. Power distribution for two different units with one 75 kw unit and one 150 kw unit will now be evaluated.

[0299] 4.8.3. Two Different Units with One 75 kw Unit and One 150 kw Unit

[0300] For two different units, hybrid fuzzy node-based implementation was used for power distribution in Matlab/Simulink according to the optimal power distribution results. FIG. **37** shows how the PMS split power between two different units.

[0301] It can be seen from the graph that when the total load power request is less than 75 kw, only the 75 kw unit (unit 1) runs and the 150 kw unit (unit 2) does not run. When the total load power request is greater than 75 kw and less than 150 kw, only the 150 kw unit (unit 2) runs and the 75 kw unit (unit 1) does not run. When the total load power request is greater than 150 kw, the 75 kw unit (unit 1) and the 150 kw unit (unit 2) split the load power corresponding to the fuzzy rules designed to approximate the optimal power distribution. FIG. **38** shows the total electrical efficiency by using this hybrid fuzzy rule-based PMS.

[0302] It can be seen that when the total load power request reaches 75 kw, the 75 kw unit (unit 1) ran full load, and the total electrical efficiency achieve its highest point. It is approximately 31%. When the total load power request reaches 150 kw, the 150 kw unit (unit 2) runs full load, and the system electrical efficiency achieve its third highest point. When the total load power request reaches 225 kw, both the 75 kw unit (unit 1) and the 150 kw unit (unit 2) run full load, and the system electrical efficiency achieve its second highest point. Identical units can find two or three identical highest efficiency points. This is not the case for different units because different units have different efficiency maps, and their maximum efficiency and the change of efficiency are different.

[0303] This system efficiency is satisfactory if compared with the optimal efficiency. This can be seen from FIG. **39** in the following page.

[0304] It can be seen that the system electrical efficiency of hybrid fuzzy rule-based implementation is almost exactly the same as that of optimal power distribution. This is the power distribution for one 75 kw unit and one 150 kw unit. The power distribution for three different units with two 75 kw units (unit 1 and unit 2) and one 150 kw unit (unit 3) will now be evaluated.

[0305] 4.8.4. Three Different Units with Two 75 kw Units and One 150 kw Unit

[0306] For three different units, hybrid fuzzy rule-based implementation is used for power distribution in Matlab/Simulink according to the optimal power distribution results. FIG. **40** shows how the power management system split power among three different units with two 75 kw units and one 150 kw unit.

[0307] It can be seen that when the total load power request is less than 75 kw, only the 75 kw unit (unit 1) runs, and the other 75 kw unit (unit 2) and the 150 kw unit (unit 3) do not run. When the total load power request is greater than 75 kw and less than 150 kw, the two 75 kw units (unit 1 and unit 2) run the same load power and the 150 kw unit (unit 3) does not run. When the total load power request is greater than 150 kw and less than 225 kw, one 75 kw unit (unit 1) does not run, the other 75 kw unit (unit 2) and a 150 kw unit (unit 3) split the total load corresponding to the fuzzy rules designed to approximate the optimal power distribution. When the total load power request is greater than 225 kw, two 75 kw units run identical load power amount, and the power of the 150 kw unit corresponds to the fuzzy rules designed to approximate the optimal power distribution. FIG. **41** shows the total electrical efficiency by using this hybrid fuzzy rule-based power management system.

[0308] It can be seen that when the total load power request reaches 75 kw, the 75 kw unit (unit 1) ran full load, and the system electrical efficiency achieve its highest point. It is approximately 31%. When the total load power request reaches 150 kw, both 75 kw units (unit 1 and unit 2) run full load, and the system electrical efficiency also achieve its highest point. When the total load power request reaches 225 kw, the 75 kw unit (unit 1) does not run, the other 75 kw unit (unit 2) and a 150 kw unit (unit 3) run full load, and the system electrical efficiency achieve its third highest point. When the total load power request reaches 300 kw, all units run full load and the system electrical efficiency achieve its second highest point. For three identical units, there are three identical highest efficiency points as seen in FIG. **36**. This is not the case for different units because different units have different efficiency maps, and different units have different maximum efficiency and different change of efficiency.

[0309] This system efficiency curve is satisfactory if compared with the optimal efficiency. This can be seen from FIG. **42**.

[0310] From the graph, it can be seen that the hybrid fuzzy rule-based implementation approximates the optimal power distribution. This is the power distribution for two 75 kw units and one 150 kw unit. The power distribution for another combination, one 75 kw unit and two 150 kw units, will now be evaluated.

[0311] 4.8.5. Three Different Units with One 75 kw Unit and Two 150 kw Units

[0312] FIG. **43** shows how the hybrid fuzzy rule-based power management system distributes power among three different units with one 75 kw unit (unit 1) and two 150 kw units (unit 2 and unit 3). All crisp rules and fuzzy rules used here are based on the optimal power distribution data.

[0313] From the graph, it can be seen that when the total load power request is less than 75 kw, only the 75 kw unit (unit 1) runs, and neither 150 kw unit (unit 2 and unit 3) runs. When the total load power request is greater than 75 kw and less than 150 kw, only one of the 150 kw units (unit 3) runs, and neither the 75 kw unit (unit 1) nor the other 150 kw unit (unit 2) run. When the total load power request is greater than 1501 cw and less than 225 kw, one of the 150 kw units (unit 3) does not run. The 75 kw unit (unit 1) and the other 150 kw unit (unit 2) split the total load corresponding to the fuzzy rules designed to approximate the optimal power distribution. When the total load power request is greater than 225 kw and less than 300 kw, the 75 kw unit (unit 1) does not run, and two 150 kw units (unit 2 and unit 3) equally split the total load power. When the total load power request is greater than 300 kw, two 150 kw units run identical load power amount; the load power is split between one 75 kw unit (unit 1) and the two 150 kw units (unit 2 and unit 3) corresponding to the fuzzy rules designed to approximate the optimal power distribution. FIG. **44** shows the total electrical efficiency by using this hybrid fuzzy rule-based power management system.

[0314] It can be seen that when the total load power request reaches 75 kw as shown in FIG. **44**, the 75 kw unit (unit 1) ran full load, and the total electrical achieve its highest point. It is approximately 31%. When the total load power request reaches 150 kw, one of the 150 kw units (unit 3) runs full load and the system electrical efficiency achieve its forth highest point. When the total load power request reaches 225 kw, the 75 kw unit (unit 1) and a 150 kw unit (unit 2) ran full load, and the system electrical efficiency achieve its second highest point. When total load power request reaches 300 kw, two 150 kw units (unit 2 and unit 3) run full load and the system electrical efficiency achieve its forth highest point, which is the same system electrical efficiency as when the total load power request is 150 kw. When the total load power request reaches 375 kw, all units run full load and the system electrical efficiency achieve its third highest point.

[0315] The efficiency curve of the hybrid fuzzy rule-based system for one 75 kw unit and two 150 kw units also is satisfactory if compared with the optimal efficiency. It can be seen from FIG. **45**.

[0316] From FIG. **45**, it can be seen that the hybrid fuzzy rule-based implementation approximates the optimal power distribution.

[0317] 4.9. Analysis of Results

[0318] The simulation results of the above five configurations in this chapter demonstrate that the performance of the proposed optimal rule-based and hybrid fuzzy rule-based PMS improve the system efficiency significantly compared to the rule-based PMS. That is currently used by our industry partner. These proposed PMS approximate the optimal results obtained by using the SQP method as described in Chapter 3. They are also easy to implement online.

5. Conclusion and Recommendations

[0319] 5.1. Conclusion

[0320] In this Part, optimal power management strategies were designed for DG system with five configurations. First, SQP was used to find optimal power distribution for each configuration offline. Then the optimal system efficiencies were compared with that of existing power distribution methods. The simulation results shows that the optimal power distribution is much better than the existing power distribution methods. It can improve the system efficiency significantly. However, because of the time typically needed to calculate the optimal solution online by using mathematic methods such as SQP, rule-based and hybrid fuzzy rule-based systems were used to implement the optimal PMS according to the summary of the optimal data. The optimal solution results of some configurations have obvious patterns, around which rule-based PMS were designed. Some other configurations do not have obvious patterns in certain regions. In those cases hybrid fuzzy rule-based PMS were designed. The proposed rule-based and hybrid fuzzy rule-based PMS approximate the optimal power distribution. It is also easy to implement online.

[0321] 5.2. Recommendations

[0322] GA can also be used to optimize the fuzzy systems. Alternatively, other evolutionary techniques such as particle swarm optimization (PSO) also can be used. Those evolutionary techniques can be used to tune fuzzy system parameters such as fuzzy membership functions and fuzzy rules.

[0323] The fuzzy system is powerful at exploring complex problems because it can be understood and built easily by engineers according to their experience since a fuzzy system encodes human reasoning. However, fuzzy logic lacks precision. Additionally, when the number of variables of a fuzzy system increases, the possible number of rules increases exponentially, and it will be difficult to design a complete rule set to achieve optimal system performance.

[0324] In the future, for the optimization part as described in Chapter 3, a DP also can be used to optimize the power distribution. DP is a global optimization method, and it can find the global optimizer directly. Additionally, engine start up cost can be taken into consideration when define the system electrical efficiency cost function. For the implementation part as described in Chapter 4, the hybrid fuzzy rule-based systems still can be used to design PMS. However, look up tables also can be used to design the PMS if the environment is stable. The look up table is easy to implement online, though it is not robust against disturbance. In addition, the output of a look up table is generally more precise than the output of a fuzzy system if there is sufficient optimal data in the look up table.

[0325] No matter whether look up table or hybrid fuzzy rule-based PMS is used, in the future, it is better to take the site condition into consideration when designing those PMS. Generally speaking, when site conditions change, the engine efficiency map also changes. In this case, the "derate" table of the engine efficiency can be added into the PMS model to calculate the difference in performance based on the actual site conditions. And then to make the implementation results more practical and robust against environmental variation.

Appendix A to Part 1

[0326]

Optimization Programs

```
                        A1 Two Identical Units
diary eff6.txt
clear all
Pmax=75;
for k=1:35
c(k)=k*2+5;
Aeq=[1 1 ];
beq=c(k);
lb=[0;0];
ub=[75;75];
if c(k)<=Pmax
   y0=c(k)−5;
   else
      y0=c(k)/2+5;%c(k)/2+30 for n=2
end
%for n=2;x0=[10;80];%[10;39];[20;40];[20;80];[30;48];[30;80];[40;60];
[40;70];[40;80]
x0=[y0;c(k)−y0];
%x0=[0;12];
options = optimset('LargeScale';off);
[x,fval]=finincon(@myfun6,x0,[ ],[ ],Aeq,beq,lb,ub,[ ],options);
x1 (k)=x(1);
x2(k)=x(2);
EFF(k)=c(k)/fval;
end
result=[c' x1' x2' EFT]
diary off
--------------myfunction6
function f = rnyfun6(x)
P=[10 14.7 20 23.3 30 40 50 60 70 80];
eff=[0.117 0.150 0.184 0.200 0.226 0.254 0.274 0.292 0.304 0.311];
n=2;
format long
p=polyfit(P,eff,n);
f=(x(1)/poIyval(p,x(1))+x(2)/polyval(p,x(2)));
                       A2 Three Identical Units
diary eff7.txt
Pmax=75;
for k=1:70
c(k)=k*2+5;
Aeq=[1 1 I];
beq=c(k);
lb=[0;0;0];
ub=[Pmax;Pmax;Pmax];
if c(k)<=Pmax
   yl=c(k)−5;
   y2=2;
elseif Pmax<c(k)<=2*Pmax
      yl=c(k)/2+2;
      y2=2;
else
   y 1 =c(k)/3+10;
   y2=c(k)/3+10; end
x0=[y 1 ;y2;c(k)−y1 −y2];
[x,fval]=finincon(@myfun7,x0,[ ],[ ],Acq,beq,lb,ub);
x1(k)=x(1);
x2(k)−−x(2);
x3(k)=x(3);
```

-continued

Optimization Programs

```
EFF(k)=c(k)/fval;
end
result=[c' x1' x2' x3' EFF']
diary off
----------myfun7----------
function f = myfun7(x)
P=[10 14.7 20 23.3 30 40 50 60 70 80];
eff=[0.117 0.150 0.184 0.200 0.226 0.254 0.274 0.292 0.304 0.311];
n=2;
format long
p=polyfit(P,eff,n);
      f=(x(1)/polyval(p,x(1))+x(2)/polyval(p,x(2))+x(3)/polyval(p,x(3)));
                        A3 Two different Units
diary eff8.txt
clear all
Pmax1=75;
Pmax2=150;
for k=1:215
c(k)=k*1+10;
Aeq=[1 1 ];
beq = c(k) ;
lb=[0;0;0];
ub=[Pmaxl;Pmax2];
if c(k)<=PmaxI
   y0=c(k)−5;%or c(k)/2;if change to c(k)/4,the result will change
elseif c(k)<=Pmax2
      y0=2;%or 12
else
         y0=c(k)/4;%or c(k)/3,or c(k)/3+40,c(k)/2
end
x0=[y0;c(k)−y0];
[x,fval]=finincon(@myfun8,x0,[ ],[ ],Aeq,beq,lb,ub);
xl (k)=x(1);
x2(k)=x(2);
EFF(k)=c(k)/fval;
end
result=[c' x1' x2' EFF']
diary off
-------------myfun8
function f= myfun8(x)
P=[10 14.7 20 23.3 30 40 50 60 70 80];
eff[ 0.117 0.150 0.184 0.200 0.226 0.254 0.274 0.292 0.304 0.311];
P2=[20 30 40 60 80 90 100 120 140 150];
eff2=[0.1029 0.14 0.1699 0.2130 0.2410 0.2502 0.2605 0.279
0.2906 0.293];
n=2;
format long
p=polyfit(P,eff,n);
q=polyfit(P2,eff2,n);
f=(x(1)/polyval(p,x(1))+x(2)/polyval(q,x(2)));
   A4 Three Different Units with Two 751cw Units and One 150 kw Unit
diary eff9.txt
Pmax1=75;
Pmax2=I50;
for k=1:290
c(k)=k*1+10;
Aeq=[I 1 1];
beq=c(k);
lb=[0;0;0];
ub=[Pmax I ;Pmaxl ;Pmax2];
if c(k)<=Pmax 1
y1=c(k)−5;
y2=2;
elseif c(k)<=Pmax2
yl=c(k)/2+2;
y2=c(k)/2+2;
elseif c(k)<=Pmax2+Pmax1
y1=2;
y2=c(k)13+10;
else
yl=c(k)/4;
y2=c(k)/4;
end
x0=[y/;y2;c(k)−y1−y2];
```

-continued

Optimization Programs

```
[xfval]=fmincon(@myfim9,x0,0,[ ],Aerbbeq,lb,ub);
xl(k)=x(1);
x2(k)=x(2);
x3(k)=x(3);
EFF(k)=c(k)/fval;
end
result-lc' xi' x2' x3' EFF']
diary off
-------------myfun9--------------------
function f = myfun9(x)
P=[ 10 14.7 20 23.3 30 40 50 60 70 80];
eff=[ 0.117 0.150 0.184 0.200 0.226 0.254 0.274 0.292 0.304 0.311];
P2=[ 20 30 40 60 80 90 100 120 140 150];
eff2=[ 0.1029 0.14 0.1699 0.2130 0.2410 0.2502 0.2605
0.279 0.2906 0.293];
n=2;
format long
p=polyfit(P,eff,n);
q=polyfit(P2,eff2,n);
f=(x(1)/polyval(p,x(1))+x(2)/polyval(p,x(2))+x(3)/polyval(q,x(3)));
    A5 Three Different Units with One 75 kw Unit and Two 150 kw Units
diary eff10.txt
clear all
Pmax1=75;
Pmax2=150;
for k=1:360
c(k)=k*1+10;
Aeq=[1 1 1];
beq=c(k);
1b=[0;0;0];
ub=[Pmaxl ;Pmax2;Pmax2];
if c(k)<=Pmaxl
yi=c(k)−5;
y2=2;
elseif c(k)<=Pmax2
y1=2
y2=2;
elseif c(k)<=Pmax2+Pmaxl
yl=c(k)/3;
y2=2;
elseif c(k)<=2*Pmax2
y1=2;
y2=c(k)/2;
else
yl=c(k)/5;
y2=c(k)/2;%or y2=2*c(k)/5
end
x0=[yl ; y2; e(k)−yl−y2] ;
[x,fval]=fmineon(@rnyfun10, x0, [ ], [ ],Aeq,be q, lb,ub);
xl(k)=x(1);
x2(k)=x(2);
x3(k) =x(3);
EFF(k)=e(k)/fval;
end
result=[c' x1' x2' x3' EPP]
diary off
      myfun10
function f = myfun10(x)
P=[ 10 14.7 20 23.3 30 40 50 60 70 80];
eff=[ 0.117 0.150 0.184 0.200 0.226 0.254 0.274 0.292 0.304 0.31 I];
P2=[ 20 30 40 60 80 90 100 120 140 150];
eff2=[ 0.1029 0.14 0.1699 0.2130 0.2410 0.2502 0.2605 0.279
0.2906 0.293];
n=2;
format Iong
p=polyfit(P,eff,n);
q=polyfit(P2,eff2,n);
f=(x(1)/polyval(p,x(1))+x(2)/polyval(q,x(2))+x(3)/polyval(q,x(3)))
```

Appendix B to Part 1

[0327]

FIS Files

```
   B1 FIS of ELSE subsystem for Three Different Units
        with Two 75 kw Units and One 150 kw Unit
[System]
Name=13MS__5__3DIFF__25__2'
Type--'mamdatur
Version=2.0
Numlnputs=1
NumOutputs=1
NumRules=4
AndMethod='rrin'
OrMethod='max'
ImpMethod='min'
AggMethod-'max.
DefuzzMethod='centroid'
[Inputl]
Name-'Total__Required__Power.
Range-[223 300]
NumMFs=4
MF1=1.2:1rapmf,[223 230 250.1944444444-44 258]
MF2='M'I'trapmf,f254 262 280.342592592593 289]
MF3=Th'Crapmf,[278.6 283.8 301A 314.6]
MF4----'01C:trapmf,[248 254259.564814814815 265
[Outputl]
Name--'unit 3'
Range=[110 150]
NumMFs=4
MF1=1;:lrapmf,[110.2 116.8 129.9 137.9]
MF2='M'ftrapmf,[140.3 142.3 146.5 148.8]
MF3-11'..'trapmf,[146.1 1473 150 162.6]
MF4-'0K'ftrapmf,[134.7 137.6 141.8 144]
[Rules]
1, 1 (1): 1
2, 2 (1): 1
3, 3 (1): 1
4, 4 (1): 1
   B2 FIS of ELSE subsystem for Three Different Units
        with One 75 kw Unit and Two 150 kw Units
[System]
Name='PIV1S__5__3D1FF__2B__2'
Type='mamdani'
Version=2.0
NumInputs=I
NumOutputs=1
NumRules=5
AndMethod−=min*
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='centroid.
[Inputl]
Name='Total Required__Power'
Range=[300 375]
NumMFs=5
MF1='12:'trimf,[297.7 312.3 334.5]
MF2='El'Atimf,[341.1 356.6 372.9]
MF3='MYtriruf,[322.3 338.6 356.8]
MF4=bk'ftrimf,[309.1 3253 341.4]
MF5=VI-1*:*trimf,[359.1 366.6 376.8]
[Outputl]
Name='unit__1'
Range=[59 75]
NumMFs=5
MF1='H'ftrirnf,[69.42 72.71 73.63]
MF2=TH':trimf,[73.2 74.54 77.18]
MF3&L':'trimf,[58.52 62.88 67.2]
MF4='M'Arimf,[65.85 70.3 70.92]
MF5='0K'ftrimf',[63.57 66.45 68.6]
[Rules]
1, 3 (1) : 1
2, 1 (I) : 1
```

-continued

FIS Files
3, 4 (1): 1
4, 5 (I): 1
5, 2 (1) 10

Part 2—System Description

1. Introduction

[0328] One object of this design is to provide an integrated and optimized Distributed Generation System Control (DGSC) module that improves the overall power generation performance attributes and lowers the capital cost of a distributed generation site installation. The described embodiment leverages the IEEE 1547 "Standard for Interconnecting Distributed Resources with Electric Power Systems" ("Standard" herein) that has defined a nationally recognized standard for grid interconnect for distributed generation (DG) resources.

2. Top Level System Controller Description

[0329] 2.1. Scope

[0330] This section outlines the characteristics of one embodiment of a top level system control of internal combustion engine generating system production units. Other types of generators and further variations of the specific design choices described herein will occur to those skilled in this technology area without undue experimentation based on this disclosure.

[0331] 2.2. Applicable Documents

[0332] 2.2.1. Standards

[0333] Commercial production components of the system may be selected to comply with the relevant aspects of IEEE 1547 and other standards listed below.

[0334] IEEE P1547 Standard for Distributed Resources Interconnected with Electric Power System

[0335] IEEE 519 IEEE Recommended Practices and Requirements for Harmonic Control in Electric Power Systems

[0336] UL 508C Standard for Power Conversion Equipment

[0337] UL 2200 Stationary Engine Generators Assemblies

[0338] NFPA 70 National Electric Code

[0339] 2.2.2. Acronyms

DG Distributed Generator

DGSC Distributed Generator System Controller

HMI Human Machine Interface

[0340] GPM Grid Parallel Mode of operation
GIM Grid Isolated Mode of operation

I/O Input/Output

ISO International Organization for Standardization

UIC Utility Interface Controller

[0341] 2.3. System Overview

[0342] 2.3.1. System Definition

[0343] A DG unit is an internal combustion engine generating system with a synchronous generator. The DG system controller coordinates the control of one or more DG units as shown in FIG. **46**. With the addition of hardware modules, the system controller shall be capable of controlling up to 8 parallel DG units that are connected to the local utility grid in GPM or to the local site distribution bus in GIM. The demonstration prototype will be capable of controlling 2 parallel units. Researchers will investigate the possibility of including DG units with induction generators.

[0344] 2.3.2. DG Unit Description

[0345] The DG unit is a natural gas fueled engine driving a synchronous generator that produces 60 Hz three-phase electrical power at 277/480 VAC under ISO standard conditions. The rated output of each DG unit will be within the range 75-200 kW.

[0346] 2.3.3 Software Coding

[0347] The preferred programming language for system control functions is C or C++, though many other computing languages would also be appropriate. Exceptions in various embodiments include relaying functions written in assembly language and monitoring software written with Microsoft Visual Basic.

[0348] 2.3.4 Controller Structure and Communication

[0349] Each DG unit may include a subsystem control module. There are two main variations of the control structure presently under consideration.

[0350] Structure A involves DG subsystem modules communicating with a supervisory controller. The supervisory controller will perform the UIC functions as shown in FIG. **47**.

[0351] Structure B involves DG subsystem modules communicating with each other and one module will assume the supervisory (i.e. master) control function. There will be an automatic procedure for determining which machine serves as the master. The user will also be able to specify a priority list for selecting the master unit. Within structure B there are two variants:

[0352] Structure B.1 has a separate UIC, somewhat similar to FIG. **47**.

[0353] Structure B.2 incorporates UIC capabilities into every subsystem module.

[0354] Regardless of the structure, in these embodiments every subsystem module and any separate supervisory controller or UIC will be connected to a CAN bus network.

[0355] Regardless of the structure, in these embodiments every subsystem module and any separate supervisory controller or UIC will have an HMI.

[0356] Regardless of the structure, a password protected TCP/IP interface will allow a local or remote operator to monitor performance of the DG system and individual units. The TCP/IP interface will also allow the operator to upload and download system and individual unit control specification parameters interactively and in .xml file format.

[0357] There may also be dedicated analog and digital signal channels in addition to the CAN bus.

[0358] 2.3.5 Control of Electrical Connections

[0359] Each DG unit shall be equipped with a motorized circuit breaker to interface with the site load bus. The DG system controller will also control the utility tie circuit breaker. The DG unit breaker and utility breaker control shall be governed according to the IEEE 1547 standard. Each DG unit shall be equipped with a fuel solenoid shutoff valve. The DG system controller will control the turning on and off of an additional fuel solenoid shutoff valve.

[0360] 2.3.6 Import/Export Control

[0361] The generator set shall include provisions for controlling the amount of power that is imported from or exported to the utility grid.

TABLE 2-1

General characteristics of DG unit
General characteristics

Rated Output Power of Each DG Unit:	75 kW < P_{rated} < 200 kW
AC Output Voltage:	3-phase, 277/480 VAC
Output Frequency:	60 Hz
Maximum Number of Units:	8 DG units
Net Electric Efficiency at ISO at 75% Load	28%
Base System Output for Pricing Purposes	480 V/60 Hz
Power Factor Capability	>0.8
Standard for Grid Interconnection	IEEE 1547

[0362] 2.4. DG System Controller Interface Descriptions

[0363] 2.4.1 Human Machine Interface

[0364] Each DG unit and any supervisory controller in this embodiment is equipped with a Human Machine Interface (HMI) device to provide a local operator control panel, parameter settings, system indicators, visual alarm or shutdown information, and emergency power off. The HMI devices shall be able to display the following information: generator voltage and current, frequency, DG output in kw and kvar, oil pressure, water temperature, synchronization status, total kilowatt hours, and alarm history.

[0365] 2.4.2. List of I/O for System Controller

[0366] Input switches in the preferred system controller include a run with load switch, mode selection switch (automatic mode or test mode), emergency stop switch, voltage raise/lower switch, and speed raise/lower switch. Utility switches include a "Utility source circuit breaker Aux in" input switch, "Utility source circuit breaker open/close" input switch, a "Utility/bus PT input" switch, a "Local bus PT disconnect," and a "Mains PT disconnect" switch.

[0367] The generator portion of the system controller in this embodiment includes these inputs: "Generator circuit breaker Aux in," "Generator circuit breaker open/close," "Generator CT," and "Generator PT." Outputs are included for voltage bias and a Generator On/off Switch.

[0368] Engine-related I/O in this embodiment includes inputs for Engine warning, Engine fault, MPU pickup, Idle/rated speed, and Engine crank, as well as outputs for Speed bias, Fuel shutoff valve, and Engine On/off Switch.

[0369] 2.5. Summary of System Functions

[0370] 2.5.1 Summary of System Controller Functions

[0371] The system controller (DGSC) in the present embodiment provides several functions. At the system level, the DGSC provides grid synchronization and protection of the DG Unit, Regulation in GPM and GIM, and a human/automatic switch between GPM/GIM. Internally, the DGSC provides control of power to ENI auxiliary loads, provides gas shutoff valve control, monitors utility voltage, and implements lower-voltage power supply control. Further, the DGSC adds communication functionality in controlling the HMI, communicating with an external DG gateway, and controlling import and export of power.

[0372] 2.5.2. Summary of UIC Functions

[0373] The UIC in the present embodiment complies with the IEEE 1547 Standard for Interconnecting Distributed Resources with Electric Power Systems.

[0374] 2.5.3 Summary of DG Unit Protection Functions

[0375] The DG units in the present embodiment include current limit protection, emergency power shutdown, protective power shutdown, warning indications, generator over/under voltage indications, generator over/under frequency indications, anti-islanding protection, loss of mains detection, loss of generator excitation, speed/frequency mismatch protection, reverse power protection, and short circuit protection.

[0376] 2.6 Mode of Operations and Regulations

[0377] 2.6.1 Single Unit Configurations

[0378] Single units in the present embodiment may operate in GIM (voltage-regulating speed control mode) or GPM (voltage-following torque control mode).

[0379] 2.6.2 Multiple Unit Configurations

[0380] Likewise, multiple-unit configurations of the present embodiment can also operate in GIM or GPM. In GIM, one unit uses voltage-regulating speed control, while other units use voltage-following torque control. In GPM, all units use voltage-following torque control.

[0381] 2.6.3 Loss of Utility Power

[0382] In the event of a utility power outage, the DG system automatically disconnects from the utility and continues to provide power to the load. If the engines are not operating at the time of the outage, they start automatically, disconnect from the utility, and provide power to the load. Upon return of the utility power, the DG system synchronizes to the utility grid, reconnects to the utility, and continues as instructed prior to the outage.

[0383] 2.6.4 Starting Parallel Units

[0384] In multi-unit operation, when a start command is sent to the DGSC, all units start and operate parallel to each other, then to the utility grid as needed to supply the load.

[0385] 2.6.5.50/60 Hz Operation

[0386] The DG system is capable of operating at either 60 Hz or 50 Hz with a gearbox changes and appropriate electrical component changes.

[0387] 2.6.6. Automatic and Manual/Test Modes

[0388] The DG system controller is capable of operating in either Manual/Test Mode or Automatic Mode.

[0389] 2.6.7. Regulation in Grid Isolated Mode (GIM)

[0390] The total harmonic distortion of the DG system electrical output is less than 5% at no load. The DG system controller is capable of regulating the output voltage within a maximum range of +/−0.5% of nominal voltage in GIM (per ANSI84.1B). The DG system controller is capable of maintaining a maximum output voltage deviation of +20% to −25% given a 25% load pickup or a 100% load rejection in GIM. The frequency operating range of DG system in GIM is defined as +/−3% for 50 Hz or 60 Hz systems.

[0391] 2.6.8 Regulation in Grid Parallel Mode (GPM)

[0392] The DGSC includes provisions for controlling the amount of power that is imported from or exported to the utility grid. The DGSC complies with the IEEE 1547 Standard for Interconnecting Distributed Resources with Electric Power Systems.

3. Modeling Overview

[0393] An integrated system model for the present embodiment was developed and validated in Matlab/Simulink. The modeling of the system included four major parts: engine, generator, load/grid, and system controller. Models for each individual component were developed and all of them were integrated into a system model.

[0394] 3.1. Engine and Engine Control Module (ECM)

[0395] The engine model in this embodiment is a simplified inertia model based on the simple engine model illustrated in FIG. **48**. The input parameter is the throttle angle command. The output parameter is engine torque generated and actual engine speed. The engine load torque is regarded as a disturbance to the engine system.

[0396] A quasi-static engine model (shown in FIG. **49**) was applied in the simulation of the DG system because the engine operates at an approximately constant speed (e.g., at 1800 rpm).

[0397] The engine control module (ECM) is an independent control executing all of the engine management functions, including fuel/air ratio, governor, ignition and health monitoring. In the case of original engine speed closed-loop control, only a PID (or PI) feedback control was considered to maintain desired engine speed by controlling the engine throttle angle. It should be emphasized that the engine speed fluctuation should be kept to a minimum in order to achieve an expected grid frequency.

[0398] FIG. **50** shows a block diagram of the engine speed feedback control. By controlling the engine throttle angle, the engine speed can be maintained a set point. For example, if the load torque increases suddenly for some reason, the engine speed will drop. Via feedback and PID controller, an engine speed drop will cause increase of the throttle angle. Therefore, more engine torque will be generated to make the engine speed return to a set point.

[0399] It should be noted that when engine is running at torque control mode, the output speed is not determined by the engine itself. The output speed will follow the speed in the grid. The system controller will monitor the output power and adjust the speed bias command to control the system.

[0400] 3.2. Automatic Voltage Regulator (AVR) and Generator Module

[0401] 3.2.1. Introduction to the AVR

[0402] The controller will send a command signal to the AVR to regulate the voltage. Because the present embodiment uses a brushless generator, the AVR cannot control the field voltage (V_f) directly. The field voltage (V_f) will be controlled pass through the exciter part. Therefore, the controller will control the exciter voltage (V_r) in instead of the field voltage (V_f). The process of controlling voltage in the AVR starts from receiving the signal from controller, changing that signal to the exciter voltage (V_r). And the exciter voltage (V_r) creates flux which in this step, the process still happens in stator part and this flux will create the field voltage from the rectifier which is in the rotor part. The process of the system is shown in FIG. **51**.

[0403] In the next step, we simplify the above system by using the relationship between the AVR and Generator in which we know as the graph of OCC (Open Circuit Condition). The relationship illustrated in FIG. **52** helps us to translate from exciter voltage (V_r) to internal voltage (Ea). Below, we show the simulation results from this graph. After that, we can calculate the terminal voltage (Vt) from the relationship between the output power, the armature reactance and the internal voltage assuming the power factor (PF) equals to 1. The formulas used to calculate the terminal voltage are derived from the phasor and schematic diagrams in FIG. **53**:

$$P = \frac{3V_t E_a \sin\delta}{X_s}$$

$$V_t = E_a \cos\delta - jX_s I_a \sin\theta$$

[0404] 3.2.2. AVR Model

[0405] We use the relationship above to generate the Simulink block diagram. As illustrated in FIG. **54**, the input to the AVR is the error voltage signal from the controller. The output of the AVR and the input of the generator are the exciter voltage. And the output of the generator is the terminal voltage (V_t).

[0406] Instead of using the exciter and rectifier, we will use graph of OCC (Open Circuit Condition) to build the relationship. Then, we use close loop control with PID to build the model below.

[0407] In the example embodiment, the model of FIG. **55** takes 480V as the reference input value. After simulation with the step function of output power signal, we get the steady-state output at 480V also.

[0408] 3.2.3. Simulation Results

[0409] The model that is shown in FIG. **56** is the AVR model by Matlab/Simulink program. This model is based on the model from FIG. **55**.

[0410] The simulation results are shown in FIGS. **57** and **58**. These figures illustrate the same data, but FIG. **55** is plotted with frequency 60 Hz. FIGS. **57**-**58** each include 3 graphs: Load Power (kw) vs. Time (s), Terminal Voltage (V) vs. Time (s) and Current (A) vs. Time (s).

[0411] For this simulation, we look closely at the effect of the terminal voltage (V_t) and current when the load power changes. We assume that the load power drops from 60 KW to 50 KW and suddenly increases from 50 KW to 60 KW at 6 and 12 seconds, respectively. And the figures show the changes of both voltage and current graphs.

[0412] 3.3. System Controller Model

[0413] The list of system I/O for the system controller in this embodiment appears in Section 2.4 of this Part. The list of system functions appears in Section 3.3.2. In this example embodiment, most of those functions are implemented in a state machine.

[0414] 3.3.1. PARAMETERS for the system controller

[0415] The parameters provided to the system controller in the present embodiment include: DG unit auxiliary load parameters, maximum current, maximum voltage, maximum engine speed, maximum load, maximum power, and maximum generator frequency.

[0416] 3.3.2. System Function

[0417] There are two types of controllers in the system: DG Unit Controllers and one or more Supervisory Controllers. The former control a single DG unit while the latter coordinate the operation of multiple units in a single location or portion of a DG system.

[0418] DG Unit Controllers regulate the engine in GPM and GIM, control power to ENI auxiliary loads, control the gas shutoff valve, monitor utility voltage, control the lower voltage power supply, provide current limit protection, control emergency power shutdown and protective power shutdown, generate warning indications, detect and indicate generator over/under voltage and generator over/under frequency conditions, provide anti-islanding protection and loss of mains detection, detect and indicate loss of generator excitation and speed/frequency mismatch, provide reverse power protection, detect and indicate short circuit conditions, and control the unit's import and export of power.

[0419] The Supervisory Controller controls grid synchronization and protection of the DG Unit, provides a human/automatic switch between GPM/GIM, controls the HMI, and communicates with an external DG Gateway. The UIC Controller complies with the IEEE 1547 Standard for Interconnecting Distributed Resources with Electric Power Systems.

[0420] 3.3.3. GPM/GIM Mode Regulation

[0421] The DG system can run at GIM mode or GPM mode. Each DG unit will be running at either speed control mode or torque control mode according to the situation. The supervisory controller will determine the control mode of each DG unit. The basic rules are as follows

TABLE 3-1

Control mode rules in GIM and GPM mode

	GIM	GPM
Single DG unit	speed control	torque control
Multiple DG units	one master unit uses speed control, other units use voltage-following torque control	all units use torque control

[0422] When running in speed control mode, DG unit controller will send out constant speed bias command to the Engine Control Module (ECM). The constant speed bias will be 1800 rpm (translated from 60 Hz frequency). DG unit controller also sends out constant voltage bias command to the Automatic Voltage Regulator (AVR).

[0423] When running at torque control mode, DG unit controller will monitor the output power of dg unit and adjust the speed bias command to control the output torque. The speed bias command sent to the ECM would change the output torque instead of the speed of the engine. The engine speed will follow the grid speed automatically. DG unit controller also sends out voltage bias command, which is same as the grid voltage, to the Automatic Voltage Regulator (AVR).

[0424] 3.3.4. Feed-Forward Controller in GIM Mode

[0425] It can be seen from the FIG. **50**, by controlling the engine throttle angle, the engine speed can be maintained a set point. For example, if the load torque increases suddenly for some reason, the engine speed will drop. Via feedback and PID controller, an engine speed drop will cause increase of the throttle angle. Therefore, more engine torque will be generated to make the engine speed return to a set point.

[0426] It can be also seen from the FIG. **50** that the engine speed definitely drops when the load torque increases suddenly no matter how the PID controller was designed. That is, an engine speed bias was sensed firstly, and then the PID controller takes effect to maintain the engine speed. The simulation results (as shown in FIG. **59**) prove that the engine speed is subject to the changing load torques when only a feedback control is used.

[0427] Combined feed-forward plus feedback control is proposed to reduce the effect of the measurable disturbance (such as load torque) on the engine speed output. In the most ideal situation, feed-forward control can entirely eliminate the effect of the measurable disturbance on the engine speed output. Even when there are modeling errors, feed-forward control can often reduce the effect of the measurable disturbance on the output better than that achievable by feedback control alone.

[0428] In the DG system, the engine load torque can be calculated based on the local consumed power and the amount of power that is imported from or exported to the utility grid. FIG. **60** gives the block diagram of a feed-forward plus feedback engine speed control system considering the engine torque is a measurable disturbance.

[0429] In FIG. **60**, the load torque can be calculated by the summary of various load powers. Therefore, the engine load torque can be regarded as a measurable signal, rather than a disturbance signal.

[0430] FIG. **61** gives the simulation results of the feed-forward plus feedback engine control system. It can be seen that feed-forward control is able to rapidly suppress the changes of the engine load torque. For example, if the load torque increases suddenly for some reason, the feed-forward control goes into action and the throttle angle will increase immediately even without engine speed drop. The engine speed is robust against the changing engine load torque. Therefore, the engine speed fluctuation is little subject to the changing engine load torque via feed-forward control.

[0431] 3.3.5. Engine Control in GPM Mode

[0432] When considering the problem of frequency control of interconnected areas, we assume that the utility grid is an infinite power pool. That is, the generator can not change the grid frequency. The following block diagram gives the power flows among the generator, various loads and tie-line and also gives the phasor diagram for the generator and utility grid.

[0433] In FIG. **62**, we can obtain the formula for the real (not including reactive) generator power, as follows:

$$P_G = P_L - P_U = \frac{|E||Vt|}{X}\sin\delta$$

where the angle δ is the phase angle between the generator emf E and the utility voltage Vt. Here the power angle δ is defined positive when E leads Vt; P_G, P_L, P_U, are the real power of the generator, various loads and utility grid respectively.

[0434] In the present embodiments, before the generator is connected with the grid, the two are synchronized. Synchronization can take place if the following conditions are all satisfied:

1. The generator and grid frequencies are equal.
2. The phase sequence of the generator matches that of the grid.
3. The generator emf E and the grid voltage U are of equal magnitude.
4. E and U have equal phase.

If and only if all these conditions are satisfied, the voltage across the circuit breaker 52G1 and 52-1 is zero. At this time, the generator can be smoothly "locked" onto the grid by closing the circuit breakers.

[0435] Once the synchronous generator is synchronized onto the grid network, its speed will always synchronize with the grid frequency as long as the grid power is large enough. That is, the speed of the engine coupling with the generator can not be changed either.

[0436] It can be observed that the magnitude and direction of the power flow of the machine will be determined according to the power angle δ. In reality, the power angle δ is controlled by means of the engine torque (or engine power). If the engine produces more power than consumed by various loads, the rest of power will flow to the grid.

[0437] To sum up, when DG units are interconnected with the grid, though the engine speed cannot be changed by the throttle, the throttle can still control the engine torque (or engine power) so as to control the power flow between the generator and the tie-line. FIG. **63** gives the block diagram of the feed-forward control for maintaining a minimum import power from the grid.

[0438] The engine speed controller (inner PID) accepts an engine speed bias as an input, but the engine speed is completely determined by the grid. So if we set the engine speed bias equal to 5 rpm and the synchronous engine speed is actually 1800 rpm, then engine speed controller might tell the engine we want to run at 1805 rpm. Then the engine would try harder, thus increasing power, but would still turn at grid synchronous speed 1800 rpm.

[0439] The outer PID controller monitors the grid power (or generator output power minus load power) in load-following mode to determine the engine speed bias sent to the engine speed controller (inner PID). As mentioned above, the inner PID can control the engine torque (or engine power) output, thus controlling the amount of the power imported from the grid indirectly.

[0440] Assuming the minimum import power in the utility contract is 10 kW, the simulation results (shown in FIG. **64**) show how the outer PID controller works to maintain 10 kW import power.

[0441] 3.4. Load/Grid Module

[0442] The grid is modeled as a voltage source and impedance, while the load is modeled as an impedance. The input parameters for the load/grid are the impedance of the grid, the working voltage of the grid, the maximum voltage of the grid, and the impedance of the load.

[0443] 3.5. Islanding Detection

[0444] The dynamic model of a DG connected to the grid resembles that of a network containing one small generator connected by a distribution line to a much larger generator. The small generator represents the DG and the large generator represents the grid. All the synchronous generators in a power system operate in lock step synchronism because the AC voltage results in synchronization of torques between the machines. Therefore, all machines in an interconnection have nearly identical average frequencies. However, there can still be differences in the instantaneous frequencies of interconnected machines. The present islanding-detection scheme uses the fact that deviations in frequency caused by a deliberate perturbation signal are much smaller when a DG is connected to the grid than when it is in an island.

[0445] The circuit diagram in FIG. **65** shows the classical model of two synchronous generators connected by a distribution line. The voltage source in series with reactance jx_2 on the right hand side represents the DG unit synchronous generator. Resistance R_2 represents a load that matches the DG real power output. The corresponding elements on the left hand side of the diagram represent a generator having 100 times the inertia of the DG along with its matching load. Reactance jx_L represents a power distribution line. Islanding is simulated by disconnecting the distribution line.

[0446] FIG. **66** shows simulated frequency measurements from the DG unit before and after the distribution line is disconnected. A 1 Hz perturbation signal is superimposed on the DG unit's throttle signal during the entire simulation. The perturbation signal produces a 1% variation of the DG mechanical input power. The horizontal axis indicates measurement samples that have a frequency of 720 Hz. Disconnection of the distribution line occurs after 5 seconds (3600 samples). Measurement noise has been simulated by adding actual frequency data from a PMU. Frequency deviations caused by the perturbation signal are almost completely obscured by noise before the DG and its matching load become an island. The 1 Hz frequency deviation is easily observed after the distribution line between the two machines is removed.

[0447] FIG. **67** shows the magnitude of the 1 Hz component of the signal in FIG. **66**. This simulation shows that a simple threshold test of this magnitude should work as the islanding detection algorithm.

[0448] After the DGSC has been modified to inject a 1 Hz perturbation signal into the throttle of a synchronous DG unit, experimentation, simulations and analysis can be carried out to determine appropriate thresholds for the algorithm. Research on such details of the algorithm will proceed in parallel with most of the engineering tasks. Ultimately, the supervisory controller will be programmed to determine an appropriate setting for the threshold test depending on the number and type of DG units connected in parallel. The island-detection scheme will be validated in the test cells at I Power according to the IEEE 1547 conformance test procedures. Possible additional variations in an island configuration, such as off-site DG units connected to the same feeder, will be studied in simulation to determine whether any additional logic is required.

4. Utility Interface Controller (UIC)

[0449] IEEE Standard 1547, "Standard for Interconnecting Distributed Resources with Electric Power Systems", provides for the first time a nationally recognized set of rules for connecting distributed generation (DG) resources to a utility grid. This standard has the potential to be used by federal legislators, by state public utilities commissions, and by over 3000 utilities in formulating technical requirements for grid interconnection of DG. This standard specifies requirements for synchronization, protective relaying, and switching coordination that were formerly determined on a utility-by-utility basis.

[0450] There is a large amount of redundancy between the basic functions used in DG operation and those required for compliance with the IEEE 1547 standard. Both require and/or can be partially met through the proper use of over/under voltage relays, over/under frequency relays, fault current detection relays, and synchronization check relays. The following section summarizes aspects of the UIC specification for use in many embodiments of this system.

[0451] 4.1. Partial Summary of IEEE 1547 Standard

[0452] 4.1.1 Synchronization (Section 4.1.3)

[0453] The Standard states that the DR must synchronize with the EPS without causing a voltage magnitude fluctuation at the Point of Common Coupling (PCC) greater than ±5% of the operating voltage. PCC is defined as "the electrical connecting point or interface between the utility distribution system and the customer's or user's electrical distribution system."

[0454] The output of the DR and the input of the EPS at the PCC have substantially the same voltage magnitude, the same frequency, and the same phase angle. IEEE Standard 1547 requires that the difference between the voltage of the distributed generator and the EPS not be more than 10%, the difference between their frequencies not exceed 0.3%, and the difference between their phase angles not exceed 20°. Only if these conditions are satisfied can the distributed generator be connected into the electric power system. The following table illustrates the requirements of the Standard.

TABLE 4-1

IEEE 1547 Requirements and conditions

Aggregate Rating of DR Units (kVA)	Frequency Difference (Δf, Hz)	Voltage Difference (ΔV, %)	Phase Angle Difference (Δø, °)
0-500	0.3	10	20
500-1,500	0.2	5	15
1,500-10,000	0.1	3	10

[0455] In order to adhere to this requirement, we can use relays to estimate the magnitudes and phase angles of the voltages, and frequencies on the DR side and the EPS side. It can then be checked if the differences between the estimations are within the ranges mentioned above. If they exceed the mentioned values then the DR will not be connected to the EPS. In order to ensure that the distributed resource will synchronize with the EPS with a voltage fluctuation limited to ±5% of the operating voltage, we can carry out the test by connecting the DR to the EPS a couple of times and checking if the voltage fluctuation is within ±5% of the operating voltage.

[0456] 4.1.2. Inadvertent Energizing of Area EPS (Section 4.1.5):

[0457] The Standard states that the distributed resource must not connect to the PCC if the Area EPS is in the de-energizing state. It also states that the distributed resource must not connect to the PCC until the voltage of the EPS is in the range of 106V-132V and the frequency is in the range of 59.3 Hz to 60.5 Hz. After the EPS has regained the normal voltage and frequency, the Standard requires that the distributed resource wait for up to 5 minutes before it reconnects to the EPS. The following table illustrates this requirement and conditions.

TABLE 4-2

EPS status and conditions

Whether the DR is connected to the PCC	EPS Status and Conditions
Not connected	EPS in a deenergized state.
Connected	EPS in a normal state: 106 V < V < 132 V and 59.3 Hz < f < 60.5 Hz.
Not connected	EPS in an abnormal state: V ≦ 106 V or V ≧ 132 V or f ≦ 59.3 Hz or f ≧ 60.5 Hz
Connected with a 5-minute delay	EPS recovered from an abnormal state to a normal one.

This requirement may be satisfied by each time checking the voltage magnitude and frequency of the EPS after the occurrence of an outage. We can then compare these values obtained with the above mentioned threshold limits. If they are within the limits then wait for 5 minutes and then check again the voltage and frequency values. If they are still within the limits then close the circuit breaker between the DR and the EPS else keep the circuit breaker open and carry out the same procedure until the condition is satisfied.

[0458] 4.1.3. Voltage Disturbances (Section 4.2.1)

[0459] The standard states that the protective functions of the interconnection system must measure the fundamental frequency component of the phase to phase or the phase to neutral voltage. If the measured quantity is less than/equal to 106V or greater than/equal to 132V then the distributed resource must stop energizing the EPS within the clearing times as indicated in the table below.

TABLE 4-3

Voltage clearing times per IEEE 1547

Voltage range (V)	Clearing times (sec)
V < 60	0.16
60 < V < 106	2
132 < V < 144	1
144 < V	0.16

[0460] Voltage disturbances bring about a change in the voltage of the power system. Relays can be used to check the voltage at the PCC. Whenever the voltage goes beyond the limits mentioned, the Standard requires that the relays disconnect the DR from the EPS within the respective clearing times as mentioned in the table above.

[0461] 4.1.4. Frequency Disturbances (Section 4.2.2):

[0462] The Standard states that the protective functions of the interconnection system must measure the frequency. If the measured frequency goes out of the normal range (59.3 Hz<f<60.5 Hz) then the distributed resource must stop energizing the EPS within the clearing times as indicated in the table below.

TABLE 4-4

Frequency disturbances requirements

DR Size	EPS Frequency	DR "Cease to Energize" Time
≦30 kW	>60.5 Hz	.16 sec.
	<59.3 Hz	.16 sec.
>30 kw	>60.5 Hz	.16 sec.
	59.3-57.0 Hz	Time Delayed
	<57.0 Hz	.16 sec.

[0463] Frequency disturbances bring about a change in the frequency of the power system. Over-/under-frequency relays can be used to check the frequency at the PCC. Whenever the frequency goes beyond the limits mentioned, the relays must disconnect the DR from the EPS within the respective clearing times as mentioned in the table above. Note that DR units with a total capacity larger than 30 kW can have an impact on distribution system security. Thus, the IEEE 1547 requirement leaves this question to the Area EPS operator who specifies the frequency setting and time delay for under-frequency trips down to 57 Hz.

[0464] 4.1.5. Disconnection of Faults

[0465] The Standard states that the distributed resource must disconnect from the area EPS for faults on the area EPS it is connected to. It requires the DR to have a current based protection and a ground fault protection suitable for detection of area EPS faults.

[0466] To provide current based protection, over current relays following the inverse definite minimum type (I.D.M.

T.) characteristics will be used. The trip and reset characteristics of the over current relay is as shown in FIG. **68**.

[0467] The time taken by the relay to operate depends on the torque, which in turn varies with respect to the current in the relay coil. The more torque, the less time taken by the relay to operate. Hence the relay is said to have inverse time characteristics.

[0468] In order to emulate an induction-type over-current relay, the various phenomena occurring in an induction disk of the electromagnetic relay can be represented in terms of mathematical equations. The equations used to simulate the trip and reset characteristics are as follows:

$T_r = TD(R/|M^2 - 1)$-------------------→for $0<M<1$---→Reset characteristic

$T_r = TD(A/(M^P - C)+B)+K$---------------→for $M>1$--------→Trip Characteristic

Where:

[0469] M=Multiple of PICKUP setting (0 to 40)

T_r=Time to trip when M≧1

T_r=Time to reset if relay is set for integrating reset when M<1. Otherwise, reset is 50 milliseconds or less

TD=TIME DIAL setting (0.0 to 9.9)

A, B, C, p, K=Constants for the particular curve

R=Constant defining the reset time.

[0470] 4.1.6. Implementation Using Matlab Code

[0471] The source code is attached in Appendix A.

[0472] 4.2. Phasor Estimation Methods

[0473] In phasor estimation we want to find the fundamental frequency sinusoid that best represents a set of measurement samples. We can estimate phasors using either least squares or orthogonal projection methods. Both methods give the same result when orthogonal vectors are used to represent the sample data. The orthogonal projection method is perhaps more intuitive and is explained in more detail below.

[0474] 4.2.1. Orthogonal Projection Method

[0475] The orthogonal projection of a vector V onto the span of $\{V_1, V_2\}$ is given by

$$V_{est} = \frac{<V, V_1> V_1}{\|V_1\|^2} + \frac{<V, V_2> V_2}{\|V_2\|^2}$$

where < > represents inner product of vectors. Orthogonal projection gives closest vector in the span of $\{V_1, V_2\}$ to V.

[0476] With K samples of data per fundamental frequency period, we use the following vectors

$$V_1(L)=[\cos((L-K+1)\theta)\cos((L-K+2)\theta)\ \ldots\ \cos(L\theta)]$$

$$V_2(L)=[\sin((L-K+1)\theta)\sin((L-K+2)\theta)\ \ldots\ \sin(L\theta)]$$

to approximate the most recent vector of samples

$$V(L)=[y_{L-K+1}y_{L-K+2}\cdots y_L]$$

where $\theta=2\pi/K=\omega_0\ \Delta t$, ω_0 is the nominal fundamental frequency, and Δt is the sampling interval. Then

$$Vest(L)=[A\cos((L-K+1)\theta+\phi)A\cos((L-K+2)\theta+\phi)\ \ldots\ A\cos(L\theta+\phi)]$$

is the best fundamental frequency sinusoidal approximation to the vector of samples, where the following expressions correspond with the most recent window of measurements:

$$A=\sqrt{Y_c^2+Y_s^2},$$

$$\varphi=\tan^{-1}\left(-\frac{Y_s}{Y_c}\right)+0.5\pi(1-\text{sign}(Y_c))$$

$$Y_{c=}\frac{<V, V_1>}{\|V_1\|^2},$$

$$Y_{s=}\frac{<V, V_2>}{\|V_2\|^2}.$$

[0477] Therefore $Ae^{j\Phi}(L)$ is the phasor estimate for the signal after sample L is received. The calculation is equivalent to the Discrete Fourier Transform with an assumed fundamental frequency equal to the nominal power system frequency ω_0. The actual power system frequency can be estimated from the rate of rotation of the calculated phasors:

$$f(L)=[\phi(L)-\phi(L-1)]/(2\pi\Delta T)$$

Averaging a few successive f(L) estimates is sufficient for filtering out the noise that is amplified by the derivative calculation.

[0478] 4.2.2 Recursive Calculation of Phasors

[0479] Recursive forms of Fourier algorithms are used to reduce the number of calculations compared with the non-recursive Fourier algorithm. For a full-cycle window (Kƒ=2π):

$$\hat{Y}_c^{(new)}=\hat{Y}_c^{(old)}+[y_{new}-y_{old}]\cos(L\theta)$$

$$\hat{Y}_s^{(new)}=\hat{Y}_c^{(old)}+[y_{new}-y_{old}]\sin(L\theta)$$

where y_{new} is the newest sample corresponding to L and y_{old} is the oldest sample corresponding to a full cycle earlier.

[0480] 4.2.3. Least Squares Formulation and the Transient Monitor Function

[0481] When the data window spans the instant of fault inception, the results obtained from almost any algorithm are unreliable as the data contains both pre-fault and post-fault samples. The least squares formulation of the sinusoidal estimate is convenient for calculating the error between the samples and the estimate. The sample values corresponding to the phasor estimate can be calculated as:

$$\hat{y}(L)=S(S^TS)^{-1}S^Ty(L)$$

where S is a matrix with two columns: $V_1(L)^T$ and $V_2(L)^T$. The reason L is not shown in the expression involving S is that the matrix S $(S^TS)^{-1}S^T$ turns out to be independent of L. The residual error vector is calculated as:

$$r(L)=\hat{y}(L)-y(L)=[S(S^TS)^{-1}S^T-I]y(L)=My(L)$$

The sum of the absolute values of the elements of r(L) is defined as the transient monitor function. A recursive calculation can also be derived for the residual error vector.

[0482] 4.3. Application of Symmetrical Component Calculations

[0483] A fault causes unbalanced current and voltages in the phases of a polyphase circuit. Three unbalanced phasors of a three-phase system can be resolved into three balanced system of phasors and are called symmetrical components of unsymmetrical phasors. The balanced sets of components are:

[0484] 1. Positive-sequence components consisting of three phasors equal in magnitude, displaced from each other by 120° in phase and having the same phase sequence as original phasors.

[0485] 2. Negative-sequence components consisting of three phasors equal in magnitude, displaced from each other by 120° in phase and having the phase sequence opposite to that of original phasors.

[0486] 3. Zero-sequence components consisting of three phasors equal in magnitude and with zero displacement from each other.

$$V_a = V_a^{(0)} + V_a^{(1)} + V_a^{(2)}$$

$$V_b = V_b^{(0)} + V_b^{(1)} + V_b^{(2)}$$

$$V_c = V_c^{(0)} + V_c^{(1)} + V_c^{(2)}$$

In matrix form: (a=1∠120°)

$$\begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & a^2 & a \\ 1 & a & a^2 \end{bmatrix} \begin{bmatrix} V_a^{(0)} \\ V_a^{(1)} \\ V_a^{(2)} \end{bmatrix} = A \begin{bmatrix} V_a^{(0)} \\ V_a^{(1)} \\ V_a^{(2)} \end{bmatrix}$$

In matrix form:

$$\begin{bmatrix} V_a^{(0)} \\ V_a^{(1)} \\ V_a^{(2)} \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \\ 1 & a & a^2 \\ 1 & a^2 & a \end{bmatrix} \begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} = A^{-1} \begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix}$$

Zero sequence components are never present in line voltages regardless of the degree of unbalance because the sum of line-to-line voltage phasors in a three-phase system is always zero.

[0487] Line currents into a delta connected circuit have no zero-sequence currents. For a delta circuit:

$$I_a^{(1)} = (1-a) I_{ab}^{(1)}$$

$$I_a^{(2)} = (1-a^2) I_{ab}^{(2)}$$

Line-to-line voltages in a star connected circuit have no zero-sequence components. For a star circuit:

$$V_{ab}^{(1)} = (1-a^2) V_{an}^{(1)}$$

$$V_{ab}^{(2)} = (1-a) V_{an}^{(2)}$$

In the absence of a neutral connection, zero-sequence currents are not present. Also, there cannot be any positive or negative sequence currents in the connections from neutral to ground.

[0488] Sequence networks are developed for various components of a power system like transmission line, transformer or synchronous machine based on these symmetrical components.

[0489] 4.3.1. SCDFT: Symmetrical Component Discrete Fourier Transform

[0490] The computational burden associated with the computation of the symmetrical components can be greatly reduced by the choice of sampling frequency. If the sampling frequencies that correspond to θ=30°, 60°, or 120° are used then the DFT and the symmetrical component calculation can be effectively combined. If θ=30° and Δ y_L=[y_{new}−y_{old}] (for full cycle algorithm) we can produce full-cycle SCFDT with these expressions:

$$Y_{oc}^{(L+1)} = Y_{oc}^{(L)} + (\Delta y_{a,L} + \Delta y_{b,L} + \Delta y_{c,L}) \cos(L\theta)$$

$$Y_{os}^{(L+1)} = Y_{os}^{(L)} + (\Delta y_{a,L} + \Delta y_{b,L} + \Delta y_{c,L}) \sin(L\theta)$$

$$Y_{1c}^{(L+1)} = Y_{1c}^{(L)} + \Delta y_{a,L} \cos(L\theta) + \Delta y_{b,L} \cos(L-4)\theta + \Delta y_{c,L} \cos(L+4)\theta$$

$$Y_{1s}^{(L+1)} = Y_{1s}^{(L)} + \Delta y_{a,L} \sin(L\theta) + \Delta y_{b,L} \sin(L-4)\theta + \Delta y_{c,L} \sin(L+4)\theta$$

$$Y_{2c}^{(L+1)} = Y_{2c}^{(L)} + \Delta y_{a,L} \cos(L\theta) + \Delta y_{b,L} \cos(L+4)\theta + \Delta y_{c,L} \cos(L-4)\theta$$

$$Y_{2s}^{(L+1)} = Y_{2s}^{(L)} + \Delta y_{a,L} \sin(L\theta) + \Delta y_{b,L} \sin(L+4)\theta + \Delta y_{c,L} \sin(L-4)\theta$$

[0491] 4.3.2 Power in Terms of Symmetrical Components

[0492] The total complex power (in volt-amperes) flowing into a three-phase circuit through the lines a, b and c is:

$$S_{3\Phi} = P + jQ = V_a I_a^* + V_b I_b^* + V_c I_c^* = 3 V_a^{(0)} I_a^{(0)*} + 3 V_a^{(1)} {}^* I_a^{(1)*} + 3 V_a^{(2)} I_a^{(2)*}$$

The transformation of a-b-c voltages and currents to symmetrical components is power-invariant.

[0493] 4.3.3 Symmetrical Component Distance Relay

[0494] The use of phasor calculations permits the use of symmetrical components in the detection of fault type and distance. For all possible fault types a general expression for the fractional distance to the fault is:

$$k = \frac{k_1 + k_2 k_2' + k_0 k_0'}{1 + k_0' + k_2' + k_1}$$

where all the quantities k (see FIG. **69**) can be calculated from the symmetrical components of the voltage and current phasors at one end of the transmission line.

[0495] 4.4. UIC Algorithm Development

[0496] 4.4.1. Problem Description

[0497] The schematic in FIG. **70** shows a typical structure of Gensets and UICs. Up to 8 distributed generation Gensets each via a circuit breaker (CB) are connected to a local bus. The functionality of a UIC is implemented in two modules, namely, Phasor Measure Unit (PMU) module and Main Utility Interface Controller (Main UIC or simply UIC) module.

[0498] Each CB in this embodiment is equipped with a PMU that implements the following functions:

[0499] Checking the status of the CB, e.g., determining if the CB is open or closed;

[0500] Sampling 3-phase currents flowing in the CB;

[0501] Sampling 3-phase voltages on both terminal sides;

[0502] Calculating current and voltage phasors;

[0503] Calculating frequencies on both terminal sides;

[0504] Buffering calculation results;

[0505] Communicating with Main UICs and sending results to them.

[0506] The Main UIC, an integrated part of the DG Genset controller, is helpful in many embodiments that are designed to satisfy the requirements of IEEE 1547. In the present embodiment, it implements the following two kinds of functions:

Slave functions:

[0507] Communicating with PMUs and retrieving current phasors, voltage phasors, frequencies, and CB statuses;

[0508] Sending commands to break/make the CB that is directly connected to the Genset controlled by its integrated DG controller; and

[0509] Receiving commands from the UIC performed as the master and executing the commands;

Master functions:

[0510] Communicating with PMUs and retrieving current phasors, voltage phasors, frequencies, and CB statuses;

[0511] Sending commands to break/make the CB that is directly connected to the Genset controlled by its integrated DG controller;

[0512] Sending commands to other UICs performed as slaves;

[0513] Performing under-/over-voltage protection;

[0514] Performing under-/over-frequency protection;

[0515] Performing fault detection and protection;

[0516] Timing synchronization and changing operation mode from isolated to parallel;

[0517] Detecting loss of synchronism and changing operation mode from parallel to isolated;

[0518] Performing other relaying functions required by IEEE 1547.

The communication protocol among Main UICs and PMUs is a CAN bus.

[0519] 4.4.2. Inputs and Outputs

[0520] Input signals to UIC include the following:

[0521] 3-phase currents, provided by Current Transformers (CTs), flowing through each CB;

[0522] 3-phase voltages, provide by Potential Transformers (PTs), appearing on both terminal sides of each CB;

[0523] Status of each CB.

Output signals from UIC include the following:

[0524] Break/Make signals to CBs;

[0525] Values need to be sent the human machine interface (HMI) or human computer interface (HCI) including currents, voltages, frequencies, etc.

[0526] 4.4.3. Programming Languages

[0527] Algorithms run on PMUs will be coded in PIC assembly language. Algorithms run on Main UICs will be coded in high level programming language, such as C, Matlab, etc.

[0528] 4.4.4. Logic Flow

[0529] The PMU's main logic flow is illustrated in the flowchart in FIG. **71**.

5. Prototype Design

[0530] 5.1. System Controller Design

[0531] Illustrated in context as a block diagram in FIG. **72**, the controller system consists of a number of PMUs (Phasor Measurement Unit) and a mainstream desktop computer with a PCI CAN card and a PCI I/O card. The PC will be the workhorse of the controller that undertakes all computing tasks of UIC (Utility Interface Controller) and the system controller. A PMU acts as a digital sensor that measures 3-phase currents or voltages. The PCI CAN card has two ports. One port is for local communications with PMUs in the same DG unit. The other is for communications between DG units as well as communications between DG unit and PMUs at Common Coupling Point (CCP). The PCI I/O card takes care of all other local inputs, such as oil pressure, and outputs, such as speed bias to Engine Control Module (ECM).

[0532] This exemplary embodiment of the hardware includes: 3.0 GHz, Pentium 4 processor with 800 MHz front side bus; 1 GB DDR2 RAM; 300 GB, 7200 rpm, 8 MB buffer SATA hard drive; 16×, double layer DVD±R/RW drive; 4 available PCI slots; 1 serial port, 1 parallel port, and 1 LAN port; 2 USB ports; Integrated audio and video; and 19-inch digital LCD. The PC's software in this embodiment includes: MS Windows XP Professional with SP2; MS Visual Studio .NET; and the Math Works Matlab with Simulink and an appropriate collection of toolboxes as will occur to those skilled in the art. Example PC hardware systems include DELL OPTIPLEX GX520 and the HP Compaq Business Desktop 5000 series.

[0533] CAN Cards in this embodiment use a PCI interface, provide 2 separate ports for data exchange, and a Windows DLL library. For example, the Advantech PCI-1680U-A: 2-Port CAN Interface Universal PCI Communication Card with Isolation works well.

[0534] The I/O Cards in this embodiment preferably have the following characteristics: A/D sampling rate of 720 samples per second per channel; A/D sampling resolution of 12-bit A/D converter; 3 A/D input channels; 16 digital input channels; 12 digital output channels; D/A resolution of 12-bit D/A converter; 2 D/A out put channels; and a corresponding Windows DLL library. In various embodiments, these I/O Cards include Advantech PCI-1710-B 100 KS/s, 12-bit High-gain PCI-bus Multifunction DAS Card; Advantech PCLD-8710-A Wiring Terminal Board for PCI-1710/1710HG; and/or Advantech PCL-10168 68-Pin SCSI Cable, 1 m.

[0535] The connection between EGCP-2's Inputs and Outputs and the PC-Based Controller's Inputs and Outputs in this embodiment as follows:

TABLE 5-1

PC-based solution wiring diagram

				PC-Based Controller	
		EGCP-2			# of
I/O	A/D	Description	Location	Counterpart	units
I	A	Input Power	1-2	N/A	
I	A	PT Inputs	40-48	PMUs + CAN card port	4 + 1
I	A	CT Inputs	89-94	PMUs + CAN card port	2 + 1
I	A	MPU (Speed)	70-72	A/D channel on I/O card	1
I	A	Process	86-88	N/A	
I	A	Coolant Temp	66-67	A/D channel on I/O card	1
I	A	Oil Pressure	68-69	A/D channel on I/O card	1
O	A	Speed Bias	73-75	D/A channel on I/O card	1
O	A	Voltage Bias	37-39	D/A channel on I/O card	1
I	D	Discrete Inputs	49-65	Digital input channel	16
O	D	Relay Outputs	5-35	Digital output channel	12
I/O	D	RS-485	76-80	CAN card port	1
I/O	D	RS-422	81-85	N/A	

[0536] 5.2. System Controller Design

[0537] 5.2.1 System Structure

[0538] The system controller includes following function modules

[0539] Supervisory control functions

[0540] DG Unit control functions

- **[0541]** UIC Control functions
- **[0542]** Local protection functions
- **[0543]** User Interface

The system platform is Microsoft Windows XP Professional. The software development platform is Microsoft Visual .Net. The main programming language is C++

[0544] The major control system runs on an event driven mechanism, wherein each module communicates with others through the event engine. The advantage of the approach is

- **[0545]** Minimize global variables to improve system reliability
- **[0546]** Modularize each sub system and make whole system scalable
- **[0547]** Easy for debugging

[0548] Due to the time constraints, the UIC module will run on an interrupt based module. The interrupt will activate 720 times per second, it will receive data from PMU and check whether they satisfy IEEE 1547. In case of emergency, an emergency handling module will be called to control the DG unit directly, otherwise, event information will be sent to the event engine.

[0549] As shown in FIG. **73**, there are several major functional blocks in the main system controller of the present embodiment: an event engine, HMI, supervisory control, engine control, generator control, local protection, and I/O card API. These functional blocks will control part of the hardware through the API to get inputs and send output information. These functional blocks communicate with each other by sending events and receiving events from the event engine.

[0550] 5.2.2 The Structure of the Event Engine and Sub Module

[0551] As mentioned before, different functional blocks communicate with each other through events. Following is a definition of the event.

```
struct sysEvent
{
    unsigned short sysPriority;
    unsigned short sysObjectFrom;
    unsigned short sysObjectTo;
    unsigned short sysCommand;
    union
    {
        struct
        {
          int iHigh;
          int iLow;
        } Integer;
        long longInfo;
        float floatInfo;
    } message;
};
```

[0552] The event engine is a FIFO event queue. Each module can push events to the queue, and the engine will send the events to every module to check.

```
class sysEventQueue
{
    protected:
        static sysEvent eventQueue[MAX_EVENTS];
        static int sysQueueHead, sysQueueTail;
```

-continued

```
    public:
        sysEventQueue ( );
        ~sysEventQueue ( );
        static void putEvent (sysEvent event);
        static void replaceEvent (sysEvent event)
            { eventQueue[sysQueueTail] = event; }
        static void getEvent (sysEvent& event);
        static int isEmpty ( );
};
```

[0553] Each submodule will be a class inherited from the class of sysObject. The major module in the class is the handleEvent function, the function that will execute normal monitor functions and communicate with other modules by sending events to the event engine and handling events from the engine.

```
class sysObject
{
    Private:
        unsigned short objectType;
        unsigned short objectID;
    public:
        sysObject ( ) { };
        ~sysObject ( ) { };
        virtual void handleEvent (sysEvent& event) { }
        virtual void clearEvent (sysEvent& event)
            { event.sysCommand = evNothing; }
        virtual void pushEvent (sysEvent& event)
            {
                sysEventQueue::putEvent (event);
                clearEvent (event);
            }
};
```

[0554] 5.3. Phasor Measurement Unit (PMU) Development

[0555] 5.3.1 Hardware

[0556] The Phasor Measurement Unit (PMU) acts as an interface that converts the standard substation measurements into digital measurements that can be processed by the controller. To attain a higher rate of data transmission between the PMU and the controller, a CAN bus will be the media connecting them. The circuit diagram is as shown in FIG. **77**.

[0557] The PMU control software was developed using assembly language and implemented in a PIC16C773 chip, which is shown in FIG. **74**. Specifications of this exemplary PMU include:

- **[0558]** Sampling rate: 720 samples per second per channel.
- **[0559]** Sampling resolution: 3 channel 12-bit A/D converters.
- **[0560]** 4K-Byte program memory and 256-Byte data memory.
- **[0561]** 2 Timers except watchdog timer.
- **[0562]** 1 CAN port.

[0563] 5.3.2. Testing of Prototype PMU

[0564] An SR 232 interface was developed to test the functionality of Phasor Measurement Unit (PMU). It includes an SP232 chip and four 0.1° F. capacitors. The purpose of introducing the interface into the test is to convert the CMOS outputs into the correct SR 232 signals. Thus, the computer can poll the outputs of PMU via its serial port. To poll data using a computer via its serial port is only a method to do testing. Some embodiments of the PMU do not include serial communication capabilities, implementing CAN communication instead. A block diagram of this subsystem is shown in FIG. **75**.

[0565] 5.3.3. PMU Software Re-Development

[0566] The PMU algorithm in one embodiment included a flaw, in that it did not consider accumulated errors caused by truncations and iterations. The new algorithm does its calculations recursively, which causes the accumulated errors, and non-recursively at the same time. After each cycle, the non-recursive results are copied to replace the recursive ones, thereby eliminating accumulated errors while still keeping the efficiency of the old algorithm.

[0567] 5.3.4 Converting Coefficients into Phasors and Frequencies

[0568] The outputs of PMU are sine and cosine coefficients. More calculations based on these coefficients are needed to yield phasors and frequencies. In various embodiments, this conversion is implemented using a Visual Basic program or using C++ consistent. The code for one embodiment is provided in Appendix B.

[0569] A diagram showing the frequencies vs. time is shown in FIG. **76**. The average of frequencies is very close to 60 Hz.

6. Additional Improvements

[0570] 6.1. System Controller

[0571] 6.1.1. Summary

[0572] The following activities have been accomplished:

[0573] Tested digital analog I/O board

[0574] Developed protection module and implemented PLC function in VC++

[0575] Developed engine/generator control module in VC++

[0576] Developed DG unit control function in VC++

[0577] Developing system controller user interface in C++

[0578] 6.1.2. Protection and PLC Function

[0579] Some protection functions were implemented in a PLC module in some embodiments because sufficiently low-level access to the system controller was unavailable. In another embodiment those functions were implemented in and will be incorporated into a new controller. The code for the PLC is attached in Appendix F.

[0580] 6.1.3. Engine and Generator Controller

[0581] Following is an example of a method of calculating. To calculate Voltage Bias

```
percent_ref = 10;          // +/- 10% of reference voltage
volt_range = 9;            // +/- 9 Vdc of voltage bias
ref_voltage = 480;         // normal voltage 480 Vac
detected_volt = 0;         // initial value
-----------------------------
volt_bias = 0;
max_volt = 0;
min_volt = 0;

 Find limits of volt

 $\text{max\_volt} = \frac{(100 + \text{percent\_ref})}{100} \times \text{ref\_voltage}$

 $\text{min\_volt} = \frac{(100 - \text{percent\_ref})}{100} \times \text{ref\_voltage}$
```

-continued

```
If (min_volt ≦ detected_volt ≦ max_volt)

    $\text{volt\_bias} = \frac{(\text{detected\_volt} - \text{ref\_voltage})}{\text{ref\_voltage}} \times \text{percent\_ref} \times \text{volt\_range}$

Elseif (detected_volt < min_volt)

    $\text{volt\_bias} = \frac{(\text{min\_volt} - \text{ref\_voltage})}{\text{ref\_voltage}} \times \text{percent\_ref} \times \text{volt\_range}$

Else

    $\text{volt\_bias} = \frac{(\text{max\_volt} - \text{ref\_voltage})}{\text{ref\_voltage}} \times \text{percent\_ref} \times \text{volt\_range}$

End
Return    volt_bias
```

To Calculate Speed Bias

[0582] The Throttle Position Sensor (TPS) provides a voltage signal that changes relative to the position of the throttle valve. Signal voltage will vary from less than 1.0 volt at idle to about 4.6 volts at wide open throttle (WOT). The TPS signal is an input used by the ECM for fuel control and for many of the ECM-controlled outputs.

```
kp = 1;                    // proportional gain
ki = 15;                   // Integral gain
imin = -100;               // minimum value before multiplying ki
imax = 100;                // maximum value before multiplying ki
kd = 0;                    // derivative gain
max_power = 85000;
min_power = 0;
speed_range = 4;
    // speed range and offset show speed bias from 0.5 to 4.5
offset = 0.5;
needed_power = 0;          // needed power from power management
present_power = 0;         // power that this generator is producing
freq = 60;                 // frequency
rad_speed = 0;             // omega rad/sec
ref_torque = 10;
feed_back_torque = 0;
error = 0;
pterm = 0.1;
istate = 0;
iterm = 0;
dterm = 0;
old_error = 0;
pid_result = 0;
instant_power = 0;
speed_bias = 0.5;
if (needed_power == 0)
        speed_bias = 0.5
else
        rad_speed = freq × 3.14159

        $\text{ref\_torque} = \frac{\text{needed\_power}}{\text{rad\_speed}}$

        $\text{feed\_back\_torque} = \frac{\text{present\_power}}{\text{rad\_speed}}$

// PID Control
        error = ref_torque - feed_back_torque
        //Calculate Proposition term
```

-continued

```
          pterm = kp × error;
          //Calculate Integral term
          if (istate > imax)
               istate = imax;
          if (istate < imin)
               istate = imin;
               iterm = ki * istate;
// some papers use iterm = ki * (istate + error)
// Calculate Derivative term
dterm = kd * (error − old_error)
// Remember old data
old_error = error;
istate = istate + error;
// PID output
pid_result = pterm + iterm + dterm;
//Calculate Power for Throttle position need
instant_power = (ref_torque + pid_result) × rad_speed;
// Check for security
If (instant_power > max_power)
               instant_power = max_power;
          if (instant power < min_power)
               instant_power = min_power;
          //Calculate Speed Bias

          $\text{speed\_bias} = \frac{\text{instant\_power}}{\text{max\_power}} \times \text{speed\_range} + \text{offset}$

End
Return Speed_bias
```

[0583] 6.1.4. Updated System Controller Schematics

[0584] The functional diagram of a system controller in FIG. **73** has also been applied in a second embodiment. There are seven major functional blocks in this main system controller: event manager, HMI, supervisory control, engine control, generator control, local protection, and I/O card API. These functional blocks control part of the hardware through the API to get inputs and send output information. These functional blocks communicate with each other by sending events to and receiving events from the event manager. The UIC communicates with the event manager through an event loop or an interrupt.

[0585] A test unit was placed in a test cell. A computer controller was connected to the unit through the digital and analog I/O card. Testing included three major tasks: I/O connection testing, functional module testing, and integrated system testing. I/O connection testing checked each input signal and each output signal to make sure that they were correctly connected and respond properly. Functional module testing tested each software module described in the previous section and performed functions as designed. Integrated system testing tested the system as a whole.

[0586] 6.2. Utility Interface

[0587] 6.2.1. Summary

[0588] The following research and development activities have been undertaken:

- **[0589]** Tested the stability/reliability of PMU by running it continuously for approximately 50 days.
- **[0590]** Developed CAN communication.
- **[0591]** Revised PMU Assembly code for the CAN communication.
- **[0592]** Programmed UIC functions in VC++.
- **[0593]** Compared compliance of existing technology with IEEE 1574
- **[0594]** Refined and tested the UIC functions.
- **[0595]** Constructed a PMU unit.
- **[0596]** Developed HIF detection technology

[0597] 6.2.2. Test of the Stability and Reliability of a Prototype PMU

Overview

[0598] Since the PMU units will be running continuously for months once put into use in practice, a prototype PMU had been put online to test its stability and reliability. The data the PMU acquired were transmitted to a PC via SR 232 communication. A VB program running on the PC converted data into meaningful variables with time stamps and recorded them on the hard drive.

Test Results

[0599] Analyzing a small portion of the recorded frequencies showed that there were no frequencies that were apparently absurd. This initial result indicated that the PMU worked properly during the test. It also indicated that the PMU assembly code is stable and reliable.

[0600] 6.2.3. Development of CAN Communication

Overview

[0601] In this embodiment, the Phasor Measurement Unit (PMU) acts as an interface that converts the standard substation measurements into digital measurements that can be processed by the controller. To attain a higher rate of data transmission between the PMU and the controller, a CAN bus is used as the media connecting them.

Schematics

[0602] The circuit diagram is shown in FIG. **78**.

Code for CAN Communication

[0603] CAN communication is implemented using PIC18F4580 chips. MCP2551 is used as CAN transceiver. The assembly code running on the PIC18F4580 for CAN communication is shown in Appendix D.

[0604] 6.2.4. Revision of PMU Assembly Code for CAN

Overview

[0605] After successful testing of the functionality of Phasor Measurement Unit (PMU) with RS-232 interface, PMU with CAN bus interface had been developed as a final design. In this design two PMU units are connected to the PC via a CAN bus.

[0606] PMU sends calculated phasors to a CAN chip using Parallel Slave Port (PSP). We used PSP instead of serial posts to achieve fast data communication. The CAN chip sends this data to PC using CAN bus. A CAN transceiver is used between the CAN chip and the CAN bus to convert voltage levels. Synchronization between the PMUs is a challenging task in this project. To achieve proper synchronization, one PMU works as the master PMU and all others (in current case only one) are slave PMUs. The master PMU will send synchronization pulses to slave PMU and slave PMU will use this pulse for synchronization.

[0607] After analyzing data transmitted to PC, we can see good synchronization between both PMU. Results will be better after using filters.

Schematics

[0608] The circuit diagram is shown in FIG. **79**.

Code

[0609] The current version is developed to address the practical CAN communication. Since more than one PMU shares the same CAN channel, the first revision is to incorporate an id number into the data each PMU sends to the computer. Therefore, the PC can distinguish and assemble data correctly. Since we also need to make all PMUs sample at substantially the same instant, the second revision is to introduce a master PMU and let all others be slave PMUs. The master PMU will send synchronization signals to slave PMUs to guarantee the substantial simultaneity of corresponding samples on all PMUs. This improves the operation of the UIC and other control functions. The code is attached in Appendix B.

[0610] 6.2.5. Programming of UIC Functions in VC++

[0611] The UIC functions were programmed for simulation using Matlab, then were rewritten in VC++. The new version also generalized some cases and provided many default parameters. The code is in Appendix E.

[0612] 6.2.6. Comparison of Compliance of Basler and Woodward Productions with IEEE 1574

[0613] One current system uses a Woodward EGCP-2 as the Genset controller and a Basler relay as the UIC. The newly developed integrated controller meets the IEEE 1574 requirements into the product.

[0614] The following table compares the Basler relay's characteristics and requirements of IEEE 1574.

Requirement	IEEE 1547 Standard	Basler Relay

Synchronization	Δf = 0.3 Hz
	ΔV = 10%
	Δø = 20 degrees
		Δf = 0.01-0.5 Hz
		ΔV = 1-20 V
		Δø = 1-45 degrees

Voltage disturbances	V < 160 → 0.16 sec 60 < V < 106 → 2 sec 132 < V < 144 → 1 sec 144 < V → 0.16 sec	Pickup: 10-300 V Time delay: 0.05-600 sec
Frequency disturbances	F > 60.5 → 0.16 sec 59.3 > F > 57.0 → user F < 57 → 0.16 sec	Pick up: 40-70 Hz Time delay: 0-600 sec
Disconnection of faults	Overcurrent protection and ground fault protection	Time overcurrent protection available with pick up to drop out ratio = 95%
Surge withstanding capability	Comply with conditions of ANSI C62.41 or IEEE C37.90.1	Complies with IEEE C37.90.1
Influence of EMI	Not influenced by EMI	Qualified to C37.90.2
Prevention of unintentional islanding		Has a reverse power protection which can be used to prevent islanding
Inadvertent energization	Area EPS Voltage and frequency must be in the range 106 V-132 V and 59.3 Hz-60.5 Hz. Reconnection can be delayed up to 5 minutes after EPS has regained normalcy	Recloses between 100 milliseconds to 600 seconds
Measurement of Voltage current and Frequency	All the 3 phases need to be measured	All 3 phases are measured

The following table shows the comparison of Woodward EGCP-2's characteristics and IEEE 1574

-continued

Requirement	IEEE 1547 Standard	EGCP2

Synchronization	Δf = 0.3 Hz
	ΔV = 10%
	Δø = 20 degrees
		Δf = 0.01-0.5 Hz
		Volt window = 0.1-10% V
		Default Voltage: 10%
		Δø = 2-20 degrees
		Default Δø = 10 degrees

Voltage disturbances	V < 160 → 0.16 sec 60 < V < 106 → 2 sec 132 < V < 144 → 1 sec 144 < V → 0.16 sec	Pickup: 50-30000 V Default Pickup(high): 250 V Default Pickup(low): 200 V Time delay: 0.1-30 sec Default time delay: 5 sec Gen volt Hi limit: 10% above rated Gen volt low limit: 10% below rated
Frequency disturbances	F > 60.5 → 0.16 sec 59.3 > F > 57.0 → user F < 57 → 0.16 sec	Pickup: 40-75 Hz Time delay: ? Default Pickup(High): 65 Hz Default Pickup(low): 55 Hz
Disconnection of faults	Overcurrent protection and ground fault protection	Pick up: 5-30000 A Default pickup: 30 A Delay: 0.1-20 sec Default delay: 1 sec
Surge withstanding capability	Comply with conditions of ANSI C62.41 or IEEE C37.90.1	Complies with IEEE C37.90.1
Influence of EMI	Not influenced by EMI	Qualified to C37.90.2
Prevention of unintentional islanding		No islanding feature
Measurement of Voltage current and Frequency	All the 3 phases need to be measured	Only 1 phase of the utility is measured and 3 phases of the Genset is measured
Voltage Regulation	Must not degrade customer voltage beyond + or − 5% of nominal voltage	Does not degrade voltage beyond 2-5% of nominal voltage

[0615] 6.2.7. Test and Refinement of the UIC Functions

[0616] In this experimental embodiment, the overcurrent relay function has been integrated into our main UIC function. The development of ground fault detection function is being developed as a standalone relay function independent from main UIC functionality.

[0617] 6.2.8 Development of HIF Detection Technology

[0618] High Impedance Fault (HIF) is a concern in the implementation of DG, even though it has not yet been officially addressed by the IEEE Standard 1547. An HIF condition occurs when a device (and, perhaps, a portion of the utility grid) becomes electrically isolated from the remainder of the grid. When the device (or portion of the grid) is not electrically isolated from the remainder of the grid, the device is in a "non-HIF" (non-high-impedance-fault) condition. For public safety and the potential huge expenses resulting from an energized downed conductor, DG operators should pay some attention to HIFs. We have developed a Decision Tree (DT) based technology that addresses this concern. It showed excellent performance in a simulation study. FIG. **80** shows a one-line system diagram based on which HIF-related simulations have been performed.

[0619] With the integrated controller, the Decision Tree (DT) based detection technology can use a simplified version of PMU to provide raw current data samples. DTs are typically trained off-line from simulated HIF and non-HIF data first, then are tested and trained using experimental data to further improve performance. The DT for use with the present system uses only current signals measured at Point of Common Coupling (PCC), though other electrical characteristics are used in alternative embodiments. Current harmonics are also used to train the DTs. The DTs apply the training to real data that characterizes the current (or other characteristic) at the PCC to determine whether or not a HIF condition exists, and provides an HIF Detection Output as a function of that determination. The DGSC receives the HIF Detection Output and modifies operation of the DG system accordingly, as discussed herein. Simulations of this subsystem were done with the aid of the Electromagnetic Transients Program (EMTP).

[0620] While the system and methods have been illustrated and described in detail in the drawings and accompanying description, they are to be considered as illustrative and not restrictive in character, it being understood that the preferred embodiment has been shown and described and that changes and modifications that come within the spirit of the invention are desired to be protected.

Appendix A to Part 2: Source Code for CHECKING CONFORMANCE WITH IEEE 1547

[0621]

```
% Function BTRIP implements IEEE 1547 requirements:
% over/under voltage relays, over/under frequency relays
% synchronization check relays, inadvertent energization relays
% Functions in Task 3.1 remain to be implemented
% reverse power relays, fault current detection relays
% Function BTRIP is called every new sample
% Function BTRIP=1 means breaker ordered open
% Function BTRIP=0 means breaker permitted closed
% input Vars:
% VMAREA, VMLOCAL (formerly V)
% FAREA, FLOCAL
% VPHAREA, VPHLOCAL
% IM, IPH
% BOPENSTAT (BOPENSTAT=1 means breaker open)
% output vars:
% BOPEN (BOPEN=1 means breaker ordered open)
% relay functions:
% over/under voltage relays ---- VMLOCAL
% over/under frequency relays ---- FLOCAL (use time-threshold table)
% reverse/min power relays ---- VMLOCAL, VPHLOCAL, IM, IPH
% (threshold=0, time=0.5s)
% synchronization check relays ---- VM, F, VPH (both sides) (table,
time=0.1s)
% fault current detection relays ----- IM
BTRIP(args)      % gets called every new sample
% args include all of above plus NEWSTART
if (NEWSTART==1) % initialize counters
% casevd1=Voltage Disturbance, casefd1=Freq Disturbance
% casesc1=Synch Check, caseie1=Inadvertent Energization
casevd1=0; casevd2=0; casevd3=0; casevd4=0;
casefd1=0; casefd2=0; casefd3=0;
casesc1=0; caseia1=0;
end
if (BOPENSTAT==0) % breaker closed
%   check stuff, set BOPEN=1 if violation
% OVER/UNDER VOLTAGE RELAY LOGIC
% Voltage Range (Volts-RMS value)    Clearing Time(sec.)
%    V<60                            0.16 % ten cycles (120 samples)
%    V<106                           2.0  % two cycles (1440 samples)
%    132<V<144                       1.0  % one cycle  (720 samples)
%    144=<V                          0.16
if (VMLOCAL<60)
     casevd1=casevd1+1;
else
     casevd1=0;
end
if (VMLOCAL <106)
     casevd2=casevd2+1;
```

-continued

```
else
     casevd2=0;
end
if (VMLOCAL >132 & VMLOCAL <144)
     casevd3=casevd3+1;
else
     casevd3=0;
end
if (VMLOCAL >=144)
     casevd4=casevd4+1;
else
     casevd4=0;
end
BOPEN=(casevd1>120 | casevd2>1440 | casevd3>720 | casevd4>120);
% OVER/UNDER FREQUENCY LOGIC
if (FLOCAL>57.0&FLOCAL<59.3)
     casefd1=casefd1+1;
else
     casefd1=0;
end
if (FLOCAL>60.5)
     casefd2=casefd2+1;
else
     casefd2=0;
end
if (FLOCAL<57.0)
     casefd3=casefd3+1;
else
     casefd3=0;
end
BOPEN=(case1>120 | case2>120 | case3>120 | case4>120 | case5>120);
end
if (BOPENSTAT==1) % breaker open
% SYNCH CHECK AND INADVERTANT ENERGIZATION LOGIC
%   check synchronism and VMAREA, set BOPEN=0 if OK to close
%measure the voltage magnitude on both sides of Circuit breakers CB1,
%CB2 ... CB8
%let voltage magnitude measured on the EPS side be VMAREA and that
%measured on DG side be VMlocal
%Measure the Voltage Phase on both sides of Circuit breakers
%CB1,CB2....CB8 let voltage phase measured on the EPS side be
%VPHAREA and that measured on DG side be VPHLOCAL
%To check for synchronisation we need to do the following:
% 1. (|VMLOCAL−VMAREA|/120)*100 must not be greater than 10
% 2. (|FLOCAL−FAREA|) must not be greater than 0.3
% 3. (|VPHLOCAL−VPHAREA| must not be greater than 20)
%It is checked if the parameters are maintained in the their respective
%ranges for 0.1 sec=72 samples
% VMAREA constraints for inadvertent energization also required for
72 samples
Vdiff=(abs(VMLOCAL−VMAREA)/120)*100;
VPHdiff=abs(VPHLOCAL−VPHAREA);
Fdiff=abs(FLOCAL−FAREA);
if (VMAREA>106 & VMAREA<132 & FAREA>59.5 & FAREA<60.5)
     caseie1 =min(caseie1+1,1000);
else
     caseie1=0;
end
if (Vdiff<10 & VPHdiff<20 & Fdiff<0.3)
     casesc1 =min(casesc1+1,1000);
else
     casesc1=0;
end
if(casesc1>72 & caseie1>72)
BOPEN=0;
end
return BOPEN
```

Appendix B to Part 2: Source code of PMU

[0622] The current version is developed to address the practical CAN communication. Since there are more than one

PMU sharing the same CAN channel, the first revision is to incorporate an id number into the data each PMU sends to the computer. Therefore, the PC can distinguish and assemble data correctly. Since we also need to make all PMUs sample at substantially the same instant, the second revision is to introduce a master PMU and let all others be slave PMUs. The master PMU will send synchronization signals to slave PMUs to guarantee the substantial simultaneity of all samples on all PMUs. This might be done, for example, in order to improve the operation of the UIC and other control functions.

[0623] The assembly code for a master PMU is as follows.

[0624] The assembly code for a slave PMU is as follows.

[0625] A program coded in VC++ running on the PC to convert data into the correct format for this embodiment is as follows.

```
// data__logger.cpp : Defines the entry point for the console application.
//#include "windows.h"
#include "stdafx.h"
#include "..\..\..\include\CanBus2.0.h"
#include "stdio.h"
#include "math.h"
#include <iostream>
#include <fstream>
using namespace std;
typedef struct
{
    UCHAR rtr;      //Remote Transmit Request
    ULONG id;
    UCHAR dlen;
    UCHAR data[8];
} CAN__MSG__T, *PCAN__MSG__T;
PCAN__MSG__T g__pData;
ULONG g__Index = 0;
int __tmain(int argc, __TCHAR* argv[ ])
{
      ofstream myfile;
      UINT m__DevNum ;
      UINT m__nBtr0 = 0x03;
      UINT m__nBtr1 = 0x1c; //baud rate = 125k
      UINT m__nAcpCode = 0;
      UINT m__nAcpMask = 0xff;
      UINT m__nOutCtrlCode = 250;
      UINT m__nIntMask = 0;
      UINT m__nPort;
      UINT m__nHostID;
      UINT m__nPreBaudRate;
      BOOL bReady = FALSE;
      unsigned int nData0;
      unsigned int nData1;
      unsigned int nData2;
      unsigned int nData3;
      int    sWord__A,cWord__A,sWord__B,cWord__B,sWord__C,cWord__C;
      double aSin;
      double aCos;
      long double aAmp,aAng,bAmp,bAng,cAmp,cAng;
    std::cout <<    "Enter Device number 3 or 4 \n";
    std::cin >> m__DevNum;
        myfile.open ("data.txt");
      if (CANPortOpen(
          m__DevNum,
          (WORD *)&m__nPort,
          (WORD *) &m__nHostID,
          (WORD *) &m__nPreBaudRate) != SUCCESS)
      {
          printf("Open port failed!\n");
          return 0;
      }
      if (CANInit(
          m__nPort,
          m__nBtr0,
          m__nBtr1,
           (UCHAR) m__nIntMask) != SUCCESS)
      {
          printf("Init can failed!\n");
          CANPortClose(m__nPort);
          return 0;
      }
      if (CANSetOutCtrl(
          m__nPort,
          m nOutCtrlCode) != SUCCESS)
```

-continued

```
        {
              printf(“Set out ctrl failed!\n”);
              CANPortClose(m_nPort);
              return 0;
        }
        if (CANSetAcp(
             m_nPort,
             m_nAcpCode,
             m_nAcpMask) != SUCCESS)
        {
             printf(“Set acp code failed!\n”);
             CANPortClose(m_nPort);
             return 0;
        }
        if (CANSetBaud(
             m_nPort,
             m_nBtr0,
             m_nBtr1) != SUCCESS)
        {
             printf(“Set baudrate failed!\n”);
             CANPortClose (m_nPort);
             return 0;
        }
        if (CANSetNormal(m_nPort) != SUCCESS)
        {
             printf(“Set normal failed!\n”);
             CANPortClose(m_nPort);
             return 0;
        }
        CANEnableRxInt( m_nPort );        //enable receive interrupt
        g_pData = ( PCAN_MSG_T )GlobalAlloc( GPTR, 10000*sizeof(CAN_MSG_T) );
        if( g_pData == NULL )
        {
           printf(“Out of memory!\n”);
           CANPortClose(m_nPort);
           return 0;
       }
      // lock the memory
      GlobalLock( g_pData );
      CANSetBufferPtr( m_nPort, g_pData, 10000, &bReady );// set the receive
buffer
      if ( !bReady )
      {
           printf(“Again CanSetBufferPtr failed!\n”);
           CANPortClose(m_nPort);
           return 0;
      }
      CANSetCountPtr( m_nPort, &g_Index, &bReady );
      if ( !bReady )
      {
       printf (“CAN set CountPtr error!”);
      }
      CANEnableEvent( m_nPort, TRUE ); //enable system event
      printf(“Open port successful, Begin to receive data!\n”);
      bReady = FALSE;
      CANWaitForFIFOEvent( m_nPort, INFINITE, &bReady);// wait for the
messages
      if ( bReady )
      {
           printf(“Receive Data Buffer half full!\n” );
      }
      bReady = FALSE;
      CANWaitForFIFOEvent( m_nPort, INFINITE, &bReady);
      if ( bReady )
      {
           printf( “Receive Data Finished!Below is data:\n”);
          for (int i = 0; i < 10000; i++ )
          {
          nData0=g_pData[i].data[0];
          nData1=g_pData[i].data[1];
          nData2=g_pData[i].data[2];
          nData3=g_pData[i].data[3];
```

-continued

```
          if(g__pData[i].id == 419)
           {  if(nData1 > 127) // Negative Value
                  sWord__A = −((255−nData1)*256+(255−nData0)+1);
              else
                  // positive values
                  sWord__A = nData1* 256 + nData0;
          if(nData3 > 127) // Negative Value
                  cWord__A = −((255−nData3)*256+(255−nData2)+1);
              else
                  // positive values
                  cWord__A = nData3 * 256 + nData2;
           aSin = static__cast<double>(sWord__A * 5)/ 4096 ;
           aCos = static__cast<double>(cWord__A * 5)/ 4096;
           aAmp = (sqrt( aSin*aSin + aCos*aCos ))/6;
           if (aCos == 0 )
               if (aSin >= 0 )
               aAng = 90;
               else
               aAng = −90;
           else if (aCos > 0)
           aAng = atan(aSin / aCos) * 180 / 3.14159265;
           else
               if (aSin >= 0)
                  aAng = 180 + atan(aSin / aCos) * 180 / 3.14159265;
               else
                  aAng = (atan(aSin / aCos) * 180 / 3.14159265) − 180;
           cout << “amplitude of Phase A is ” << aAmp << endl;
           cout << “angle of Phase A is ” << aAng << endl;
           myfile << i <<“ \t”<< “PMU1 amplitude” <<“ ” << aAmp << “\t” <<“
PMU1 angle” <<“ ”<< aAng<<“\n”;
           // myfile << “\t” <<    nData0 << “\t” <<    nData1<< “\t” <<
nData2 << “\t” <<    nData3 << “\t” <<       sWord__A << “\t” <<    cWord__A << “\t”<<
aSin << “\t ”<< aCos << “\t ”<< aAmp<< “\n”;
}
//-----------------
            if(g__pData[i].id == 427)
             {   if(nData1 > 127) // Negative Value
                  sWord__A = −((255−nData1)*256+(255−nData0)+1);
              else
                  // positive values
                  sWord__A = nData1* 256 + nData0;
            if(nData3 > 127) // Negative Value
                  cWord__A = −((255−nData3)*256+(255−nData2)+1);
               else
                  // positive values
                  cWord__A = nData3 * 256 + nData2;
            aSin = static__cast<double>(sWord__A * 5)/ 4096 ;
            aCos = static__cast<double>(cWord__A * 5)/ 4096;
            aAmp = (sqrt( aSin*aSin + aCos*aCos ))/6;
            if (aCos == 0 )
               if (aSin >= 0 )
               aAng = 90;
               else
              aAng = −90;
           else if (aCos > 0)
           aAng = atan(aSin / aCos) * 180 / 3.14159265;
           else
           if (aSin >= 0)
               aAng = 180 + atan(aSin / aCos) * 180 / 3.14159265;
           else
               aAng = (atan(aSin / aCos) * 180 / 3.14159265) − 180;
           cout << “amplitude of Phase A is ” << aAmp << endl;
           cout << “angle of Phase A is ” << aAng << endl;
           myfile << i <<“ \t”<< “PMU2 amplitude” <<“ ” << aAmp << “\t” <<“
 PMU2 angle” <<“ ”<< aAng<<“\n”;
          // myfile << “\t” <<    nData0 << “\t” <<    nData1<< “\t” <<    nData2
 << “\t” <<    nData3 << “\t” <<       sWord__A << “\t” <<    cWord__A << “\t”<< aSin
 << “\t ”<< aCos << “\t ”<< aAmp<< “\n”;
            }
           }
     }
     myfile.close( );
     CANPortClose(m__nPort);
     return 0;
}
```

Appendix C to Part 2: Source Code for Calculating Phasors and Frequencies

[0626]

```
Public gNum As Integer
Public dAng1 As Double
Public dAng2 As Double
Public dAng3 As Double
Public dAng4 As Double
Public dAng5 As Double
Public dAng6 As Double
Public gInd As Integer
Private Sub cmdStop_Click( )
Close #1
MsgBox “Stopped”
    Unload Me
End Sub
Private Sub cmdStart_Click( )
‘ Fire Rx Event Every Four Bytes
MSComm1.RThreshold = 4
‘ When Inputting Data, Input 4 Bytes at a time
MSComm1.InputLen = 4
‘ Optional settings: 9600 Baud, No Parity, 8 Data Bits, 1 Stop Bit
‘ Preferred settings: 115200 Baud, No Parity, 8 Data Bits, 1 Stop Bit
MSComm1.Settings = “115200,N,8,1” ‘Set baud rate value for the chip.
‘MSComm1.Settings = “9600,N,8,1” ‘Set baud rate value for the chip.
‘ Disable DTR
MSComm1.DTREnable = False
‘ Open COM1
MSComm1.CommPort = 1
MSComm1.PortOpen = True
cmdStop.Enabled = True
Shape1.FillColor = vbRed
‘ Initialize gNum
gNum = 1
‘ Initialize gInd
gInd = 1
‘ Initialize angles
dAng1 = 0
dAng2 = 0
dAng3 = 0
dAng4 = 0
dAng5 = 0
dAng6 = 0
End Sub
Private Sub crtCommand_Click( )
On Error GoTo FileOpenError
If txtFilename.Text = “” Then
    MsgBox “Please Enter the File name .”
Else
     Open “C:\pmu\vikram\” & txtFilename.Text & “.txt” For Output As #1
     cmdStart.Enabled = True
End If
 Exit Sub
FileOpenError:
        MsgBox “There was a problem in creating the file.”
        End
End Sub
Private Sub MSComm1_OnComm( )
Dim sData As String                          ‘ Holds incoming data
Dim sHighByte As Byte                        ‘ Holds sin HighByte
value
Dim sLowByte As Byte                         ‘ Holds sin LowByte
value
Dim sWord As Long                            ‘ Holds sin result
Dim cHighByte As Byte                        ‘ Holds cos HighByte
value
Dim cLowByte As Byte                         ‘ Holds cos LowByte
value
Dim cWord As Long                            ‘ Holds cos result
Dim sSin As Double                           ‘ Holds sin coeff
Dim cCos As Double                           ‘ Holds cos coeff
Dim sAmp As Double
Dim sAng As Double
Dim sFre As Double
```

-continued

```
    Select Case MSComm1.CommEvent
    ‘ Handle each event or error by placing
    ‘ code below each case statement
    ‘ Errors
      Case comEventBreak                                ‘ A Break was received.
        Write #1, Format(Now, “mm/dd/yyy - hh:mm:ss”),   “Error: A Break was
received.”
      Case comEventFrame                                ‘ Framing Error
        Write #1, Format(Now, “mm/dd/yyy - hh:mm:ss”),   “Error: Framing Error.”
      Case comEventOverrun                              ‘ Data Lost.
        Write #1, Format(Now, “mm/dd/yyy - hh:mm:ss”),   “Error: Data Lost.”
      Case comEventRxOver                               ‘ Receive buffer
overflow.
        Write #1, Format(Now, “mm/dd/yyy - hh:mm:ss”),   “Error: Receive buffer
overflow.”
      Case comEventRxParity                             ‘ Parity Error.
        Write #1, Format(Now, “mm/dd/yyy - hh:mm:ss”),   “Error: Parity Error.”
     Case comEventTxFull                                ‘ Transmit buffer full.
        Write #1, Format(Now, “mm/dd/yyy - hh:mm:ss”),   “Error: Transmit buffer
full.”
      Case comEventDCB                                  ‘ Unexpected error
retrieving DCB
        Write #1, Format(Now, “mm/dd/yyy - hh:mm:ss”),   “Error: Unexpected
error retrieving DCB.”
    ‘ Events
      Case comEvCD                                      ‘ Change in the CD line.
        Write #1, Format(Now, “mm/dd/yyy - hh:mm:ss”),   “Event: Framing Error.”
      Case comEvCTS                                     ‘ Change in the CTS
line.
        Write #1, Format(Now, “mm/dd/yyy - hh:mm:ss”),   “Event: Change in the
CTS line.”
      Case comEvDSR                                     ‘ Change in the DSR
line.
        Write #1, Format(Now, “mm/dd/yyy - hh:mm:ss”),   “Event: Change in the
DSR line.”
      Case comEvRing                                    ‘ Change in the Ring
Indicator.
        Write #1, Format(Now, “mm/dd/yyy - hh:mm:ss”),   “Event: Change in the
Ring Indicator.”
      Case comEvReceive                                 ‘ Received RThreshold #
of chars.
        sData = MSComm1.Input                           ‘ Get data (4 bytes)
        cHighByte = Asc(Mid$(sData, 4, 1))              ‘ get 1st byte ASCII
value
        cLowByte = Asc(Mid$(sData, 3, 1))               ‘ Get 2nd byte ASCII
value
        sHighByte = Asc(Mid$(sData, 2, 1))              ‘ get 3rd byte ASCII
value
        sLowByte = Asc(Mid$(sData, 1, 1))               ‘ Get 4th byte ASCII
value
        ‘ Combine bytes into a word for sin
        If sHighByte > 127 Then
            ‘ negative values
            sWord = −((255 − sHighByte) * 256 + (255 − sLowByte) + 1)
        Else
            ‘ positive values
            sWord = sHighByte * 256 + sLowByte
        End If
        ‘ Combine bytes into a word for cos
        If cHighByte > 127 Then
            ‘ negative values
            cWord = −((255 − cHighByte) * 256 + (255 − cLowByte) + 1)
        Else
            ‘ positive values
            cWord = cHighByte * 256 + cLowByte
        End If
        sSin = sWord * 5 / 4096
        cCos = cWord * 5 / 4096
        sAmp = (Sqr(sSin * sSin + cCos * cCos)) / 6
        If cCos = 0 Then
            If sSin >= 0 Then
                sAng = 90
            Else
                sAng = −90
            End If
```

-continued

```
        ElseIf cCos > 0 Then
            sAng = Atn(sSin / cCos) * 180 / 3.14159265
        Else
            If sSin >= 0 Then
                sAng = 180 + Atn(sSin / cCos) * 180 / 3.14159265
            Else
                sAng = Atn(sSin / cCos) * 180 / 3.14159265 − 180
            End If
        End If
        ‘ Calculate frequency
        If gInd = 1 Then
            sFre = 60 − (sAng − dAng1) / 3
            dAng1 = sAng
        ElseIf gInd = 2 Then
            sFre = 60 − (sAng − dAng2) / 3
            dAng2 = sAng
        ElseIf gInd = 3 Then
            sFre = 60 − (sAng − dAng3) / 3
            dAng3 = sAng
        ElseIf gInd = 4 Then
            sFre = 60 − (sAng − dAng4) / 3
            dAng4 = sAng
        ElseIf gInd = 5 Then
            sFre = 60 − (sAng − dAng5) / 3
            dAng5 = sAng
        Else
            sFre = 60 − (sAng − dAng6) / 3
            dAng6 = sAng
            gInd = 0
        End If
        gInd = gInd + 1
        If gNum >= 12 Then
‘               Write #1, Format(Now, “mm/dd/yyyy - hh:mm:ss”), Format(sAmp,
“0.000000”), Format(sAng, “000.0000”), Format(sFre, “000.00”)
            Write #1, sFre
            gNum = 1
        Else
            gNum = gNum + 1
        End If
      Case comEvSend    ‘ There are SThreshold number of
                        ‘ characters in the transmit
                        ‘ buffer.
        Write #1, Format(Now, “mm/dd/yyy - hh:mm:ss”), “Event: There are
SThreshold number of characters in the transmit buffer.”
      Case comEvEOF     ‘ An EOF charater was found in
                        ‘ the input stream
        Write #1, Format(Now, “mm/dd/yyy - hh:mm:ss”), “Event: An EOF charater
was found in the input stream.”
    End Select
End Sub
```

Appendix D to Part 2: Code for CAN Communication

[0627] CAN communication is implemented using PIC18F4580 chips. MCP2551 is used as CAN transceiver. The assembly code running on the PIC18F4580 for CAN communication is as follows.

```
list p=18f4580
list n=0                                ; list directive to define processor
radix dec
#include <p18f4580.inc>                 ; processor specific variable definitions
CONFIG      OSC=HS,PWRT=ON,BOR=OFF, BORV=20, WDT = OFF, WDTPS = 1,LVP =
OFF,STVREN = OFF,PBADEN=OFF
        ;* baudrate and other parameters
        ; READ DATA SHEET AND CAN DOCUMENTS FOR SETTINGS.
        ;DO NOT CHANGE THESE ,IF NOT SURE.
        ; Baud Rate 1 Mbits/sec
```

-continued

```
     BRGCON1_CONST               EQU 0xc0;0x04; SJW = 1 Tq, BRP = 4
     BRGCON2_CONST               EQU 0x9a; Seg2 freely programmable,
                                                                  ;1 samples/bit,Ph1=7 Tq, Prop
seg = 6 Tq
     BRGCON3_CONST               EQU 0x01; Seg2 = 2 Tq ---> Sample at 80%
     START_CAN                                  EQU 0x03; request normal mode
        TXB0CON_SEND                     EQU 0x0B; requests that TXB0 be sent on CAN bus
     DLC_CONST                                EQU 0x04; value for DLC field of message
     CIOCON_CONST                     EQU 0x20; value to load into CIOCON
                                 ; Drive TX pin to Vdd, no CAN Capture function on RC2
                                                 ; Register Variables
        CBLOCK 0x020
        var1:1
            var2:1
            var3:1
        ENDC
        org 0000h                                  ;Reset Vector
        goto              Start
     org 0008h                                       ;interrupt vector
        goto int_ser
Start
        LFSR 0, 100h
        clrf CCPR1H
        clrf CCPR1L
        clrf CCP1CON
        ;disable following feature to avoid conflict with CAN or PSP
        clrf ECCPR1H
        clrf ECCPR1L
        clrf ECCP1CON
        clrf ECCP1DEL
        clrf ECCP1AS
        movlw      0x07
        movwf      CMCON
        movlw      0x0f
        movwf      ADCON1
        movlw                        b'11111011'                                        ; RB2 is output
(CANTX)
        movwf                        TRISB
        movlw      0xFF
        movwf      TRISD                      ;Set Port_D to all inputs
        CLRF       PORTD
     movlw         b'00010110'            ;Set RD, WR, and CS as
        movwf      TRISE                        ; inputs, Enable Parallel Slave port
           bsf         LATE,0
      movlw        0x00
      movwf        TRISC                      ;Set Port_C to all outputs
        CLRF     WDTCON
; initialize CAN for TX Buffer 0
        movlw      CIOCON_CONST
        movwf      CIOCON
      movlw        0x88                          ;Configuration mode with buffer 0
selected.
        movwf    CANCON
  check1:
        btfss            CANSTAT, OPMODE2                       ;select configuration mode
        bra              check1
        movlw      BRGCON1_CONST
        movwf      BRGCON1
        movlw      BRGCON2_CONST
        movwf      BRGCON2
        movlw      BRGCON3_CONST
        movwf      BRGCON3
        movlw      START_CAN
        movwf      CANCON          ; start CAN in Normal Mode
        ;Standard Identifiers 11-bits.
; set up TX Buffer 0 with Identifier for 301 H= 0x25, L = 0xA0; 201 H=0x19,
L=0x20; 101 H=0x0C, L=0xA0;
        BANKSEL TXB0CON
        movlw 0x19
        movff WREG, TXB0SIDH
        movlw 0x20
        movff WREG, TXB0SIDL
        movlw 0x08
        movff WREG, TXB0DLC
        ; set up TX Buffer 1 with identifier 302 H= 0x25, L = 0xC0; 202 H=0x19,
```

-continued

```
L=0x40; 102 H=0x0C, L=0xC0;
      BANKSEL TXB0CON
      movlw 0x19
      movff WREG, TXB1SIDH
      movlw 0x40
      movff WREG, TXB1SIDL
      movlw DLC__CONST
      movff WREG, TXB1DLC
      ; set up TX Buffer 2 with identifier 303 H= 0x25, L = 0xE0; 203 H=0x19,
L=0x60; 103 H=0x0C, L=0xE0;
      BANKSEL TXB0CON
      movlw 0x19
      movff WREG, TXB2SIDH
      movlw 0x60
      movff WREG, TXB2SIDL
      movlw DLC__CONST
      movff WREG, TXB2DLC
;CAN intiliasation finished
;****************************************************************************
;
   ;enable Parallel Slave Port
      movlw 0xFF
      movwf INTCON2
      movlw 0xC0
      movwf INTCON3
      bsf        INTCON,PEIE    ; Enable peripheral interrupts
      bsf        INTCON,GIE     ; Enable global interrupts
      bcf INTCON,RBIE
      bcf PIR1,PSPIF
      movlw               0x80
      movwf               IPR1
      bsf        PIE1,PSPIE
      movlw      0x0C
      movwf      var1
loop nop
      CLRF              WDTCON
      goto loop
int__ser
           btfss PIR1,PSPIF                                    ;test if
interrupt due to PSP
          bra nothing
         bcf PIR1,PSPIF
          btfss TRISE,IBF
          bra finish
              call delay
              MOVFF PORTD,LATC
          NOP
          NOP
             MOVF LATC,w
         MOVWF POSTINC0 ; store data into memory block
         DCFSNZ var1
         call CAN ; Call CAN subroutine to send data
             goto finish
nothing
              goto        $
finish
        bcf PIR1,PSPIF ;clear PSP flag
            bcf TRISE,IBF
        retfie
CAN                     ;store data for CAN communication
                        LFSR 0, 100h
                        movlw 0x0C
                        movwf var1
                        MOVF POSTINC0,w
                        movwf TXB0D0
                        MOVF POSTINC0,w
                        movwf TXB0D1
                        MOVF POSTINC0,w
                        movwf TXB0D2
                        MOVF POSTINC0,w
                        movwf TXB0D3
;;Adding 8 bytes in one messege.*****************
                                    MOVF POSTINC0,w
                        movwf       TXB0D4
                        MOVF POSTINC0,w
                        movwf       TXB0D5
                        MOVF POSTINC0,w
```

-continued

```
                              movwf        TXB0D6
                              MOVF POSTINC0,w
                              movwf        TXB0D7
;************************************************
                              MOVF POSTINC0,w
                              movwf        TXB1D0
                              MOVF POSTINC0,w
                              movwf        TXB1D1
                              MOVF POSTINC0,w
                              movwf        TXB1D2
                              MOVF POSTINC0,w
                              movwf        TXB1D3
; *************CAN MESSAGE SENDING****************************
                        movlw           TXB0CON_SEND
                        movff               WREG, TXB0CON         ; request sending TXB0 on
bus
                        BTFSC TXB0CON, TXREQ ; Is it transmitted?
                        BRA $-2 ; No. Continue to wait...
                        movff               WREG, TXB1CON         ; request sending TXB0 on
bus
                        BTFSC TXB1CON, TXREQ ; Is it transmitted?
                        BRA $-2 ; No. Continue to wait...
                        LFSR 0, 100h
                        RETURN
delay
             movlw        0x04
                movwf               var3
loop1 decfsz        var3,1            ;Subtract 1
                goto          loop1
                return
                        end
```

Appendix E to Part 2: Code for UIC Functions

[0628] The UIC functions were programmed for simulation using Matlab. Those functions have been rewritten in VC++. The new version also generalized some cases and provided many default parameters.

[0629] The header file is as follows:

```
#ifndef __UIC__
#define __UIC__
class UIC {
public:
        UTC(
                double BaseVoltage = 120.0,
                double DGSize = 85.0,
                double AdjUFTripFrequency = 59.8,
                double AdjUFTripClearingTime = 300.0,
                double AggregateRatingofDGUnits = 100.0,
                double ReversePowerThreshold = 10.0,
                double OverCurrentRelayPickUpCurrent = 5.0,
                double OverCurrentRelaySettingA = 28.0,
                double OverCurrentRelaySettingB = 0.13
        ); // Constructor
        ~UIC( ); // Destructor
        bool CBTrip(
                const bool CBStatus,
                const bool NewStart,
                const double VCurrentAreaEPS[3][2],
                const double ICurrentAreaEPS[3][2],
                const double VCurrentLocal[3][2],
           const double VOldLocal[3][2],
                const double VOldAreaEPS[3][2],
                const double ICurrentLocal[3][2]
        ); // Circuit Breaker Trip Function
        double Difference;
        double fAreaEPS; // frequency of Area EPS
        double fLocal; // frequency of DG
```

-continued

```
    double rPower; // Active power exchange with the utility,
                          // negative values mean output power to utility,
                          // and positive values mean input power from utility
      double xPower;
    //Amplitude Filter Coeff
    double Y1oldGen[3]; double Y2oldGen[3];
    double Y3oldGen[3]; double Y4oldGen[3];
    double Y1newGen[3]; double Y2newGen[3];
    double Y3newGen[3]; double Y4newGen[3];
    double YdataGen[3]; double Y4PrimeGen[3];
    double Y1oldGrid[3]; double Y2oldGrid[3];
    double Y3oldGrid[3]; double Y4oldGrid[3];
    double Y1newGrid[3]; double Y2newGrid[3];
    double Y3newGrid[3]; double Y4newGrid[3];
    double YdataGrid[3]; double Y4PrimeGrid[3];
// Angle Filter Coeff.
    double aY1oldGen[3]; double aY2oldGen[3];
    double aY3oldGen[3]; double aY4oldGen[3];
    double aY1newGen[3]; double aY2newGen[3];
    double aY3newGen[3]; double aY4newGen[3];
    double aYdataGen[3]; double aY4PrimeGen[3];
    double aY1oldGrid[3]; double aY2oldGrid[3];
    double aY3oldGrid[3]; double aY4oldGrid[3];
    double aY1newGrid[3]; double aY2newGrid[3];
    double aY3newGrid[3]; double aY4newGrid[3];
    double aYdataGrid[3]; double aY4PrimeGrid[3];
    double CoeffAmpGenX[3][2];
    double CoeffAmpGenY[3][2];
    double CoeffAmpGenZ[3][2];
    double CoeffAmpGridX[3][2];
    double CoeffAmpGridY[3][2];
    double CoeffAmpGridZ[3][2];
    double CoeffAmpCurrX[3][2];
    double CoeffAmpCurrY[3][2];
    double CoeffAmpCurrZ[3][2];
    bool GridFlag,GenFlag,first;
    // Frequency Filter Coeff.
    double CoeffGridA[3][11];
    double CoeffGridB[3][11];
    double CoeffGridC[3][11];
      double CoeffGenA [3][11];
      double CoeffGenB [3][11];
      double CoeffGenC [3][11];
      int WindNumberGen[3];
    int WindNumberGrid,v;
      // Filter Coeff.
    double IAreaEPS[3][2];
    double ILocal[3][2];
      double VLocal[3][2];
    double VAreaEPS[3][2];// 3 phase voltages/current of Area EPS/Local,
                                  // where, column 0 -> Amplitudes, and column 1
-> Angles;
                                  // row 0 -> phase A, row 1 -> phase B, and row
2 -> phase C;
private:
      // Variables
      double BaseVoltage; // Base voltage in Volts
      double DGSize; // DG size in kilo Watts
      double AdjUFTripFrequency; // Adjustable under-frequency trip settings
      double AdjUFTripClearingTime;
      double AggregateRatingofDGUnits; // Aggregate rating of DG units in kVA
      double ReversePowerThreshold; // Threshold of Reverse Power Relay in kW
      //double VAreaEPS[3][2];
      //double IAreaEPS[3][2]; // 3 phase currents of Area EPS
      //double VLocal[3][2]; // 3 phase voltages of DG
    double VLocalOld[3][2];
      //double ILocal[3][2]; // 3 phase currents of DG
      double Power;
      double ReactivePower;
      int CaseVoltageDisturbance[3][4]; // counters for 3 phase voltages under
4 circumstances
      int CaseFrequencyDisturbance[3];     // counters for frequency under 5
circumstances
                                        // 2 circumstances are for DG size <=
30 kW
```

-continued

```
                                        // 3 circumstances are for DG size > 30
kW
                                                    // First 2 case
counters are shared.
        int CaseSynchronizationCheck; // counters for sychronization check
        int CaseInadvertentEnergization; // counters for inadvertent
energization
        double AreaPreviousAngles[3][2]; // 3 phase area angle history of 100 ms
or 6 cycles
        double LocalPreviousAngles[3][72]; // 3 phase local angle history of 100
ms or 6 cycles
        int IndexOfPreviousAngles; // index showing which previous angle is
currently used in calculation
        double OverCurrentRelayPickUpCurrent; // Overcurrent relay pick up
current
        double OverCurrentRelaySettingA; // Overcurrent relay setting A
        double OverCurrentRelaySettingB; // Overcurrent relay setting B
        double CaseOverCurrentRelay[3]; // 3 phase overcurrent relay's induction
disk positions
        // Functions
        void ResetCounters( ); // function to reset all counters
        void InitializePreviousAngles( ); // function to initialize previous
angle array
        double FrequencyCalculation1(double V[3][2]);
        double FrequencyCalculation2(double V[3][2]);// Calculate frequency from
3 phase angle history
        double GetThreePhaseReActivePower( );
        double GetThreePhaseActivePower( );
     //double GetThreePhaseActivePower( ); // Retrieve 3 phase active power
     bool RelayofOverUnderVoltage( ); // Over/Under-voltage relay logic
        bool RelayofOverUnderFrequency( ); // Over/Under-frequency relay logic
        bool RelayofSynchronizationCheckandInadvertentEnergization( ); //
Sycnroniztion check and inadvertent energization relay logic
        bool RelayofReversePower( ); // Reverse power relay logic
        bool RelayofOverCurrent( ); // Overcurrent relay logic
        bool RelayofGroundFaultDetection( ); // Ground fault detection relay
logic
        void InitializeCaseOverCurrentRelay( ); // Initialize induction disk
positions
};
#endif
```

[0630] The implementation file is as follows:

```
#include "stdafx.h"
#include "UIC.h"
#include "globals.h"
#include <cmath>
#define PI 3.14159265359
#define DELTA_T 1.0 / 720
#define CONSTANT 5*2*3.14159265
#define SAMPLE_CONSTANT 0.0013888 //(1/720)
extern Variables allStatus;
UIC::UIC(double BV, double DGS, double AUFTF,
                double AUFTCT, double ARoDGU, double RPT,
                double OCRPUC, double OCRSA, double OCRSB)
                : BaseVoltage(BV), DGSize(DGS), AdjUFTripFrequency(AUFTF),
            AdjUFTripClearingTime(AUFTCT), AggregateRatingofDGUnits(ARoDGU),
                ReversePowerThreshold(RPT),
OverCurrentRelayPickUpCurrent(OCRPUC),
                OverCurrentRelaySettingA(OCRSA), OverCurrentRelaySettingB(OCRSB)
{
        ResetCounters( );
        InitializePreviousAngles( );
        IndexOfPreviousAngles = 0;
        InitializeCaseOverCurrentRelay( );
}
UIC::~UIC( ) {
}
double UIC::GetThreePhaseReActivePower( ) {
```

-continued

```
double ReactivePower3 = 0.0;
      // Calculate 3 phase active power
      for(int i=0;i<3;i++)
           ReactivePower3 += VLocal[i][0] * ILocal[i][0] * sin((VLocal[i][1]
– ILocal[i][1]) * PI / 180);
      ReactivePower = ReactivePower3;
      return ReactivePower;
}
double UIC::GetThreePhaseActivePower( ) {
double RealPower3 = 0.0;
      // Calculate 3 phase active power
      for(int i=0;i<3;i++)
           RealPower3 += VLocal[i][0] * ILocal[i][0] * cos((VLocal[i][1] –
ILocal[i][1]) * PI / 180);
      Power = RealPower3;
      return Power;
}
void UIC::ResetCounters( ) {
      for(int i=0;i<4;i++)
           for(int j=0;j<3;j++)
                CaseVoltageDisturbance[i][j] = 0;
      for(int i=0;i<5;i++)
           CaseFrequencyDisturbance[i] = 0;
      CaseSynchronizationCheck = 0;
      CaseInadvertentEnergization = 0;
      return;
}
void UIC::InitializePreviousAngles( ) {
        first = 0;
        v =0;
            GridFlag=0;
         GenFlag =0;
for(int i=0;i<3;i++)
{
    Y1oldGen[i]= 0;     Y2oldGen[i]= 0;
    Y3oldGen[i]= 0;     Y4oldGen[i]= 0;
    Y1newGen[i]= 0;     Y2newGen[i]= 0;
    Y3newGen[i]= 0;     Y4newGen[i]= 0;
    YdataGen[i]= 0;     Y4PrimeGen[i]= 0;
    Y1oldGrid[i]= 0; Y2oldGrid[i]= 0;
    Y3oldGrid[i]= 0; Y4oldGrid[i]= 0;
    Y1newGrid[i]= 0; Y2newGrid[i]= 0;
    Y3newGrid[i]= 0; Y4newGrid[i]= 0;
    YdataGrid[i]= 0; Y4PrimeGrid[i]= 0;
    aY1oldGen[i]= 0;     aY2oldGen[i]= 0;
    aY3oldGen[i]= 0;     aY4oldGen[i]= 0;
    aY1newGen[i]= 0;     aY2newGen[i]= 0;
    aY3newGen[i]= 0;     aY4newGen[i]= 0;
    aYdataGen[i]= 0;     aY4PrimeGen[i]= 0;
    aY1oldGrid[i]= 0; aY2oldGrid[i]= 0;
    aY3oldGrid[i]= 0; aY4oldGrid[i]= 0;
    aY1newGrid[i]= 0; aY2newGrid[i]= 0;
    aY3newGrid[i]= 0; aY4newGrid[i]= 0;
    aYdataGrid[i]= 0; aY4PrimeGrid[i]= 0;
   for(int j=0;j<2;j++)
{//Initialize Generator voltage Amp Filter coeff.
    CoeffAmpGenX[i][j] = 0;
    CoeffAmpGenY[i][j] = 0;
    CoeffAmpGenZ[i][j] = 0;
//Initialize Grid Voltage Amp Filter coeff.
    CoeffAmpGridX[i][j] =0;
    CoeffAmpGridY[i][j] =0;
    CoeffAmpGridZ[i][j] =0;
//Initialize Current Amp Filter coeff.
    CoeffAmpCurrX[i][j]= 0;
    CoeffAmpCurrY[i][j]= 0;
    CoeffAmpCurrZ[i][j]= 0;
}
}
   for(int i=0;i<3;i++)
           for(int j=0;j<11;j++) {
                CoeffGenA[i][j] = 0;
                CoeffGenB[i][j] = 0;
           CoeffGenC[i][j] = 0;
                CoeffGridA[i][j] = 0;
```

-continued

```
            CoeffGridB[i][j] = 0;
            CoeffGridC[i][j] = 0;
            }
        return;
}
void UIC::InitializeCaseOverCurrentRelay( ){
        for(int i=0;i<3;i++)
            CaseOverCurrentRelay[i] = 0.0;
        return;
}
bool UIC::RelayofOverUnderVoltage( ) {
        // Over/Under-voltage relay logic
        // Sampling rate is 720 samples per second for a 60 Hz system.
        // Based on Table 1 of IEEE 1547 (p8).
        // Voltage Range (% of base voltage - rms)                Clearing Time
        //          V < 50                      0.16 s /   10 cycles /   120
samples
        //      50 <= V < 88                    2.00 s / 120 cycles / 1440
samples
        //      110 < V < 120                   1.00 s /   60 cycles /   720
samples
        //          V >= 120                    0.16 s /   10 cycles /   120
samples
        for(int i=0;i<3;i++) { // 3 phase
            if(VAreaEPS[i][0] < 0.5*BaseVoltage) {
                if(++CaseVoltageDisturbance[i][0] >= 120)
                    return 1;
            }
            else
                CaseVoltageDisturbance[i][0] = 0;
            if(VAreaEPS[i][0] < 0.88*BaseVoltage) {
                if(++CaseVoltageDisturbance[i][1] >= 1440)
                    return 1;
            }
            else
                CaseVoltageDisturbance[i][1] = 0;
            if(VAreaEPS[i][0] > 1.1*BaseVoltage) {
                if(++CaseVoltageDisturbance[i][2] >= 720)
                    return 1;
            }
            else
                CaseVoltageDisturbance[i][2] = 0;
            if(VAreaEPS[i][0] >= 1.2*BaseVoltage) {
                if(++CaseVoltageDisturbance[i][3] >= 120)
                    return 1;
            }
            else
                CaseVoltageDisturbance[i][3] = 0;
        }
        return 0;
}
bool UIC::RelayofOverUnderFrequency( ) {
        // Over/Under-frequecny relay logic
        // Sampling rate is 720 samples per second for a 60 Hz system.
        // Based on Table 2 of IEEE 1547 (p9).
        if(DGSize <= 30.0) {
            // For DGs of size <= 30 kW
            // Frequency Range (Hz)                   Clearing Time
            //           f < 59.3               0.16 s / 10 cycles / 120 samples
            //           f > 60.5               0.16 s / 10 cycles / 120 samples
            if(fLocal < 59.3) {
                if(++CaseFrequencyDisturbance[0] >= 120)
                    return 1;
            }
            else
                CaseFrequencyDisturbance[0] = 0;
            if(fLocal > 60.5) {
                if(++CaseFrequencyDisturbance[1] >= 120)
                    return 1;
            }
            else
                CaseFrequencyDisturbance[1] = 0;
        }
        else {
            // For DGs of size > 30 kW
            // Frequency Range (Hz)                 Clearing Time
```

-continued

```
        //          f < 57.0                0.16 s / 10 cycles / 120 samples
        //  57.0 < f <= 59.8                Adjustable 0.16 s / 10 cycles / 120
samples
        //                                     to 300 s / 18000 cycles / 216000
samples
        //          f > 60.5                0.16 s / 10 cycles / 120 samples
        if(fLocal < 57.0) {
            if(++CaseFrequencyDisturbance[0] >= 120)
                return 1;
        }
        else
            CaseFrequencyDisturbance[0] = 0;
        if(fLocal < AdjUFTripFrequency) {
            if(++CaseFrequencyDisturbance[1] >=
720*AdjUFTripClearingTime)
                return 1;
        }
        else
            CaseFrequencyDisturbance[1] = 0;
        if(fLocal > 60.5) {
            if(++CaseFrequencyDisturbance[2] >= 120)
                return 1;
        }
        else
            CaseFrequencyDisturbance[2] = 0;
    }
    return 0;
}
bool UIC::RelayofSynchronizationCheckandInadvertentEnergization( ) {
    //
    double MaxVAmplitudeDifference = 0.0;
    double MaxVAngleDifference = 0.0;
    double MaxVAmplitudeAreaEPS = 0.0;
    for(int i=0;i<3;i++) {
        // Get the maximum difference of voltage amplitudes in 3 phases
        double temp = abs(VAreaEPS[i][0] − VLocal[i][0])/BaseVoltage*100;
        if(MaxVAmplitudeDifference < temp)
            MaxVAmplitudeDifference= temp;
        // Get the maximum difference of voltage angles in 3 phases
        temp = abs(VAreaEPS[i][1] − VLocal[i][1]);
        if(MaxVAngleDifference < temp)
            MaxVAngleDifference = temp;
        // Get the maximum Area EPS voltage in 3 phases
        temp = VAreaEPS[i][0];
        if(MaxVAmplitudeAreaEPS < temp)
            MaxVAmplitudeAreaEPS = temp;
   }
    //MaxVAngleDifference = abs(VAreaEPS[0][1] − VLocal[0][1]);
    double fDifference = abs(fAreaEPS−fLocal);
    if(MaxVAmplitudeAreaEPS > 0.88 * BaseVoltage
        && MaxVAmplitudeAreaEPS < 1.1 *BaseVoltage
        && fLocal > 59.3
        && fLocal < 60.5) {
        if(++CaseInadvertentEnergization > 300000)
            CaseInadvertentEnergization = 300000;
        }
    else
        CaseInadvertentEnergization = 0;
    double fDifferenceTolerance, VAmplitudeDifferenceTolerance,
VAngleDifferenceTolerance;
    if(AggregateRatingofDGUnits <= 500) {
        fDifferenceTolerance = 0.5;
        VAmplitudeDifferenceTolerance = 10;
        VAngleDifferenceTolerance = 20;
    }
    else if(AggregateRatingofDGUnits <= 1500) {
        fDifferenceTolerance = 0.2;
        VAmplitudeDifferenceTolerance = 5;
        VAngleDifferenceTolerance = 15;
    }
    else {
        fDifferenceTolerance = 0.1;
        VAmplitudeDifferenceTolerance = 3;
        VAngleDifferenceTolerance = 10;
    }
```

-continued

```
    if(fDifference < fDifferenceTolerance
        && MaxVAmplitudeDifference < VAmplitudeDifferenceTolerance
        && MaxVAngleDifference < VAngleDifferenceTolerance) {
        if(++CaseSynchronizationCheck > 1000)
            CaseSynchronizationCheck = 1000;
        }
    else
        CaseSynchronizationCheck = 0;
    if(CaseInadvertentEnergization > 21600 && CaseSynchronizationCheck >
720)
        return 0;
    return 1;
}
bool UIC::RelayofReversePower( ) {
    double RealPower3 = 0.0;
    // Calculate 3 phase active power
    for(int i=0;i<3;i++)
        RealPower3 += VAreaEPS[i][0] * IAreaEPS[i][0] *
cos((VAreaEPS[i][1] − IAreaEPS[i][1]) * PI / 180);
    Power = RealPower3;
    if(RealPower3 < ReversePowerThreshold * 1000)
        return 1;
    else
      return 0;
}
bool UIC::RelayofOverCurrent( ) {
    for(int i=0;i<3;i++) { // Update induction disk positions for each phase
        // Define two temperary variables
        double ApplitudeMultiplier1 =
IAreaEPS[i][0]/OverCurrentRelayPickUpCurrent;
        double ApplitudeMultiplier2 =
ApplitudeMultiplier1*ApplitudeMultiplier1 − 1;
        // If phase current is greater than or equal to pick up current,
position increases
        if(IAreaEPS[i][0]>=OverCurrentRelayPickUpCurrent) {
            CaseOverCurrentRelay[i] +=
1/((OverCurrentRelaySettingA/ApplitudeMultiplier2)+OverCurrentRelay-
SettingB)*DELTA_T;
            // Return trip if the phase's induction disk position is
great than 1
            if(CaseOverCurrentRelay[i]>1.0)
                return 1;
        }
        // If phase current is less than pick up curren, postion decreases
        else {
            CaseOverCurrentRelay[i] +=
1/(OverCurrentRelaySettingA/ApplitudeMultiplier2)*DELTA_T;
            // Stop decreasing if the phase's induction disk position is
less than 0
            if(CaseOverCurrentRelay[i]<0.0)
                CaseOverCurrentRelay[i]=0.0;
        }
    }
    return 0;
}
bool UIC::RelayofGroundFaultDetection( ) {
    return 0;
}
bool UIC::CBTrip(const bool CBStatus,
                 const bool NewStart,
            const double VCurrentAreaEPS[3][2],
                 const double ICurrentAreaEPS[3][2],
                 const double VCurrentLocal[3][2],
            const double VOldLocal[3][2],
                 const double VOldAreaEPS[3][2],
                 const double ICurrentLocal[3][2]
                 )
{
//Filter Coeff
   float a3 = 280.0;
   float a2 = 34400.0;
   float a1 = 2072000.0;
   float a0 = 54760000.0;
    // Localize all measurements.
    for(int i=0;i<3;i++)
        for(int j=0;j<2;j++)
```

-continued

```
          {
               VAreaEPS[i][j] = VCurrentAreaEPS[i][j];
               IAreaEPS[i][j] = ICurrentAreaEPS[i][j];
               VLocal[i][j] = VCurrentLocal[i][j];
          VLocalOld[i][j]= VOldLocal[i][j];
               ILocal[i][j] = ICurrentLocal[i][j];
               AreaPreviousAngles[i][j] = VOldAreaEPS[i][j];
          }
// Calculate area frequency, Local Freq, Real Power and Reactive Power
      fAreaEPS = FrequencyCalculation2(VAreaEPS);
      fLocal = FrequencyCalculation1(VAreaEPS);
//double Const = 5*2*3.14159265;
        if(first){
          first = 1;
    for(int i=0;i<3;i++)
    {
       Y1oldGrid[i]= VAreaEPS[i][0];
       Y1oldGen[i]= VLocal[i][0];
      }
      }
      for(int i=0;i<3;i++)
        {
           YdataGen[i] = VLocal[i][0];
       YdataGrid[i] = VAreaEPS[i][0];
      Y1newGen[i]= Y1oldGen[i]+SAMPLE__CONSTANT*Y2oldGen[i];
      Y2newGen[i]= Y2oldGen[i]+SAMPLE__CONSTANT*Y3oldGen[i];
      Y3newGen[i]= Y3oldGen[i]+SAMPLE__CONSTANT*Y4oldGen[i];
      Y4PrimeGen[i]=a0*(YdataGen[i]−Y1oldGen[i])−a1*Y2oldGen[i]−
a2*Y3oldGen[i]−a3*Y4oldGen[i];
      Y4newGen[i]= Y4oldGen[i]+SAMPLE__CONSTANT*Y4PrimeGen[i];
      Y1newGrid[i]= Y1oldGrid[i]+SAMPLE__CONSTANT*Y2oldGrid[i];
      Y2newGrid[i]= Y2oldGrid[i]+SAMPLE__CONSTANT*Y3oldGrid[i];
      Y3newGrid[i]= Y3oldGrid[i]+SAMPLE__CONSTANT*Y4oldGrid[i];
      Y4PrimeGrid[i]=a0*(YdataGrid[i]−Y1oldGrid[i])−a1*Y2oldGrid[i]−
a2*Y3oldGrid[i]−a3*Y4oldGrid[i];
      Y4newGrid[i]= Y4oldGrid[i]+SAMPLE__CONSTANT*Y4PrimeGrid[i];
          VLocal[i][0] = Y1newGen[i];
          VAreaEPS[i][0]= Y1newGrid[i];
    //    ILocal[i][0] = CoeffAmpCurrZ[i][1];
     //   IAreaEPS[i][0]= CoeffAmpCurrZ[i][1];
          allStatus.FilteredPMU.VLocal[i][0] = Y1newGen[i];
       allStatus.FilteredPMU.VLocal[i][1] = VCurrentLocal[i][1];
       allStatus.FilteredPMU.VAreaEPS[i][0] = Y1newGrid[i];
       allStatus.FilteredPMU.VAreaEPS[i][1] = VCurrentAreaEPS[i][1];
       Y1oldGen[i] = Y1newGen[i];
       Y2oldGen[i] = Y2newGen[i];
       Y3oldGen[i] = Y3newGen[i];
       Y4oldGen[i] = Y4newGen[i];
       Y1oldGrid[i] = Y1newGrid[i];
       Y2oldGrid[i] = Y2newGrid[i];
       Y3oldGrid[i] = Y3newGrid[i];
       Y4oldGrid[i] = Y4newGrid[i];
    }
     allStatus.FilteredPMU.FLocal = fLocal;
   rPower = GetThreePhaseActivePower( );
     xPower = GetThreePhaseReActivePower( );
     allStatus.UIC.rPower = rPower;
     allStatus.UIC.xPower = xPower;
     // Initialize counters if necessary
     if(NewStart == 1)
          ResetCounters( );
     if(CBStatus ==0)
          // Circuit breaker is closed.
          return RelayofOverUnderVoltage( );/* ||
RelayofOverUnderFrequency( )|| RelayofReversePower( )
                || RelayofReversePower( ) || RelayofOverCurrent( ) ||
RelayofGroundFaultDetection( );*/
     else
          // Circuit breaker is open.
          return RelayofSynchronizationCheckandInadvertentEnergization( );
}
double UIC::FrequencyCalculation2( double V[3][2]) {
     double freq[3] = {0.0};
     double Freq,diff1,diff2;
    //Filter Coeff
    float a3 = 120.0;
```

-continued

```
    float a2 = 5600.0;
    float a1 = 120000.0;
    float a0 = 1000000.0;
// Start Looping here//
      if(GridFlag == 0){
       GridFlag =1;
         for(int i=0;i<3;i++)
         {
         aY1oldGrid[i]= V[i][1];
       // aY1oldGen[i]= VLocal[i][1];
           }
    }
         for(int i=0;i<3;i++)
       {
            //aYdataGen[i] = VLocal[i][1];
         aYdataGrid[i] = V[i][1];
         /*     diff1 = abs(aYdataGen[i] − aY1oldGen[i] − 360);
                diff2 = abs(aYdataGen[i] − aY1oldGen[i] + 360);*/
          diff1 = abs(aYdataGrid[i] − aY1oldGrid[i] − 360);
                diff2 = abs(aYdataGrid[i] − aY1oldGrid[i] + 360);
                double PhaseShift = 0.0;
          if ( diff1 < 180.0 )
              PhaseShift = 360.0;
          else if ( diff2 < 180.0 )
              PhaseShift = −360.0;
       aY1oldGrid[i] = aY1oldGrid[i] + PhaseShift;
       aY1newGrid[i]= aY1oldGrid[i]+SAMPLE__CONSTANT*aY2oldGrid[i];
       aY2newGrid[i]= aY2oldGrid[i]+SAMPLE__CONSTANT*aY3oldGrid[i];
       aY3newGrid[i]= aY3oldGrid[i]+SAMPLE__CONSTANT*aY4oldGrid[i];
       aY4PrimeGrid[i]=a0*(aYdataGrid[i] − aY1oldGrid[i]) − a1*aY2oldGrid[i] −
a2*aY3oldGrid[i] − a3*aY4oldGrid[i];
       aY4newGrid[i] = aY4oldGrid[i]+SAMPLE__CONSTANT*aY4PrimeGrid[i];
       freq[i] = 60.0 + (aY2newGrid[i]/360.00);
        aY1oldGrid[i] = aY1newGrid[i];
        aY2oldGrid[i] = aY2newGrid[i];
        aY3oldGrid[i] = aY3newGrid[i];
        aY4oldGrid[i] = aY4newGrid[i];
    }
             Freq = (freq[0]);//+freq[1]+freq[2])/3;
      return Freq;
}
double UIC::FrequencyCalculation1(double V[3][2]) {
      double freq[3] = {0.0};
      double New,Freq,Diff1,Diff2, PhaseShift;
// Start Looping here//
    for(int i=0;i<1;i++){
    New = V[i][1];
      if(GenFlag == 0){
       GenFlag =1;
     for(int j=0;j<9;j++){
          CoeffGenB[i][j]= New;}
    }
    Diff1 = abs(New − CoeffGenB[i][0] − 360.0);
    Diff2 = abs(New − CoeffGenB[i][0] + 360.0);
    PhaseShift = 0.0;
    if(Diff1 < 180.0)
        PhaseShift = 360.0;
    else if (Diff2 < 180.0)
        PhaseShift = −360.0;
    for(int j=0;j<9;j++){
        CoeffGenB[i][j]= CoeffGenB[i][j]+ PhaseShift;}
    CoeffGenA[i][0]= CONSTANT*(New − CoeffGenB[i][0]);
    CoeffGenC[i][0] = CoeffGenB[i][0] + SAMPLE__CONSTANT* CoeffGenA[i][0];
    for(int j =1 ; j <9; j++){
        CoeffGenA[i][j] = CONSTANT*(CoeffGenB[i][j−1]−CoeffGenB[i][j]);
        CoeffGenC[i][j] = CoeffGenB[i][j]+ SAMPLE__CONSTANT*CoeffGenA[i][j];
    }
        freq[i] = 60 + (CoeffGenA[i][8]/360.0);
          for(int j=0;j< 9;j++){
          CoeffGenB[i][j]=CoeffGenC[i][j];}
    }//Finish Looping here//
             Freq = (freq[0]);
      return Freq;
}
```

[0631] One UIC implementation software file in the illustrated embodiment is:

```
// DG.cpp : Defines the entry point for the console application.
//
#include <iostream>
#include <fstream>
#include <cmath>
#include <time.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <sys/timeb.h>
#include “UIC.h”
#include “Protection.h”
#include “DGIO.h”
#include “stdafx.h”
#include “Globals.h”
#include “Control.h”
#include “debugging.h”
#define PI 3.14159265359
PCAN_MSG_T z;
PMUDatafromCAN *PMU1;
UIC *uic;
static DGIO *myio;
Protection *protect;
Control *dgControl;
Debugging *debug;
Variables allStatus;
Settings setting;
bool SyncCheckLoop;
bool GPMControlLoop;
bool perturbationLoop;
using namespace std;
int initPMU ( );
void UICCheck ( );
void menu ( );
int interactive ( );
void SyncCheckPIDControl ( );
void GPMPIDControl ( );
double PIDOutput (double error, PIDData& pid);
void allStatusReset ( );
double perturbation (double Amplitude);
void PerturbationLoop ( );
int _tmain(int argc, _TCHAR* argv[ ])
{
   printf (“Start initialization ...\n”);
      debug = new Debugging (57600, true);
      allStatusReset ( );
   initPMU ( );
   myio = new DGIO(2, 4);
   if (myio->errorCode != 1000)
   {
      printf (“IO cards initialization error!!!\n”);
      getch ( );
      exit (0);
   }
   protect = new Protection ( );
   dgControl = new Control ( );
   uic = new UIC(277.,85.,59.8,300.,100.,0.1,5.,28.,0.13); // Define an
instance of UIC
   printf (“Initialization completed...\n”);
   menu ( );
   int loop = 1;
   int counter1 = 0;
   int counter2 = 0;
   SyncCheckLoop = false;
   GPMControlLoop = false;
   perturbationLoop = false;
   while (loop)
   {
      loop = interactive ( );
      counter1 ++;
      counter2 ++;
      if (counter1 >= 100)
      {
```

-continued

```
            myio->readInputs ( );
            counter1 = 0;
        }
        PMU1->CANUpdateStatus( ); // Check for PMU status
        // If data is there get values, otherwise skip.
        if (PMU1->bReady)
            PMU1->getData(allStatus.PMU.Data); // Read all data in buffer*/
        UICCheck ( );
////        protect->doEvents ( );
        dgControl->doEvents ( );
        if (counter2 >= 20)
        {
            counter2 = 0;
            if (perturbationLoop == true)
                PerturbationLoop ( );
            if (SyncCheckLoop == true)
                SyncCheckPIDControl ( );
            if (GPMControlLoop == true)
                GPMPIDControl ( );
            debug->recording (allStatus);
            myio->writeOutputs ( );
        }
        if (debug->reachLimit ( ))
            loop = 0;
    }
    struct tm *newtime;
    struct __timeb64 timebuffer;
    __ftime64(&timebuffer);
    newtime = __gmtime64(&timebuffer.time);
    char fileName[30];
    sprintf (fileName, “%02d%02d%02d%02d”, newtime->tm__mon+1,
newtime->tm__mday,newtime->tm__hour-5, newtime->tm__min);
    debug->writeFile (fileName);
}
void menu ( )
{
    printf (“\n---------------- Menu --------------------\n”);
    printf (“e: exit\n”);
    printf (“z: syncControl\n”);
    printf (“x: GPM Power Control\n”);
    printf (“t: Test Mode\n”);
    printf (“b: perturbation GIM mode\n”);
    printf (“u: CB Status, power, frequencies\n”);
    printf (“i: IO values\n”);
    printf (“y: Sync Check\n”);
    printf (“p: PMU values\n”);
    printf (“s: start or stop recording\n”);
    printf (“enter a command: ”);
}
int interactive ( )
{
    char c;
    if (__kbhit ( ))
    {
        c = __getch ( );
        switch (c)
        {
          case ‘z’:
              printf (“\nSync Control Mode:\n”);
              debug->recording ( );
              SyncCheckLoop = true;
              GPMControlLoop = false;
              break;
          case ‘x’:
              printf (“GPM Power Control Model: \n”);
              debug->recording ( );
              SyncCheckLoop = false;
              GPMControlLoop = true;
              break;
          case ‘t’:
              printf (“\nTest Mode:\n”);
              allStatusReset ( );
              SyncCheckLoop = false;
              GPMControlLoop = false;
              break;
```

-continued

```
    case ‘b’:
      perturbationLoop = true;
      break;
    case ‘d’:
      setting.RealLoadReference −= 2000;
      if (setting.RealLoadReference < 0)
        setting.RealLoadReference = 0;
      printf (“RealLoadReference: %f\n”, setting.RealLoadReference);
      break;
    case ‘e’:
      return 0;
      break;
    case ‘s’:
      debug->startRecording = ! debug->startRecording;
      if (debug->startRecording == true)
        printf (“start recording data...”);
      else
        printf (“Stop recording data!”);
      break;
      case ‘u’:
      printf (“\nPower: CB status %d\n”,
allStatus.IO.digitalOut.genBreaker);
      printf (“Real Power: %f\n”, allStatus.UIC.rPower);
      printf (“Reactive Power: %f\n”, allStatus.UIC.xPower);
      printf (“Frequencies: Grid: %f, Local: %f\n”, uic->fAreaEPS,
uic->fLocal);
      break;
    case ‘y’:
      printf (“\nSync Check: \n”);
      printf (“SpeedBias:%4.2f, %4.2f\n”,
allStatus.IO.analogMonitoring.speedBias, allStatus.IO.analogOut.speedBias);
      printf (“VoltageBias:%4.2f, %4.2f\n”,
allStatus.IO.analogMonitoring.voltageBias,
allStatus.IO.analogOut.voltageBias);
      printf (“%4.2f,”, allStatus.PMU.Data[3].Phase-
allStatus.PMU.Data[6].Phase);
      printf (“%4.2f,”, allStatus.PMU.Data[4].Phase-
allStatus.PMU.Data[7].Phase);
      printf (“%4.2f,”, allStatus.PMU.Data[5].Phase-
allStatus.PMU.Data[8].Phase);
      printf (“\n”);
      break;
    case ‘p’:
      printf (“\nPMU reading:\n”);
      printf (“Current All (A, B, C): ”);
      for (int i =0; i < 3; i ++)
        printf (“%6.4f(%4.2f),”, 0.0335*4*allStatus.PMU.Data[i].Amp,
allStatus.PMU.Data[i].Phase);
      printf (“\nVoltage Grid (A, B, C): ”);
      for (int i =0; i < 3; i ++)
        printf (“%6.4f(%4.2f),”, uic->VAreaEPS[i][0],
allStatus.PMU.Data[3+i].Phase);
      printf (“\nVoltage Gen (A, B, C): ”);
      for (int i =0; i < 3; i ++)
        printf (“%6.4f(%4.2f),”,uic->VLocal[i][0],
allStatus.PMU.Data[6+i].Phase);
      printf (“\n”);
      break;
    case ‘i’:
      printf (“\nDigital Inputs: ”);
      for (int i = 0; i < DIGITAL__IN; i ++)
      {
        printf (“%1d”, allStatus.IO.digitalIn.value[i]);
      }
      printf (“\n”);
      printf (“Digital Outputs: ”);
      for (int i = 0; i < DIGITAL__OUT; i ++)
      {
        printf (“%1d”, allStatus.IO.digitalOut.value[i]);
      }
      printf (“\n”);
      printf (“Analog Inputs: Coolant %7.2f C, Oil %7.2f PSI, MPU %7.2f
rpm\n”,
        allStatus.IO.analogIn.CoolantTemp,
        allStatus.IO.analogIn.OilPressure,
        allStatus.IO.analogIn.MPUSpeed);
```

-continued

```
       printf ("Analog Output: SPD__BIAS %7.2f, VOL__BIAS %7.2f",
allStatus.IO.analogOut.speedBias, allStatus.IO.analogOut.voltageBias);
       break;
       default:
       printf ("wrong input\n");
       break;
    }
    menu ( );
  }
  return 1;
}
int initPMU ( )
{
  UINT x =0 ;
  bool y = false;
   z = ( PCAN__MSG__T )GlobalAlloc( GPTR, BUFFER*sizeof(CAN__MSG__T) );
   if( z == NULL )
  {
    cout<<"Out of memory!"<<endl;
     CANPortClose(x);
     return −1;
   }
  PMU1 = new PMUDatafromCAN(x,y,z);
  PMU1->CANOpen(1);
  return 0;
}
void UICCheck ( )
{
  double VCurrentAreaEPS[3][2], ICurrentAreaEPS[3][2]; // Area EPS or grid
voltage and current
  double VCurrentLocal[3][2],ICurrentLocal[3][2]; // Local or gen voltage and
current
   double VOldLocal[3][2],VOldAreaEPS[3][2];
   double GridFactor[3];
   double GenFactor[3];
   GenFactor[0]= 0.2336;GenFactor[1]= 0.2331;GenFactor[2]= 0.2323;
   GridFactor[0]=0.2391;;GridFactor[1]=0.2379;;GridFactor[2]=0.2382;
  bool NewStart = 0;
  // The status of the circuit breaker should be a digital input.
  // 1 means CB is open, and 0 indicates CB is closed.
  bool CBStatus;
  //check if the logic is consistant with egcp2 board
  if (allStatus.IO.digitalIn.genCBAuxSwitch==0)
    CBStatus = 0;
  else
    CBStatus = 0;
  for(int j=0;j<3;j++)
  {
    // 3 phase current either Area EPS (grid) or Local (gen)
    ICurrentAreaEPS[j][0]= 0.0335*4*allStatus.PMU.Data[j].Amp;
    ICurrentAreaEPS[j][1]= allStatus.PMU.Data[j].Phase;
    ICurrentLocal[j][0]= 0.0335*4*allStatus.PMU.Data[j].Amp;
    ICurrentLocal[j][1]= allStatus.PMU.Data[j].Phase;
    // 3 phase voltage for Area EPS (grid)
    VCurrentAreaEPS[j][0]= GridFactor[j]*allStatus.PMU.Data[3+j].Amp;
    VCurrentAreaEPS[j][1]=allStatus.PMU.Data[3+j].Phase;
    // 3 phase voltage for Local (gen)
    VCurrentLocal[j][0]= GenFactor[j]*allStatus.PMU.Data[6+j].Amp;
    VCurrentLocal[j][1]=allStatus.PMU.Data[6+j].Phase;
  }
    // Call UIC functions by Yong Sheng
  bool trip=uic->CBTrip(CBStatus, NewStart, VCurrentAreaEPS, ICurrentAreaEPS,
VCurrentLocal,VOldLocal,VOldAreaEPS, ICurrentLocal);
  //check the following logic !!!!
  allStatus.UIC.freqGrid = uic->fAreaEPS;
  allStatus.UIC.freqLocal = uic->fLocal;
  //allStatus.IO.digitalOut.genBreakerTrip = trip;
  if(trip==1 && CBStatus==0) // Trip CB if it is closed initially.
  {
    allStatus.IO.digitalOut.genBreakerTrip = 1; //                    cout<<" Trip
the circuit breaker!"<<endl;
    allStatus.IO.digitalOut.genBreaker = 1;
     allStatus.IO.digitalOutChanged = 1;
    //   CBStatus = 1;
  }
  else if(trip==0 && CBStatus==1) // Close CB if it is open initially.
```

-continued

```
// else if(trip==0 && CBStatus==0) original code from Sheng, changed CBStatus
from 0 to 1 in the code.
  {
     CBStatus = 0;
    allStatus.IO.digitalOut.genBreaker = 0; //        cout<<“ Close the
circuit breaker!”<<endl;
    allStatus.IO.digitalOut.genBreakerTrip = 0; //          cout<<“ Close the
circuit breaker!”<<endl;
  }
}
void SyncCheckPIDControl ( )
{
  //Phase sync
  double error = allStatus.FilteredPMU.VAreaEPS[0][1] –
allStatus.FilteredPMU.VLocal[0][1];
// double error = allStatus.PMU.voltageLocalA.Phase –
allStatus.PMU.voltageGridA.Phase;
  if (error > 180)
    error −= 360.0;
  if (error < −180.0)
    error += 360.0;
  double speedBias = PIDOutput (error, setting.PID.syncCheckSpeedBiasPID);
  speedBias = speedBias *4/10.0+2.50;
  if (speedBias > 3.5)
    speedBias = 3.5;
  if (speedBias < 1.5)
    speedBias = 1.5;
  if (speedBias != allStatus.IO.analogOut.speedBias)
  {
    allStatus.IO.analogOut.speedBias = speedBias;
    allStatus.IO.analogOutChanged = true;
  }
  //voltage Bias
  error = allStatus.FilteredPMU.VAreaEPS[0][0]–
allStatus.FilteredPMU.VLocal[0][0];
  double voltageBias = PIDOutput (error,
setting.PID.syncCheckVoltageBiasPID);
  voltageBias = voltageBias *4.5/20.0+4.50;
  if (voltageBias > 9.0)
    voltageBias = 9.0;
  if (voltageBias < 0.0)
    voltageBias = 0.0;
  if (voltageBias != allStatus.IO.analogOut.voltageBias)
  {
    allStatus.IO.analogOut.voltageBias = voltageBias;
    allStatus.IO.analogOutChanged = true;
  }
}
void GPMPIDControl ( )
{
  //realPower Control
  double error = setting.RealLoadReference-allStatus.UIC.rPower;
  if (error > 20000.0)
    error = 20000.0;
  if (error < −20000.0)
    error = −20000.0;
  error /= 1000;
  double speedBias = PIDOutput (error, setting.PID.realPowerPID);
  speedBias = speedBias+2.50;
  if (speedBias > 4.5)
    speedBias = 4.5;
  if (speedBias < 0.5)
    speedBias = 0.5;
// speedBias += perturbation(0.1);
  if (speedBias != allStatus.IO.analogOut.speedBias)
  {
    allStatus.IO.analogOut.speedBias = speedBias;
    allStatus.IO.analogOutChanged = true;
  }
  double voltageBias;
  //voltage Bias
  error = setting.ReactiveLoadReference-allStatus.UIC.xPower;
  error /= 1000;
  if (error > 20.0)
    error = 20.0;
```

-continued

```
  if (error < −20.0)
    error = −20.0;
  voltageBias = PIDOutput (error, setting.PID.reactivePowerPID);
  voltageBias = voltageBias+4.50;
  if (voltageBias > 7.5)
    voltageBias = 7.5;
  if (voltageBias < 1.5)
    voltageBias = 1.5;
  if (voltageBias != allStatus.IO.analogOut.voltageBias)
  {
    allStatus.IO.analogOut.voltageBias = voltageBias;
    allStatus.IO.analogOutChanged = true;
  }
}
double PIDOutput (double error, PIDData& pid)
{
  double proportional = error*pid.Gains.p;
  double derivative = (error − pid.lastError) * pid.Gains.d;
  pid.lastError = error;
  pid.accumulatedError += error;
  double accumulated = pid.accumulatedError*pid.Gains.i;
  double output = proportional+derivative+accumulated;
  return output;
}
void allStatusReset ( )
{
  allStatus.IO.analogOut.speedBias = 2.50;
  allStatus.IO.analogOut.voltageBias = 4.50;
  setting.RealLoadReference = 20000.0;
  setting.ReactiveLoadReference = 0.0;
  setting.PID.syncCheckSpeedBiasPID.accumulatedError = 0;
  setting.PID.syncCheckSpeedBiasPID.lastError = 0;
  setting.PID.syncCheckSpeedBiasPID.Gains.p = 0.005;
  setting.PID.syncCheckSpeedBiasPID.Gains.i = 0.0005;
  setting.PID.syncCheckSpeedBiasPID.Gains.d = 0.01;
  setting.PID.syncCheckVoltageBiasPID.accumulatedError = 0;
  setting.PID.syncCheckVoltageBiasPID.lastError = 0;
  setting.PID.syncCheckVoltageBiasPID.Gains.p = 0.08;
  setting.PID.syncCheckVoltageBiasPID.Gains.i = 0.0008;
  setting.PID.syncCheckVoltageBiasPID.Gains.d = 0.01;
  setting.PID.realPowerPID.accumulatedError = 0;
  setting.PID.realPowerPID.lastError = 0;
  setting.PID.realPowerPID.Gains.p = 0.05;
  setting.PID.realPowerPID.Gains.i = 0.002;
  setting.PID.realPowerPID.Gains.d = 0.02;
  setting.PID.reactivePowerPID.accumulatedError = 0;
  setting.PID.reactivePowerPID.lastError = 0;
  setting.PID.reactivePowerPID.Gains.p = 0.06;
  setting.PID.reactivePowerPID.Gains.i = 0.004;
  setting.PID.reactivePowerPID.Gains.d = 0.1;
}
double perturbation (double Amplitude)
{
  struct __timeb64 tstruct;
  _ftime64( &tstruct );
  double msec = tstruct.millitm;
  double results = Amplitude*sin (msec*PI/500.0);
  return results;
}
void PerturbationLoop ( )
{
  double speedBias = 2.50 + perturbation (0.1);
  if (speedBias != allStatus.IO.analogOut.speedBias)
  {
     allStatus.IO.analogOut.speedBias = speedBias;
     allStatus.IO.analogOutChanged = true;
  }
}
```

Appendix F to Part 2: Code for Protection and PLC Functions

[0632] A header file in an exemplary embodiment is:

```
class Protection
{
  private:
    double OilPressSensorInput,CoolTempSensorInput,EngineSpeedSensorInput;
    bool
e__stop__cre,EmergencyStop,auto__on__ss1,test__on__ss2,run__with__load__ss3,voltage__inc
rease__ss4,voltage__decrease__ss4;
    bool
speed__increase__ss6,speed__decrease__ss6,process__on__ss8,egcp2__engine__fuel__sol__on,
egcp2__fuel__sol21,egcp2__engine__crank;
    bool ecm__idle__rated__output,egcp2__visual__alarm,egcp2__idle__rated;
    bool egcp2__generator__cb__close,egcp2__generator__cb__trip;
    bool
egcp2__mains__cb__close,egcp2__mains__cb__trip,generator__cb__aux,mains__cb__aux;
    bool ecm__warning,ecm__fault__cr103,protective__relay__cr102;
    bool
crankcase__high__press__ps11,chp__coolant__pump__on__m401,chp__coolant__normal__flow__fls32,
chp__box__high__temp__ts;
    bool
system__fault,egcp2__auto__mode,egcp2__test__mode,egcp2__run__w__load__mode,egcp2__proce
ss__mode;
    bool egcp2__voltage__raise__mode,egcp2__voltage__lower__mode;
    bool egcp2__speed__raise__mode,egcp2__speed__lower__mode;
    bool generator__cb__close__cr24,generator__cb__trip__cr25,
egcp2__generator__cb__aux;
    bool mains__cb__close,mains__cb__trip, egcp2__mains__cb__aux;
    bool ecm__key__on__cr28, undelayed__ecm__key__on__cr28,
ecm__crank__cr29,idle__rated__delay;
    bool chp__box__vent__blower__fault, undelayed__chp__box__vent__blower__fault;
    bool chp__coolant__low__flow__fault,chp__coolant__pump__fault;
    bool egcp2__remote__fault__1,egcp2__remote__fault__2;
    bool ecm__warning__fault,undelayed__ecm__warning__fault;
    bool oil__level__low__fault,undelayed__oil__level__low__fault;
    bool protective__relay__fault,undelayed__protective__relay__fault;
    bool
protective__cb107__alarm__fault,undelayed__protective__cb107__alarm__fault;
    bool
ecm__fault,undelayed__ecm__fault,inlet__gas__low__psi__fault,undelayed__inlet__gas__low__psi__fault;
    bool crankcase__high__psi__fault,undelayed__crankcase__high__psi__fault;
    bool
undelayed__chp__coolant__pump__fault,undelayed__chp__coolant__low__flow__fault;
    bool chp__box__high__temp__flt,undelayed__chp__box__high__temp__flt;
    bool visual__alarm,reset__faults,last__egcp2__visual__alarm__status;
    void ModifyingCtrlInputs ( );
    void hardShutdown( );
    void softShutdown( );
    bool ResetFault( );
  public:
    Protection( );
    ~Protection ( );
    void doEvents ( );
    //need to change following functions.
    bool timerFunction__p2sec(bool q);
    bool timerFunction__1sec(bool q);
    bool timerFunction__2sec(bool q);
    bool timerFunction__3sec(bool q);
    bool timerFunction__5sec(bool q);
    bool timerFunction__10sec(bool q);
};
```

[0633] An implementation file is:

```
#include "Protection.h"
#include "Globals.h"
#using <mscorlib.dll>
using namespace System;
```

-continued

```
extern Variables allStatus;
extern Settings setting;
Protection::Protection( )
{
}
Protection::~Protection( )
{
}
void Protection::doEvents ( )
{
   ModifyingCtrlInputs( );
   hardShutdown ( );
   softShutdown ( );
}
void Protection::ModifyingCtrlInputs ( )
{
   auto__on__ss1 = Convert::ToBoolean(allStatus.IO.digitalIn.manualAutoSwitch);
   test__on__ss2 = Convert::ToBoolean(allStatus.IO.digitalIn.testEngineSwitch);
   run__with__load__ss3 =
Convert::ToBoolean(allStatus.IO.digitalIn.runWithLoadSwitch);
   voltage__increase__ss4 =
Convert::ToBoolean(allStatus.IO.digitalIn.voltageRaiseSwitch);
   voltage__decrease__ss4 =
Convert::ToBoolean(allStatus.IO.digitalIn.voltageLowerSwitch);
   speed__increase__ss6 =
Convert::ToBoolean(allStatus.IO.digitalIn.speedRaiseSwitch);
   speed__decrease__ss6 =
Convert::ToBoolean(allStatus.IO.digitalIn.speedLowerSwitch);
   generator__cb__aux =
Convert::ToBoolean(allStatus.IO.digitalIn.genCBAuxSwitch);
   mains__cb__aux = Convert::ToBoolean(allStatus.IO.digitalIn.mainsCBAuxSwitch);
   process__on__ss8 =
Convert::ToBoolean(allStatus.IO.digitalIn.ProcessModeSwitch);
   egcp2__remote__fault__1 =
Convert::ToBoolean(allStatus.IO.digitalIn.remoteFault1);
   egcp2__remote__fault__2 =
Convert::ToBoolean(allStatus.IO.digitalIn.remoteFault2);
   EmergencyStop = Convert::ToBoolean(allStatus.IO.digitalIn.remoteFault3);
   e__stop__cre = !EmergencyStop;
   egcp2__mains__cb__close =
Convert::ToBoolean(allStatus.IO.digitalOut.mainsBreaker);
   egcp2__mains__cb__trip =
Convert::ToBoolean(allStatus.IO.digitalOut.mainsBreakerTrip);
   egcp2__generator__cb__close =
Convert::ToBoolean(allStatus.IO.digitalOut.genBreaker);
   egcp2__generator__cb__trip =
Convert::ToBoolean(allStatus.IO.digitalOut.genBreakerTrip);
   egcp2__engine__fuel__sol__on =
Convert::ToBoolean(allStatus.IO.digitalOut.fuelSolenoid);
   egcp2__engine__crank =
Convert::ToBoolean(allStatus.IO.digitalOut.engineCrank);
   egcp2__idle__rated =
Convert::ToBoolean(allstatus.IO.digitalOut.idleRatedSwitch);
   egcp2__auto__mode = (e__stop__cre & (!system__fault)) & auto__on__ss1;
   egcp2__test__mode = (e__stop__cre & (!system__fault)) & test__on__ss2;
   egcp2__run__w__load__mode = (e__stop__cre & (!system__fault)) & run__with__load__ss3;
   egcp2__process__mode = (e__stop__cre & (!system__fault)) & process__on__ss8;
   egcp2__voltage__raise__mode = (e__stop__cre & (!system__fault)) &
voltage__increase__ss4;
   egcp2__voltage__lower__mode = (e__stop__cre & (!system__fault)) &
voltage__decrease__ss4;
   egcp2__speed__raise__mode = (e__stop__cre & (!system__fault)) &
speed__increase__ss6;
   egcp2__speed__lower__mode = (e__stop__cre & (!system__fault)) &
speed__decrease__ss6;
   generator__cb__close__cr24 = e__stop__cre & egcp2__generator__cb__close;
   generator__cb__trip__cr25 = egcp2__generator__cb__trip;
   egcp2__generator__cb__aux = generator__cb__aux;
   mains__cb__close = e__stop__cre & egcp2__mains__cb__close;
   mains__cb__trip = egcp2__mains__cb__trip;
   egcp2__mains__cb__aux= mains__cb__aux;
   egcp2__fuel__sol21=e__stop__cre & egcp2__engine__fuel__sol__on;
   ecm__key__on__cr28 = timerFunction__3sec(egcp2__fuel__sol21);
   ecm__crank__cr29 = e__stop__cre & egcp2__engine__crank;
   ecm__idle__rated__output=egcp2__idle__rated;
   idle__rated__delay = timerFunction__5sec(ecm__idle__rated__output);
```

-continued

```
   system__fault = egcp2__remote__fault__1 || egcp2__remote__fault__2;
   allstatus.IO.digitalIn.manualAutoSwitch =
Convert::ToInt16(egcp2__auto__mode);
   allStatus.IO.digitalIn.testEngineSwitch =
Convert::ToInt16(egcp2__test__mode);
   allStatus.IO.digitalIn.runWithLoadSwitch =
Convert::ToInt16(egcp2__run__w__load__mode);
   allstatus.IO.digitalIn.ProcessModeSwitch =
Convert::ToInt16(egcp2__process__mode);
   allStatus.IO.digitalIn.voltageRaiseSwitch =
Convert::ToInt16(egcp2__voltage__raise__mode);
   allStatus.IO.digitalIn.voltageLowerSwitch =
Convert::ToInt16(egcp2__voltage__lower__mode);
   allStatus.IO.digitalIn.speedRaiseSwitch =
Convert::ToInt16(egcp2__speed__raise__mode);
   allStatus.IO.digitalIn.speedLowerSwitch =
Convert::ToInt16(egcp2__speed__lower__mode);
   allStatus.IO.digitalIn.genCBAuxSwitch =
Convert::ToInt16(egcp2__generator__cb__aux);
   allStatus.IO.digitalIn.mainsCBAuxSwitch =
Convert::ToInt16(egcp2__mains__cb__aux);
   allstatus.IO.digitalOut.genBreaker =
Convert::ToInt16(generator__cb__close__cr24);
   allStatus.IO.digitalOut.genBreakerTrip =
Convert::ToInt16(generator__cb__trip__cr25);
   allStatus.IO.digitalOut.mainsBreaker = Convert::ToInt16(mains__cb__close);
   allStatus.IO.digitalOut.mainsBreakerTrip = Convert::ToInt16(mains__cb__trip);
   allStatus.IO.digitalOut.fuelSolenoid = Convert::ToInt16(ecm__key__on__cr28);
   allStatus.IO.digitalOut.engineCrank = Convert::ToInt16(ecm__crank__cr29);
   allStatus.IO.digitalOut.idleRatedSwitch =
Convert::ToInt16(idle__rated__delay);
}
void Protection:: hardShutdown( )
{
   CoolTempSensorInput =allStatus.IO.analogIn.CoolantTemp;
   EngineSpeedSensorInput = allStatus.IO.analogIn.MPUSpeed;
   EmergencyStop = Convert::ToBoolean(allStatus.IO.digitalIn.remoteFault3);
   egcp2__fuel__sol21 =
Convert::ToBoolean(allStatus.IO.digitalOut.fuelSolenoid);
   idle__rated__delay =
Convert::ToBoolean(allStatus.IO.digitalOut.idleRatedSwitch);
   e__stop__cre = (!EmergencyStop);
   ecm__fault__cr103 =
Convert::ToBoolean(allStatus.IO.digitalIn.remoteFault1);//Remote__Fault__1
   crankcase__high__press__ps11 =
Convert::ToBoolean(allStatus.IO.digitalIn.remoteFault3);//Emergency__stop
connected in series
   chp__coolant__pump__on__m401 =
Convert::ToBoolean(allStatus.IO.digitalIn.remoteFault5);//R__F__6
   chp__coolant__normal__flow__fls32 =
Convert::ToBoolean(allStatus.IO.digitalIn.remoteFault6);//R__F__6
   if    (CoolTempSensorInput > setting.ProtectionLimits.CoolTempMaxLimit)
   {
      allStatus.Alarms.limitsErrorCode = 1;
      allStatus.Alarms.hardShutdown = true;
   }
   if (EngineSpeedSensorInput > setting.ProtectionLimits.EngineSpeedMaxLimit)
   {
      allStatus.Alarms.limitsErrorCode = 2;
      allStatus.Alarms.hardShutdown = true;
   }
   if (Convert::ToBoolean(EmergencyStop) == true)
   {
      allStatus.Alarms.limitsErrorCode = 3;
      allStatus.Alarms.hardShutdown = true;
   }
   reset__faults = ResetFault( );
   egcp2__remote__fault__2 = ecm__fault || inlet__gas__low__psi__fault ||
crankcase__high__psi__fault || chp__coolant__pump__fault ||
chp__coolant__low__flow__fault || chp__box__high__temp__flt;
   if (egcp2__remote__fault__2 == true)
         {
            allStatus.Alarms.limitsErrorCode = 4;
            allStatus.Alarms.hardShutdown = true;
         }
         undelayed__ecm__fault = (ecm__fault || ((!ecm__fault__cr103) &
```

-continued

```
egcp2__fuel__sol21)) & (!reset__faults);
      ecm__fault = timerFunction__1sec(undelayed__ecm__fault);
      if ( ecm__fault == true)
      {
        allStatus.Alarms.limitsErrorCode = 5;
        allStatus.Alarms.hardShutdown = true;
      }
      undelayed__inlet__gas__low__psi__fault = (inlet__gas__low__psi__fault ||
egcp2__fuel__sol21) & (!reset__faults);
      inlet__gas__low__psi__fault =
timerFunction__p2sec(undelayed__inlet__gas__low__psi__fault);
      if ( inlet__gas__low__psi__fault == true)
      {
        allStatus.Alarms.limitsErrorCode = 6;
        allStatus.Alarms.hardShutdown = true;
      }
      undelayed__crankcase__high__psi__fault = (crankcase__high__psi__fault ||
((!crankcase__high__press__ps11 ) & egcp2__fuel__sol21)) & (!reset__faults);
      crankcase__high__psi__fault =
timerFunction__5sec(undelayed__crankcase__high__psi__fault);
      if ( crankcase__high__psi__fault == true)
      {
        allStatus.Alarms.limitsErrorCode = 7;
        allStatus.Alarms.hardShutdown = true;
      }
      undelayed__chp__coolant__pump__fault = (chp__coolant__pump__fault) &
(!reset__faults);
      chp__coolant__pump__fault =
timerFunction__10sec(undelayed__chp__coolant__pump__fault);
      if ( chp__coolant__pump__fault == true)
      {
        allStatus.Alarms.limitsErrorCode = 8;
        allStatus.Alarms.hardShutdown = true;
      }
      undelayed__chp__coolant__low__flow__fault = (chp__coolant__low__flow__fault ||
(!chp__coolant__normal__flow__fls32)) & (!reset__faults);
      chp__coolant__low__flow__fault =
timerFunction__10sec(undelayed__chp__coolant__low__flow__fault);
      if ( chp__coolant__low__flow__fault == true)
      {
        allStatus.Alarms.limitsErrorCode = 9;
        allStatus.Alarms.hardShutdown = true;
      }
      undelayed__chp__box__high__temp__flt = (chp__box__high__temp__flt ||
(!chp__box__high__temp__ts)) & (!reset__faults);
      chp__box__high__temp__flt =
timerFunction__5sec(undelayed__chp__box__high__temp__flt);
      if ( chp__box__high__temp__flt == true)
      {
        allStatus.Alarms.limitsErrorCode = 10;
        allStatus.Alarms.hardshutdown = true;
      }
      allStatus.IO.digitalIn.remoteFault2 =
Convert::ToInt16(egcp2__remote__fault__2);
    }
void Protection::softShutdown( )
{
  OilPressSensorInput =allStatus.IO.analogIn.OilPressure;
  egcp2__fuel__sol21 =
Convert::ToBoolean(allStatus.IO.digitalOut.fuelSolenoid);
  idle__rated__delay =
Convert::ToBoolean(allStatus.IO.digitalOut.idleRatedSwitch);
  EmergencyStop = Convert::ToBoolean(allStatus.IO.digitalIn.remoteFault3);
  e__stop__cre = (!EmergencyStop);
  protective__relay__cr102 =
Convert::ToBoolean(allStatus.IO.digitalIn.remoteFault4);//R__F__4
  if    ((OilPressSensorInput > setting.ProtectionLimits.OilPressMaxLimit) ||
      (OilPressSensorInput < setting.ProtectionLimits.OilPressMinLimit))
  {
    allStatus.Alarms.limitsErrorCode = 11;
    allStatus.Alarms.softShutdown = true;
  }
  reset__faults = ResetFault( );
  egcp2__remote__fault__1 = ecm__warning__fault || oil__level__low__fault ||
protective__relay__fault || protective__cb107__alarm__fault ||
chp__box__vent__blower__fault;
```

-continued

```
   if (egcp2__remote__fault__1 == true)
   {
      allStatus.Alarms.limitsErrorCode = 13;
      allStatus.Alarms.softShutdown = true;
   }
   undelayed__ecm__warning__fault = (ecm__warning__fault || (egcp2__fuel__sol21 &
ecm__warning)) & (!reset__faults);
   ecm__warning__fault = timerFunction__2sec(undelayed__ecm__warning__fault);
   if (ecm__warning__fault == true)
   {
      allStatus.Alarms.limitsErrorCode = 14;
      allStatus.Alarms.softShutdown = true;
   }
   undelayed__oil__level__low__fault = (oil__level__low__fault) & (!reset__faults);
   oil__level__low__fault = timerFunction__2sec(undelayed__oil__level__low__fault);
   if (oil__level__low__fault == true)
   {
      allStatus.Alarms.limitsErrorCode = 15;
      allStatus.Alarms.softShutdown = true;
   }
   undelayed__protective__relay__fault = (protective__relay__fault ||
protective__relay__cr102) & (!reset__faults);
   protective__relay__fault =
timerFunction__2sec(undelayed__protective__relay__fault);
   if (protective__relay__fault == true)
   {
      allStatus.Alarms.limitsErrorCode = 16;
      allStatus.Alarms.softShutdown = true;
   }
   undelayed__protective__cb107__alarm__fault = (protective__cb107__alarm__fault ) &
(!reset__faults);
   protective__cb107__alarm__fault =
timerFunction__2sec(undelayed__protective__cb107__alarm__fault);
   if ( protective__cb107__alarm__fault == true)
   {
      allStatus.Alarms.limitsErrorCode = 17;
      allStatus.Alarms.softShutdown = true;
   }
   undelayed__chp__box__vent__blower__fault = chp__box__vent__blower__fault &
(!reset__faults);
   chp__box__vent__blower__fault =
timerFunction__10sec(undelayed__chp__box__vent__blower__fault);
   if ( chp__box__vent__blower__fault == true)
   {
      allStatus.Alarms.limitsErrorCode = 18;
      allStatus.Alarms.softShutdown = true;
   }
   allStatus.IO.digitalIn.remoteFault1 =
Convert::ToInt16(egcp2__remote__fault__1);
}
bool Protection::ResetFault( )
{
         egcp2__visual__alarm = allStatus.Alarms.hardShutdown ||
allStatus.Alarms.softShutdown;
         if (reset__faults == false)
         {
            if (egcp2__visual__alarm == false && last__egcp2__visual__alarm__status
== false)
               reset__faults = true;
         }
         else
         {
            if (egcp2__visual__alarm == true && last__egcp2__visual__alarm__status
== false)
               reset__faults = false;
         }
         last__egcp2__visual__alarm__status = egcp2__visual__alarm;
         return reset__faults;
      }
bool Protection::timerFunction__p2sec(bool q)
      {
         int counter = 0;
         while (q == true)
         {
            counter ++;
            if (counter >= 720*0.2)
```

-continued

```
            {
                q = false;
                return true;
            }
        }
        return false;
    }
bool Protection::timerFunction__1sec(bool q)
    {
        int counter = 0;
        while (q == true)
        {
            counter ++;
            if (counter >= 720)
            {
                q = false;
                return true;
            }
        }
        return false;
    }
bool Protection::timerFunction__2sec(bool q)
    {
        int counter = 0;
        while (q == true)
        {
            counter ++;
            if (counter >= 720*2)
            {
                q = false;
                return true;
            }
        }
        return false;
    }
bool Protection::timerFunction__3sec(bool q)
    {
        //This timer is a TOF type timer function; therefore, the output of
the function is different than the others.
        int counter = 0;
        while (q == true)
        {
            counter ++;
            if (counter >= 720*3)
            {
                q = false;
                return false;
            }
        }
        return true;
    }
bool Protection::timerFunction__5sec(bool q)
    {
        int counter = 0;
        while (q == true)
        {
            counter ++;
            if (counter >= 720*5)
            {
                q = false;
                return true;
            }
        }
        return false;
    }
bool Protection::timerFunction__10sec(bool q)
    {
        int counter = 0;
        while (q == true)
        {
            counter ++;
            if (counter >= 720*10)
            {
                q = false;
                return true;
            }
```

-continued

```
        }
        return false;
    }
```

Appendix G to Part 2: Code for Engine and Generator Control

[0634]

```
Class Control
{
    public:
        Control ( );
        ~Control ( );
        int doEvents ( );
        void speedBias ( );
        void voltageBias ( );
};
#include "Globals.h"
#include "Control.h"
#define SPEED__BIAS__CONSTANT 2.50
#define VOLTAGE__BIAS__CONSTANT 4.50
extern Variables allStatus;
Control::Control ( )
{
    allStatus.IO.analogOut.speedBias = SPEED__BIAS__CONSTANT;
    allStatus.IO.analogOut.voltageBias = VOLTAGE__BIAS__CONSTANT;
    allStatus.IO.analogOutChanged = true;
    for (int i = 0; i < DIGITAL__OUT; i ++)
        allStatus.IO.digitalOut.value[i] = 0;
    allStatus.IO.digitalOut.audibleAlarm = 0;
    allStatus.IO.digitalOut.engineCrank = 0;
    allStatus.IO.digitalOut.genBreakerTrip = 0;
    allStatus.IO.digitalOutChanged = true;
}
Control::~Control ( )
{
}
int Control::doEvents ( )
{
    voltageBias ( );
//   speedBias( );
    return 0;
}
void Control::voltageBias ( )
{
    double voltageBias = 0.0;
    double Range = 10.0;
    double Limit = 2.0;
    double tolerance = 6.0;
    double p = 0.1;
    if (!allStatus.Modes.GPM)
    {
        return;
    }
    else
    {
        if (allStatus.Modes.SyncCheck)
        {
            voltageBias = allStatus.PMU.voltageGridA.Amp−
allStatus.PMU.voltageLocalA.Amp;
            if (voltageBias < tolerance && voltageBias > −1.0*tolerance)
            {
                return;
            }
            else
                {
            voltageBias = voltageBias * p;
        }
    }
    }
```

-continued

```
      if (voltageBias > Range) voltageBias = Range;
      if (voltageBias < −1.0*Range) voltageBias = −1*Range;
      voltageBias = voltageBias * Limit/Range;
      allStatus.IO.analogOut.voltageBias = voltageBias;
}
void Control::speedBias ( )
{
/*    double speedBias = SPEED_BIAS_CONSTANT;
      double error = 0;
      double Range = 10.0;
      double Limit = 2.0;
      if (allStatus.Status.GPM)
      {
            if (allStatus.Status.SyncCheck)
            {
                  error = allStatus.PMU.voltageGridA.Phase−
allStatus.PMU.voltageLocalA.Phase;
                  double proportional =
error*allStatus.Data.PowerLoopPID.Gains.p ;
                  double derivative = (error −
allStatus.Data.PowerLoopPID.lastError) * allStatus.Data.PowerLoopPID.Gains.d;
                  allStatus.Data.PowerLoopPID.lastError = error;
                  allStatus.Data.PowerLoopPID.accumulatedError += error;
                  double accumulated =
allStatus.Data.PowerLoopPID.accumulatedError*allStatus.Data.PowerLoopPID.Gains.i;
                  speedBias = proportional+derivative+accumulated;
            }
      }
      if (speedBias > Range) speedBias = Range;
      if (speedBias < −1.0*Range) speedBias = −1*Range;
      speedBias = speedBias * Limit/Range+2.50;
      allStatus.IO.analogOut.speedBias = speedBias;*/
}
```

What is claimed is:

1. A power management system, comprising:

a heterogeneous plurality of generators powering a load; and

a controller that controls the power output of each of the plurality of generators

at a first point in time as a function of a first rule; and

at a second point in time as a function of two or more fuzzy rules.

2. The system of claim **1**,

further comprising an input signal to the controller characterizing the amount of power demanded by the load; and

wherein the controller produces one or more output signals that collectively indicate the power to be output by each of the plurality of generators.

3. The system of claim **1**, wherein the controller comprises a plurality of controller units.

4. A distributed power generation system adapted to interface with an electric utility distribution system at a point of common coupling, comprising:

a plurality of distributed power generation resources, collectively having an aggregate output rating; and

a controller;

wherein the distributed power generation resources export power at a first time to the electric utility distribution system, and

wherein, if a first portion of the electric utility distribution system that includes the distributed power generation resources becomes electrically disconnected from the remainder of the electric utility distribution system, the controller causes the distributed power generation resources to export power to the first portion of the electric utility distribution system for not more than about two seconds.

5. The system of claim **4**, wherein the distributed power generation system complies with IEEE Standard 1547, section 4.4.

6. The system of claim **4** wherein, while the first portion of the electric utility distribution system is electrically disconnected from the remainder of the electric utility distribution system, the controller prevents the distributed power generation resources from beginning to export power to the first portion of the electric utility distribution system.

7. The system of claim **4**, further comprising:

a decision tree that is trained at least in part on training data characterizing the electrical characteristics during HIF and non-HIF conditions at the point of common coupling, then configured to accept real data characterizing the electrical characteristics at the point of common coupling and provide an HIF detection output; and

a control system that modifies the operation of the distributed power generation system based on the HIF detection output.

8. The system of claim **7**, w herein the training data and real data are measurements of one or more currents at the point of common coupling.

9. A method, comprising:

measuring the current through or line-to-neutral voltage of the electrical signals on each line in a three-phase AC circuit;

computing from the measurements phasors associated with the electrical signals; and

communicating data relating to the phasors with a controller via a Controller Area Network (CAN).

10. The method of claim **9**, wherein the measuring occurs at a phase of the AC signal offset from a most recent sample on the same respective line by an angle selected from the group consisting of 30°, 60°, and 120°.

11. The method of claim **9**, further comprising determining from the symmetrical components of the phasors the distance from the point of measurement to the fault relative to the length of the transmission line.

12. The system of claim **9**, wherein the data relating to the phasors includes the magnitude and direction of either the current or voltage of each phase of the AC circuit.

13. A device, comprising:

a distributed generation system controller (DGSC) for controlling the power outputs of a plurality of generators; and

a utility interface controller (UIC) that selectively connects the plurality of generators to a utility power grid and to an electrical load;

wherein the DGSC and UIC are contained in a single enclosure.

14. The system of claim **13**, further comprising:

a decision tree that is trained at least in part on training data characterizing the electrical characteristics at the point of common coupling during HIF and non-HIF conditions, then configured to accept real data characterizing the electrical characteristics at the point of common coupling and provide an HIF detection output; and

a control system that modifies the operation of the distributed power generation system based on the HIF detection output.

15. The system of claim **13**, wherein the training data and real data are measurements of one or more currents at the point of common coupling.

16. A kit, comprising:

a generator; and

a device according to claim **13**.

17. A system comprising:

a device according to claim **13**; and

one or more distributed generation units, each including:

a distributed generation controller in communication with the DGSC;

a generator controlled by the distributed generation controller;

wherein power output by the generator in each distributed generation unit supplies

a load at a first point in time, and

the utility power grid at a second point in time.

18. The system of claim **17**, wherein at least the generator in at least one distributed generation unit supplies both the load and the utility power grid at the same point in time.

19. A distributed generation subsystem that detects islanding of the subsystem, comprising:

a plurality of generators that produce AC output power for a load, in parallel with a power grid that has a first frequency characteristic; and

a controller that controls the output level of each of a plurality of generators using one or more control signals;

wherein the controller superimposes a perturbation signal on the control signals,

the controller detects the frequency characteristic of the AC output power;

the effect of the perturbation on the AC output power has a second frequency characteristic that is detectably different from the first frequency characteristic; and

the controller produces a first signal if the detected frequency characteristic is sufficiently like the first frequency characteristic, and a second signal if the detected frequency characteristic is sufficiently like the second frequency characteristic.

20. The subsystem of claim **19**, wherein:

in response to the first signal, the controller allows export of power from the generators to the power grid; and

in response to the second signal, the controller

ceases export of power from the generators to the power grid, if such export is occurring when the second signal begins to be produced, and

prevents export of power from the generators to the power grid while the second signal is being produced.

21. The subsystem of claim **20**, wherein the cessation of the export of power from the generators to the power grid occurs within two seconds of the time at which the second signal begins to be produced.

22. The subsystem of claim **20**, wherein the cessation of the export of power from the generators to the power grid occurs within two seconds of the time at which the frequency characteristic sufficiently like the second frequency characteristic begins to be detected.

23. The subsystem of claim **19**, wherein the controller comprises a supervisory controller and, for each generator, a subordinate controller that receives control signals from the supervisory controller and controls the output level of that generator.

* * * * *