DPJ391UE

Contactor Unit Design, Implemented As Part Of The Utility Load Management Project

By
Jacob Jacobus Greeff (3692-247-1)

In partial fulfillment of the requirements
for the Ndip: Electrical Engineering

Table of Contents

1. ForeWord

2. Conceptual Design Report

3. Detailed Design Report

4. Design Implementation Report

5. Appendix: Firmware

Foreword:

I would like to thank all of the people who have made this thesis, and in fact all of my studies at Unisa over the years possible. First and foremost I would like to thank my wife Carol, whose constant support has been invaluable over the years we have been married, but especially so during stressful times. I would like to thank my mentor mr. Nicholas Prozesky for his help and guidance throughout the years we have worked together, and especially on his project. I would also especially like to thank my manager mr. Andrew Goedhart who has been a great source of knowledge and guidance in my career.I am also grateful to Unisa who have afforded me the opportunity to further my studies in Engineering and I hope to continue my academic career through them in the future.

DPJ391UE

Conceptual Design Report

By
Jacob Jacobus Greeff (3692-247-1)

In partial fullfulment of the requirements
for the Ndip: Electrical Engineering

Table of Contents

1 Confidentiality....2
2 Background....3
2.1 Outline....3
2.2 Specifications....5
3 Design Concepts....6
3.1 Processor....6
3.2 Analogue measurement....7
3.3 Power Supplies....7
3.4 Contactor....7
3.5 Memory....7
3.6 Interfaces....7
4 Conclusions....9
5 Contact List....10
6 Bibliography....11

1 Confidentiality

At the current time I am employed as a Design Engineer By Gemini Moon consulting, and the Project I am working on is in collaboration with Eon Technologies. Due to the sensitive nature of some of the intellectual property, I have been requested to keep my involvement in this project confidential. I therefor request that all design information contain herein as well as in future progress reports will be treated confidentially by TSA and its examiners and lecturers. I will include a list of the relevant parties at the end of this report so that should any queries arise, they can dealt with accordingly.

2 Background

The project that I will be implementing forms a part of a much bigger project with the goal of alleviating some of the problems caused by the current power crisis in South Africa. The Project forms part of the Load limiter project. A short outline of the project follows:

2.1 Outline

Eskom is currently having massive problems to supply the required amount of electricity to the whole country. They have identified the need to drop the consumption across the board by a nominal 10-15%. This has up till this point been achieved by using controlled load shedding. This process however has proven difficult to manage as well as placing considerable strain on the infrastructure of the electrical network. The goal of the load limiter project is to alleviate the problems caused by load shedding in the following ways:

- Create a centralized point where the status of the network can be efficiently monitored, and the required dynamic consumption changes can be enforced.
- Alleviate strain from the current infrastructure by allowing each house to be load limited separately.
- Alleviate consumer frustration by making the load management process a controlled and proportional shed, rather than an across the board uncontrolled shed.
- Provide a means by which critical power users (such as hospitals) can ensure that they do not form part of the load shedding network.

These are not the only goals that need to be reached, but they form a general scope from which the different parts of the Load shedding network can be identified. The network layout is as follows:

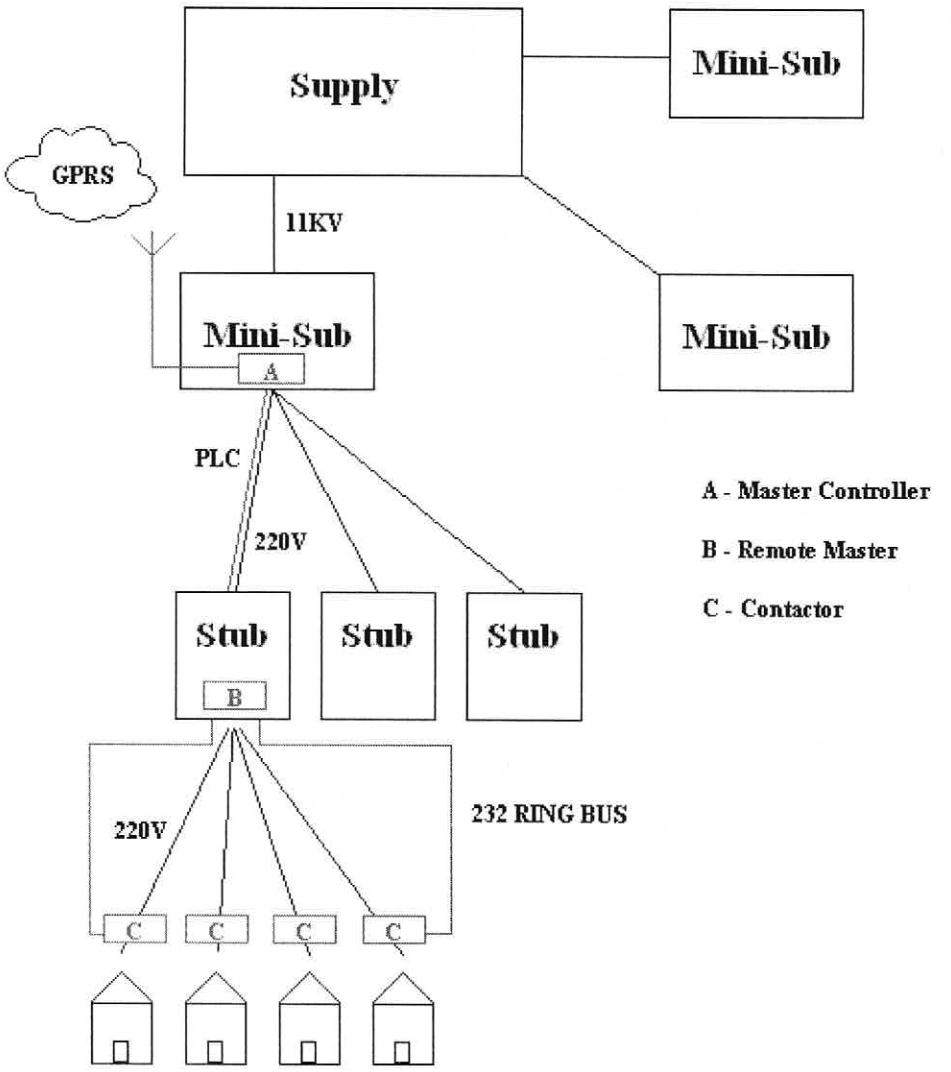


Figure 2.1.1 – Power Network and Data network

Power is supplied from the main utilities through the High voltage networks to the 11KV-230V Mini Sub Transformers. At this point, we install a remote master unit. The remote master unit will be responsible for collecting information about all the devices connected to it. The Remote Master maintains a GPRS connection to a main server, and has a PLC (Power Line Carrier) network between itself and all the Remote master units. The power in the minisub gets converted from 11KV to 230V Ac, where it gets distributed to a number of Stubby Boxes. These Stubbies are the distribution points to each of the houses connected onto the electrical network. At this point we install a remote master. The remote master is responsible for collecting information from a number of Contactor Devices which are also connected in the stubby. The remote master has a PLC interface for connecting to the Master controller, and also has an RS232 ring bus that connects to a number of contactor devices. The contactor devices are in effect electricity meters that collect the consumption information about the house they are connected to. The main function of the contactor device however comes from its name, in that all the contactor devices contain a 100A 230V contactor which can be used to cut the mains supply to the house. The final part of the network is the display units, which are basically just dumb

terminals connected onto the PLC network which give the user the ability to find out about load shedding periods and compliance to load shedding measures. The display devices do not contribute any new information into the system and are therefor not included in the data network model.

The part of the project I will be working on is the Contactor device.

2.2 Specifications

The contactor device needs to conform to the following specifications:

- The wiring Code for south Africa (SANS 10142)
- The metering standard for single phase domestic houses for south Africa as well as the code of practices (SANS 62052) (SANS 474)
- The contactor must be able to handle 100A switched current for a minimum period of ten years
- The memory capacity of the unit must be able to hold for a minimum period of ten years
- The device must be able to fit onto a standard DIN rail (mechanical design has been done)
- The device must have a debug port
- The device must be able to interface onto the RS232 ring bus to the remote master

On the technical side, the following restraints must be taken into account:

- The power supply to the contactor needs to float on one of the feed lines (Phase or neutral) to allow measurement of the power signal.
- The power supply will also need to conform to the 4VA max power drawn specification for meters.
- All the debug and serial lines will need to be opto coupled to ensure the safety of users and equipment.
- In order to increase the life of the contactor, the switching needs to only be done on the zero crossing point on the of the 50 Hz wave. This will need to be calibrated for every mechanical contact.

3 Design Concepts

Based on the requirements of the product, the following modules can be identified:

- Processor
- Power supply (s)
- Analog measurement (metering)
- Contactor
- Memory
- Interface (s)

A case study follows which evaluates the different parts of the system.

3.1 Processor

The micro processor on the device needs to have a low power consumption, but also needs to have the following specifications :

- The processor will need at least one UART and one other appropriate bus interfaces in order to fulfill the requirements of the system (debug, and RS232 ring bus)
- The processor needs to have a bus that will allow an interface to the memory module.
- The processor needs to have sufficient Io pins to drive the contactor and any other required io devices.
- The processor needs to have sufficient RAM/ROM to run the EOS operating system. EOS is a parallel project that I am working on which is the base operating system for each of the embedded devices.

The following processors have been identified as possible candidates:

Device	*Memory*	*Peripherals*	*Power Consumption*	*Price*
STM32F103	*1*	*1*	*2*	*1*
LPC2103	*3*	*3*	*3*	*3*
PIC32F320F	*2*	*2*	*4*	*4*
STM08A	*4*	*4*	*1*	*2*

Based on the comparisons of different processor, I decoded to go with the STM32. In addition to those advantages listed above, the processor uses the new cortex core, and is exceptionally fast. It is also very well priced for such a powerful processor. The second best option would be the STM8 series of processors, but due to the fact that they are 8 bit and not 32, they will not be compatible with the EOS operating system. I will however definitely be using them in future projects due to their exceptionally low power consumption.

3.2 Analogue measurement

The electrical measurement will be done using a measurement IC. The three options are:

- STPM11
- SA2002H
- ADE7755

After looking at the datasheets, all three these chips offer what I need for my design, and in the end I decided to use the SA2002H since I have used SAMES measurement chips on a previous project and found them quite easy to work with.

3.3 Power Supplies

When looking at the product in terms of modules it's found that 2 power supplies will be needed, one for the floating analog circuitry which will measure the power consumption and wave form analysis, and the other for the microprocessor, interfaces and memory. Different supplies have been proposed, and through testing on proto board, the LM317/LM337 supply and the switch mode supplies were chosen. This decision was made as I have already tested and used these supplies in other projects and have found them to work work very well. The only difference is that I also tested out using the floating power supply from the sames reference design, but that was found to not be able to supply enough current. I am in the process of laying out the pcbs and will include circuits and gerbers in my next progress report.

3.4 Contactor

With the help of the mechanical engineer, Mr Jan Olwagen, It was decided that we would use the American Zettler series of Contactors with 12V coil and 1,44W switch power. The contactors conform to all the required specifications. The drive circuit to the contactor will be a standard H bridge as the coil requires current in one direction to open and in the opposite direction to close. The two proposed flyback protection circuits are :

- Crowbar protection
- RC Snubber

After testing each of these circuits on the bench, we chose to use the crowbar protection as the RC snubbers tends to slow down the switching process which complicates the calibration for the zero cross switching.

3.5 Memory

The memory chip which will store all the configuration and the wh counter is the 24LC01 by microchip. This chip was chosen as drivers have already been written for another project using EOS.

3.6 Interfaces

To conform to the specification set out about the RS232 ring bus and the debug port on all the other

devices, the connectors to the interface will be RJ45, and both the ports will be RS232. The interface to the memory chip is I^2C.

4 Conclusions

Based on the above discussion, I am going ahead with the design of the circuit board for the unit in conjunction with Mr. Nicholas Prozesky and Mr. Jan Olwagen. Although I am using the contactor as my design project, the three of us in addition to one more Engineer which still needs to be employed will make up the hardware team on the load limiter project. The goal for finishing all of these hardware products will be the end of august for a test rollout.

5 Contact List

- Jacob Greeff — Engineer on Load limiter project
  email: japieg@gmail.com — Tel : 078 214 6148
- Nicholas Prozesky — Engineer on Load limiter project
  email: nicholasp@eon.co.za — Tel : 083 798 7408
- Andrew Goedhart — Development Leader
  email: andrewg@eon.co.za — Tel : 084 556 7711

6 Bibliography

1. K. Billings, Switchmode power supply handbook. New York: McGraw-Hill, 1989, pp.1.89-1.100
2. S.D'Souza, TB008 Transformerless Power Supply. Microchip semiconductor, [Online] Available: www.microchip.com
3. Reston Condit, AN954 Transfomerless Power Supplies: Resistive and Capacitive. Microchip semiconductor, [Online] Available: www.microchip.com
4. SANS62051, "1.00 Electricity metering – Glossary of terms," SABS, 1999
5. SANS10142, "The wiring of premises, Part1: Low-voltage installations", SABS, 2003
6. E. Hughes, Electrical Technology. Edinburgh Gate: Addison Wesley Longman Limited, 1998

DPJ391UE

Detailed Design Report

By
Jacob Jacobus Greeff (3692-247-1)

In partial fulfillment of the requirements
for the Ndip: Electrical Engineering

Table of Contents

1. Update....3
2. Detailed Design....4
2.1. Contactor Logical Breakdown....4
2.2. Display Logical Breakdown....4
2.3. Contactor Block diagram....5
2.3.1. Current Shunt:....5
2.3.2. Power Sense Circuitry:....6
2.3.3. Zero Cross Sense....8
2.3.4. Floating +2.5V and -2.5V Power Supply....10
2.3.5. Opto Isolation....11
2.3.6. Contactor Driver....12
2.3.7. Power Supply (+3.3V and +12V)....13
2.3.8. EEPROM storage....14
2.3.9. Flash Storage....15
2.3.10. Ring Bus communication....16
2.3.11. Display Leds....17
3.1. Mechanical Design....18
4.1. Design Information – Schematics and Pcb Documents....19
4.2.1. EON000 Measurement Schematic....20
4.2.2. EON001 Power Supply Schematic....21
4.2.3. EON001 IO Schematic....22
4.2.4. EON001 Processor Schematic....23
4.3.1. EON000 Top Pcb Layer....24
4.3.2. EON000 Top Pcb Overlay....25
4.3.3. EON000 Bottom Pcb Layer....26
4.3.4. EON000 Bottom Pcb Overlay....27
4.3.5. EON001 Top Pcb Layer....28
4.3.6. EON001 Top Pcb Overlay....29
4.3.7. EON001 Bottom Pcb Layer....30
4.3.8. EON001 Bottom Pcb Overlay....31
5. Contact List....32
6. Bibliography....33

1. Update

On further investigation into the proposed design, a number of issues have been raised and have had to be dealt with in order to make a success of the project. The issue that most impacts the Contact unit that I am using as my project is the fact that in consultation with the Sabs on The metering standard for single phase domestic houses for south Africa as well as the code of practices (SANS 62052) (SANS 474), it has come the the for that for our contactor unit to function as an electricity meter, it would need both a measurement component as well as a display component. We have therefor incorporated a new component into our system that will allow the displaying of the appropriate information. This new new component will be called the display unit.

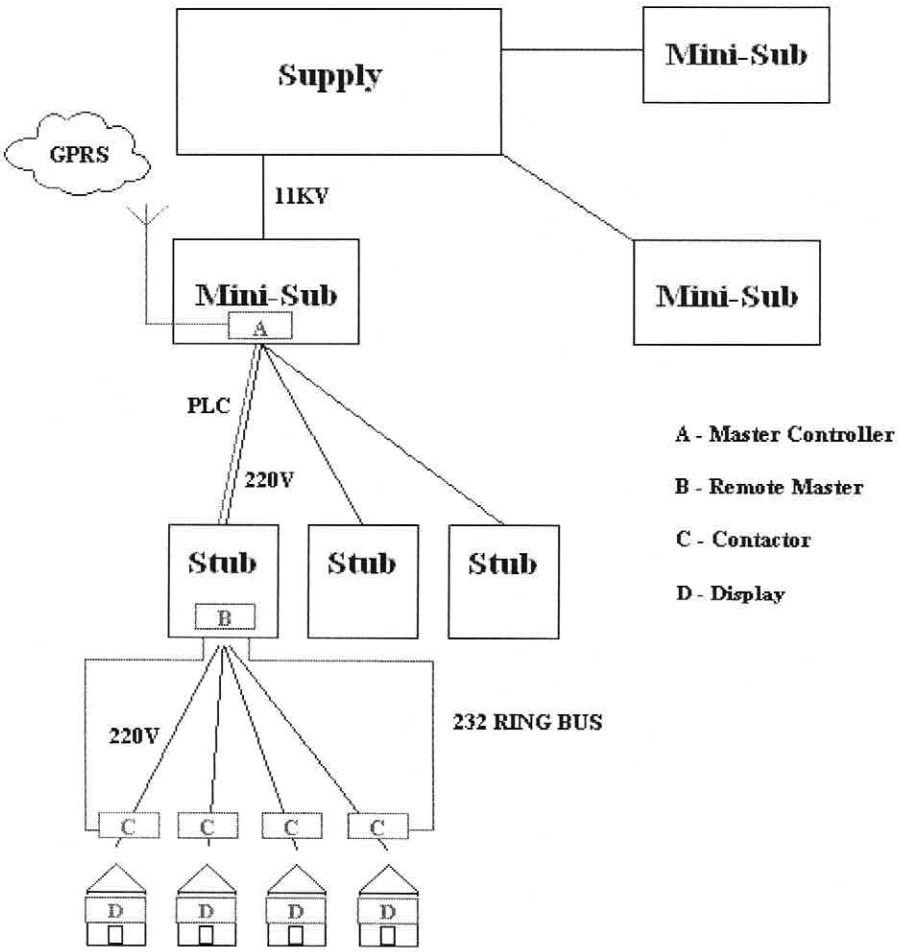


Figure 1. System overview including display unit

2. Detailed Design

In this section I will be evaluating the two devices. The evaluation will be done initially by breaking the devices up into their logical functionality, then by looking at the block diagram design of the devices, and finally I will look at the circuit diagrams and Pcb gerbers.

2.1. Contactor Logical Breakdown

The contactor device has the following responsibilities in the system.

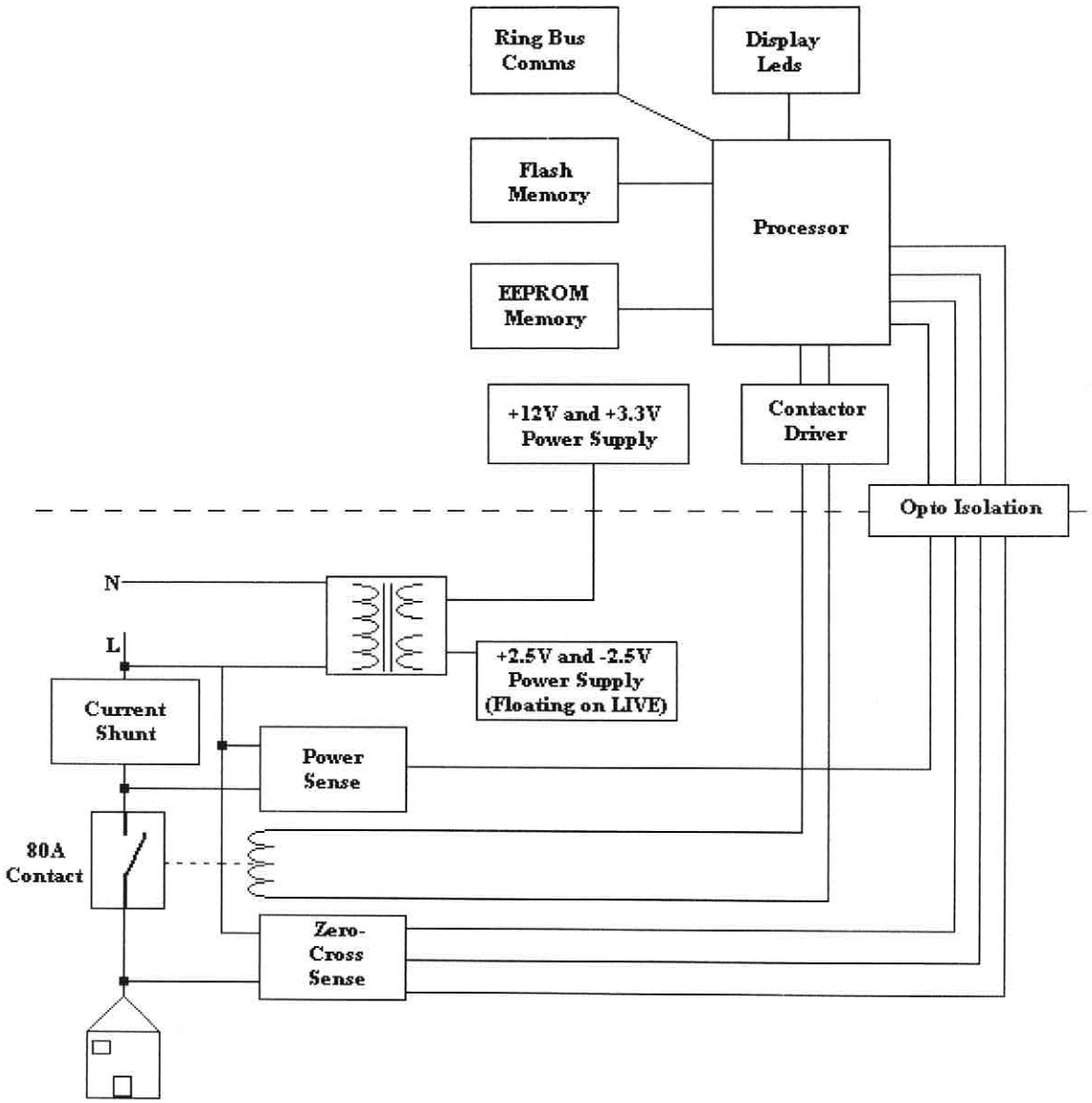
- It needs to be able to interrupt the supply of power to the house that it is connected to. The switching of power to the house needs to be done on the zero point of the ac waveform.
- It needs to connect to the external Serial ring bus network to it's Remote master unit to propagate information to the rest of the system.
- It needs to measure the amount of power consumed by the household. (data to be sent onto the ring bus.)

2.2. Display Logical Breakdown

The Display device has the following responsibilities in the system.

- Displaying information about the status of the system.
- Notifying the user of messages and load limiting periods in effect on the system.

2.3. Contactor Block diagram



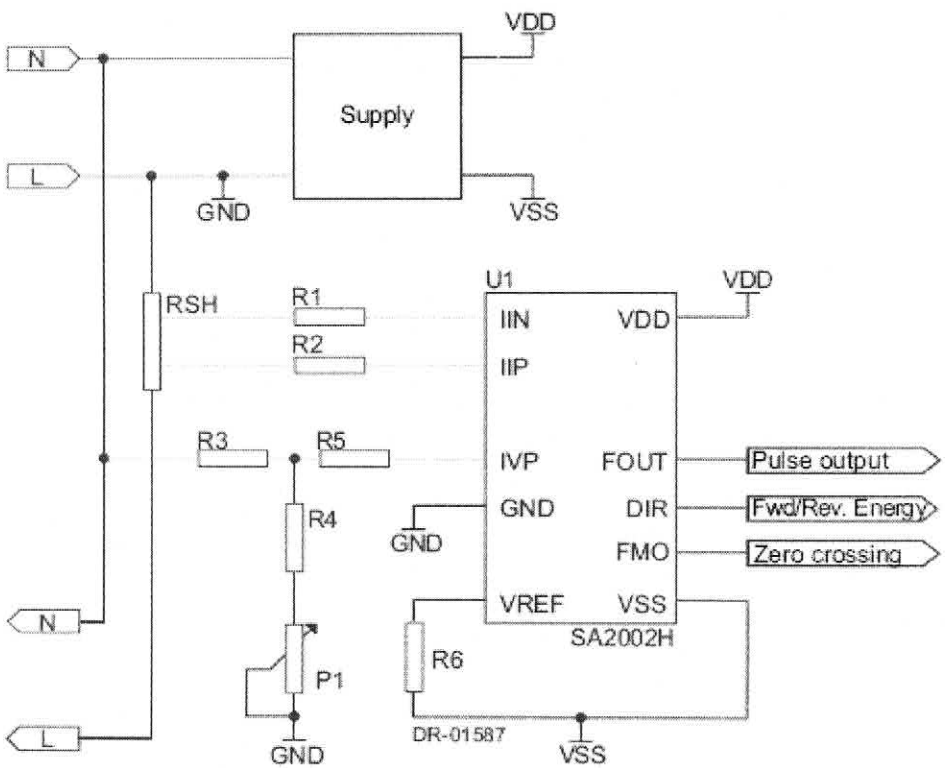
Breakdown of blocks in the system:

2.3.1. Current Shunt:

The current shunt is a 10W Series resistor composed of two 0.001 Ohm/5W surface mount resistors in parallel. The voltage drop across the shunt is proportional to the current flowing through the contactor and is fed into the power sense circuitry to determine the power consumption.

2.3.2. Power Sense Circuitry:

The power sense circuitry is based around a Sames SA2002 measurement chip. On the front end, the voltage drop across a shunt resistor across two series resistors (R40 and R41 in the design) is used to measure the amount of current flowing through the contactor (II, measured from IIP to IIN) and a resistor divider circuit is used to drop the line voltage from 230V rms to 15.7V rms. The voltage is measured by feeding into the Sames chip through a 1M resistor (IVP). This puts the input current at 15.7 uA for the a/d converter (nominal 14 uA). From the datasheet [1]:



$IVP_{ref} = ((V_{line} * (R4+P1)/(R4+R3+P1)))/R5$

On the circuit diagram EON000 (annexure A), the resistor Divider has been set up with R80,R90,R91,R92 replacing R3 from the reference design, and R60 and R30 replacing R4 and P1 in the reference design. The reason for the large amount of resistors is that all the resistors used are SMD resistors with a rated break over voltage of 200V maximum and a rated power consumption of 250mW. The decision was made to split the resistors up since the voltage drop across R3 in the reference design will be 215V and constant power dissipation approx. 150mW. The potentiometer was removed as all calibration will be done in software and not in the hardware. In my design the current limiting resistor R5 is a 1M resistor labeled R120. The design only measures power consumed and rate of consumption, and not fwd/reverse energy. A separate Zero crossing circuit is used for the zero cross switching because even though the zero crossing circuitry in the SA2002H guarantees that the zero crossing pulse will happen at the same point in each waveform, it does not guarantee that it will happen at the exact point of zero cross, so it is only useful for measuring frequency.

1 Single Phase Bi-directional Power / Energy Metering IC with Instantaneous Pulse Output SA2002H, Sames Semiconductor. [Online] Available

Calculations:

VP_{design} = (V_{line}* (R30+R60)/(R30 + R60 + R80 + R90 + R91 + R92))

= (230 * (2K4 + 22K)/(2K4 + 22K + 33K + 100K + 100K + 100K))

= 15.702V

IVP_{design} = VP_{design} / R120

= 15.702 / 1000000

= 15.702 uA

V_{shunt} = I_{max} * R_{shunt}

= I_{max} * ((R00*R01)/(R00+R01))

= 80 * ((0.001*0.001)/(0.001+0.001))

= 40mV

II_{design} = Vshunt / (R40 + R41)

= 40mV / (820 * 820)

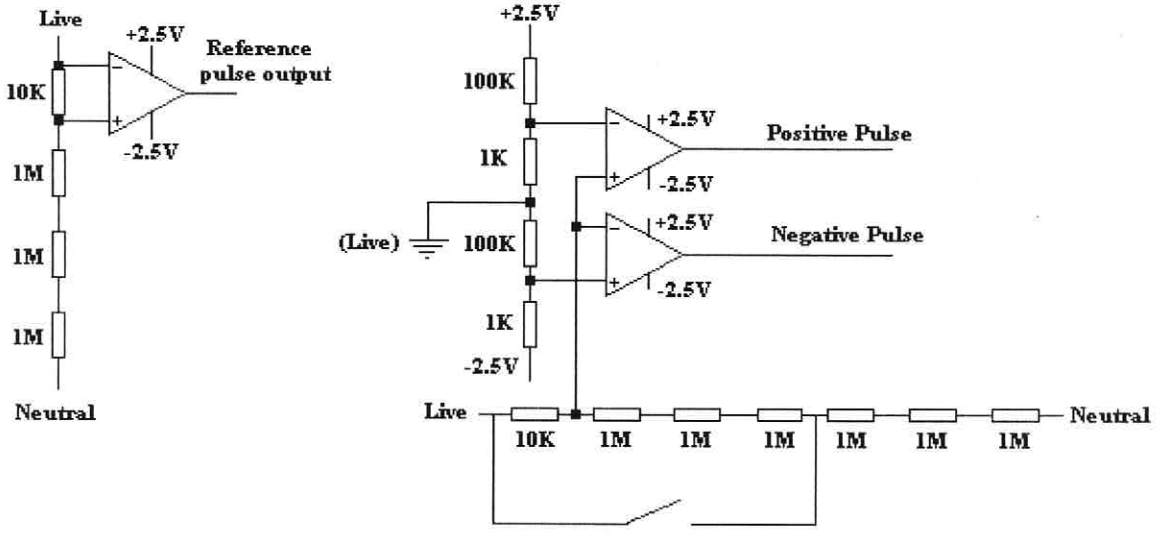
= 24.39 uA (25uA nominal)

The following calculations have been done without taking tolerance into account as the manufacturing calibration procedure will compensate for this.

Single Phase Bi-directional Power / Energy Metering IC with Instantaneous Pulse Output SA2002H, Sames Semiconductor, [Online] Available www.sames.co.za

2.3.3. Zero Cross Sense

The Zero cross sensing circuitry is created using three op amps used as comparators to generate a positive, negative and reference zero cross pulse. The reference pulse is used to check the moment of zero crossing (with 1% tolerance allowed.) , and the positive and negative zero crossing pulses are used to check at which point in the waveform a switch of the contactor has happened.



When the contactor which is in parallel with the resistive divider on the input of the positive and negative sense is closed, then the input of both the op amps are forced to ground (LIVE) and no pulses are present. If however the contactor is open, then the op amps will measure the voltage drop across to 10K resistor (in the circuit diagram of Annexure A this is labeled as R51). Because they are set up as comparators (approaching infinite gain) the op amps will swing their outputs between +2.5V and -2.5V as soon as the voltage becomes higher or lower than 2.5V/101 (due to the other inputs of the op amps which are connected to resistor dividers between of 1K and 100K resistors). The contactor used has a switching time of between 9ms and 16 ms, but the repeatability of this switching time is fairly good.

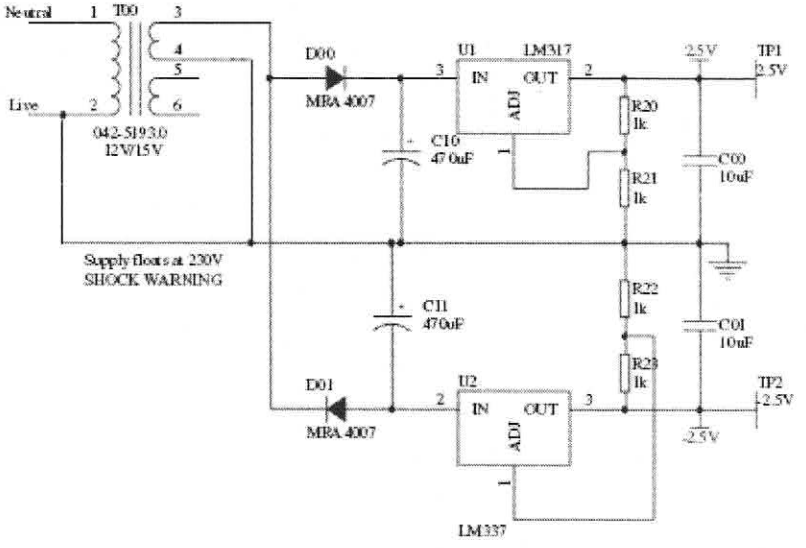
Every time a pulse is received on the reference zero crossing pulse, the software checks if a negative or positive pulse has been received in the last cycle.(the reference pulse only works on one of the half cycles, so if the negative and positive detectors are running, you can always be sure of receiving at least one pulse.). If there is a pulse present, we know that the contactor is open, if there are no pulses present, we know the contactor is closed. This is used to check if the contactor is in the state we expect it to be in after we have switched, so if a contactor goes faulty in the field, or a contactor unit gets bridged out externally our firmware can notify the server of an error. When the contactor is switching into the closed position from a previously open position, the following logic is used:

The switch trigger is sent to switch the contactor on the next rising edge of the zero crossing pulse train. The positive and negative pulses are constantly monitored to check for pulses that are not the same length as the reference pulses (with 2% tolerance). Because these pulses correspond to the shape of the waveform after the contactor, as soon as the contactor switches half way through a waveform, a half pulse will be received on either the negative or positive pulse trains. When this happens, the time taken to switch (t_{switch}) can be measured as the time between the sending the switch trigger, to the point where the half pulse ends. A calibration value can now be generated, which corresponds to the

difference in length between the half pulse and one of the normal reference pulses. If the software now waits for the length of this calibration as soon as the reference pulse starts to switch the contactor, we can be assured that the contactors mechanical contacts will close on exactly the zero point. The same logic can be followed for opening the contactor, except when opening, no pulses will be received in the close position, and as soon as the contactor switching into the open position, a half pulse will be the first pulse received, as the contactor will be opening halfway through a cycle, and the length of this pulse must now be calibrated to be as short as possible. This auto calibration procedure will be run at any time that a pulse is received of incorrect length during a switching procedure, so if we switch over to the open or closed position, and the software does not receive a half length pulse, the software just ensures that the contactor is in the correct position, and if it is not, it will notify the server.

2.3.4. Floating +2.5V and -2.5V Power Supply

Since the metering chip works by direct measurement of the current through a shunt which is floating at live, the whole power supply of the metering chip needs to float around live. The reference design of the SA2002H contains the design of a floating RC transformerless power supply, however, this power supply does not supply sufficient current to supply the measurement chip, op amps and opto isolators. For this reason a different design was decided on that incorporates a transformer (as an isolated supply was already needed to supply the communication lines for the ring bus and debug ports.) The design decided on uses an LM317 and LM337 which are connected to a common ground point which is in turn connected to Live. The rectifying circuit is a simple half wave single diode rectifier to each of the regulators as the currents drawn are small enough that the efficiency is not too much of a problem.



The LM337 and LM317 were used instead of fixed regulators, as they are cheaper than their fixed versions when ordered in quantity. The regulated voltage output is approximately determined by[23]:

Vreg = Vref*((R1+R2)/(R1))

Thus : Vreg = 1.25*((1K + 1K)/1K)

= 2.5V

where R1 is resistor on the ground end of the feedback resistive divider (in the design in Annexure A R20 and R22) and R2 is the resistor at the output end of the regulator (in the design in Annexure A R21 and R23). The reference voltage is assumed as 1.25V.

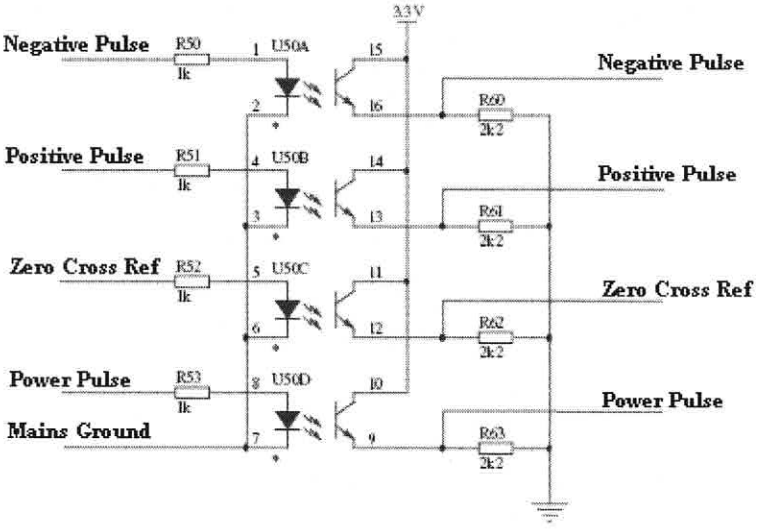
---

2 3 - Terminal Positive Adjustable Regulator LM317, Fairchild Semiconductor, [Online] Available www.fairchildsemi.com/ds/LM/LM317.pdf

3 3 - Terminal Negative Adjustable Regulator LM337, Fairchild Semiconductor, [Online] Available www.fairchildsemi.com/ds/LM/LM337.pdf

2.3.5. Opto Isolation

In order to keep the communication lines on the ring bus and debug port of the device isolated from the mains voltage, a separate power supply is used for the communication hardware and processor, and all the input signal lines from the measurement and zero crossing circuits are optically isolated. The signal lines that are at mains potential are the zero crossing pulses and power measurement pulses. None of these lines are running at a particularly high frequency so no special design requirements beyond those of electrical isolation are required in the optical isolation paths.



The optcoupler used is the CNY74-4. The typical current transfer characteristic of this opto coupler is specified as typically 100% so the $I_{out} = I_{in} = I_{io}$ [4]. The forward diode voltage is taken at 1.6V (from typical characteristics). The current is then set at:

$$I_{io} = (V_{app} - V_d)/R$$
$$= (5 - 1.6) / 1K$$
$$= 3.4mA$$

this output current is sufficient to ensure that almost all the voltage on the output side is across the output resistor since, So there is no significant voltage drop across the Collector emmiter junction of the output transistor, and therefor power dissipation is negligible.:

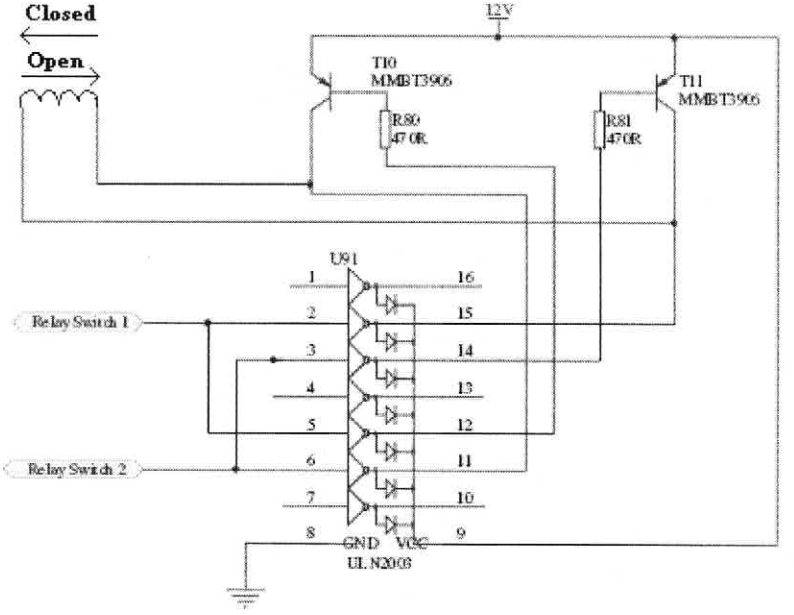
$$V_r = I_{io} R$$
$$= 3.4mA * 2K2$$
$$= 7.48V$$

$$P_t = V_{ce} * I_{i/o}$$
$$= 0.1V * 3.4mA$$
$$= 0.34mW$$

4 Multichannel Optocoupler With Phototransistor Output, Temic Semiconductor, [Online] Available http://www.alldatasheet.com/datasheet-pdf/pdf/81944/TEMIC/CNY74-4.html

2.3.6. Contactor Driver

Since the contactor drive coil needs to be driven differentially (Current in one direction = Open/ Current in the opposite direction = Closed), it is driven by a standard H-Bridge drive circuit composed of PNP MBT3906 Transistors and a ULN NPN Darlington Driver Chip[5]. The contactor coil takes about 120mA at 12V (1.44W) to switch over to either direction, so the bridge is specified to drive a maximum current of 200mA (maximum continuous drive current of the MBT3906[6]).



Power dissipation across the is MMBT3906 is approx.:

$P_{ce} = V_{sat} * (I_c + I_b)$

$= 0.4 * (120mA + 18.7mA)$

$= 55.48mW$

$I_b = (12V - 0.7V - 2.5V) / R$

$= 8.8V / 470$

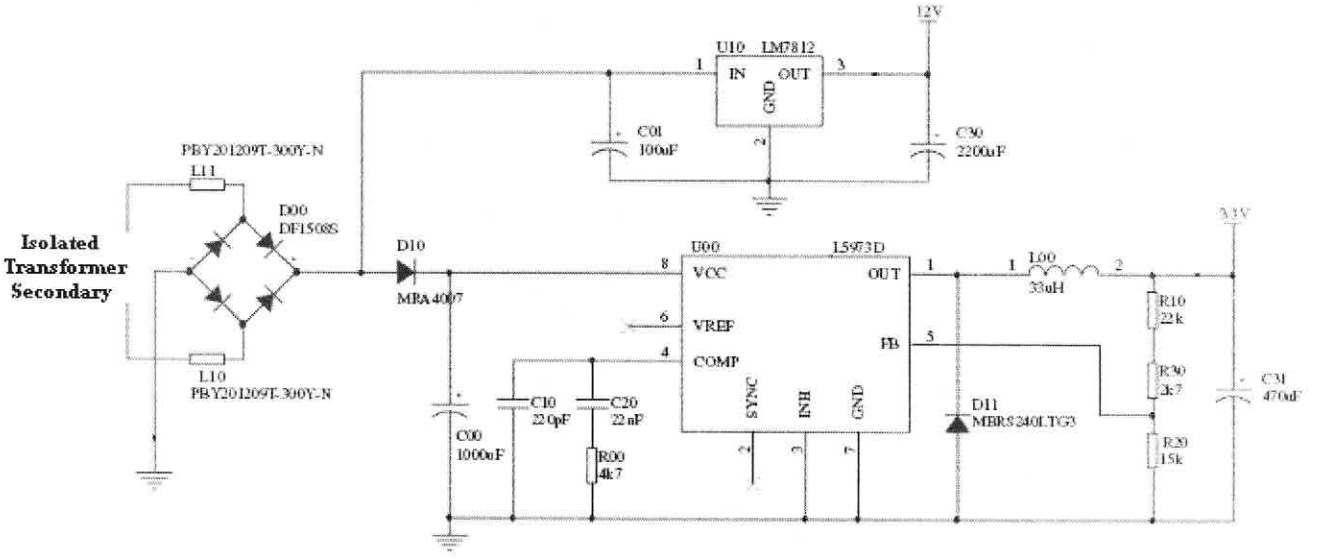
$= 18.7mA$

5 Seven Darlington Arrays, ST Semiconductor, [Online] Available www.st.com/stonline/products/literature/ds/5279.pdf

6 PNP General Purpose Amplifier, Fairchild Semiconductor, [Online] Available www.fairchildsemi.com/ds/2N/2N3906.pdf

2.3.7. Power Supply (+3.3V and +12V)

After the optocouplers, the microcontroller, comms, contactor driver, flash and eeprom are all powered by the isolated 3.3V and 12V power supplies. The power supplies are isolated as they are on a separate secondary on the transformer to the power supplies that float at 230V. The 3.3V supply uses an L5973 switch mode power supply chip and the 12V supply uses an LM7812 linear fixed supply. The input lines to the rectifier are filtered using two PBY210 ferrites (0805). After the rectifier, the two supplies are split using D10 to stop the 12V supply drawing too much current from the input buffer capacitor to the 3.3V supply. This way, if the 12V supply tries to take too much power due to a failure on the coil, it will just drop the voltage, without killing the 3.3V supply to the microcontroller.



The feedback network to the switch mode power supply is calculated as follows:

V_{out} = ((R20)/(R20+R30+R10)) * Vref

= (15K)/(15K + 2K7 + 22K) * 1.235V

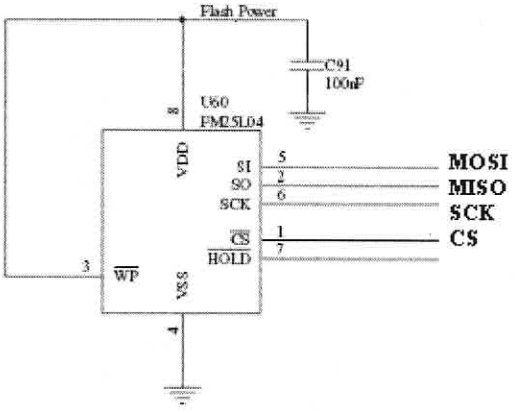
= 3.1V

The coil and diode configurations are from the datasheet graphs[7].

7 2.5A switch step down switching regulator, ST Semiconductor, [Online] Available www.alldatasheet.net/datasheet-pdf/pdf/22521/STMICROELECTRONICS/L5973.html

2.3.8. EEPROM storage

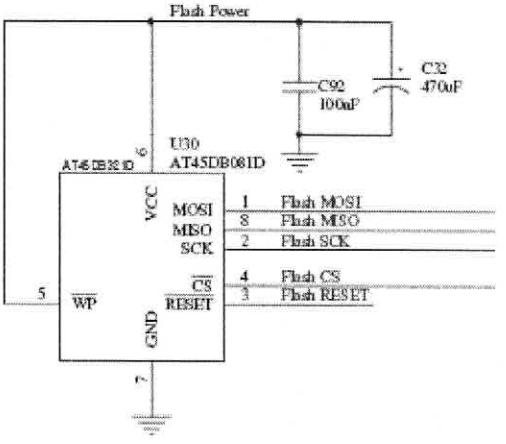
The EEPROM chip is used to store the Kwh register count of the measured pulses. The EEPROM is powered at 3.3V through a diode, so that when the power to the system is cut, the 1F super capacitor doesn't power everything, it only powers the microprocessor. The EEPROM used is a Ramtron FM25L04 [8]which uses the same interface as the standard 25LC series eeproms from microchip semiconductor, except the ramtron chips have a much longer life cycle, which was needed if the device is to have a life expectancy of 10 years.



8 PNP General Purpose Amplifier, Fairchild Semiconductor, [Online] Available www.fairchildsemi.com/ds/2N/2N3906.pdf

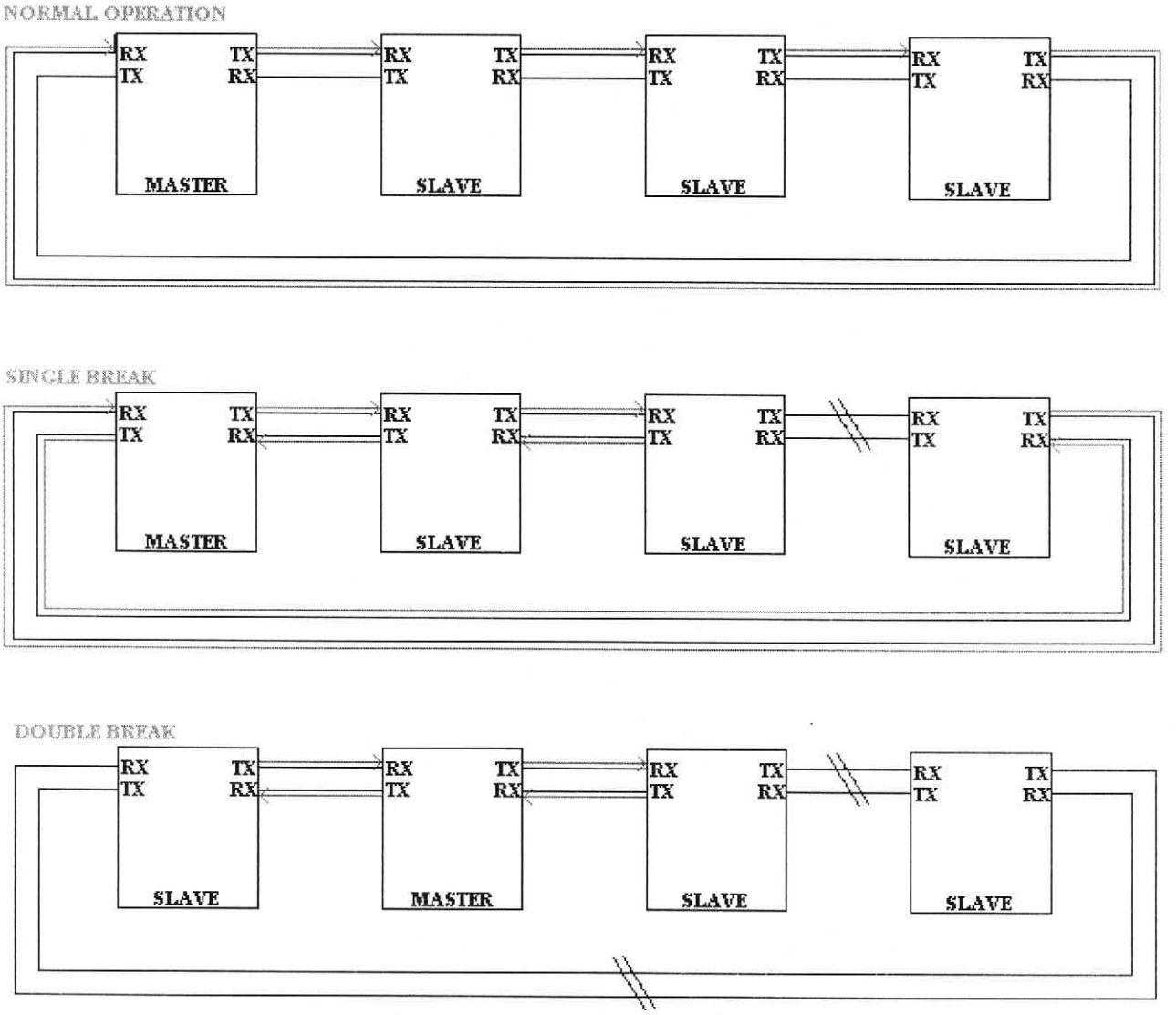
2.3.9. Flash Storage

The Flash storage is used to store the encrypted keys and forms the basis of the transactional file system. There is a small reservoir capacitor that feeds only the flash chip (it is isolated from the rest of the supply by a diode), that ensures that any operation currently in progress can finish when power to the unit is cut for whatever reason. The At45DB08 chip has 8 Mega bit of storage (1MByte), and is broken up into a number of pages. Files in the file system are into a number of pages depending on the size, but never smaller than one page so that the page erase functionality of the chip can be used. The file system component of the EOS operating system is transactional in nature, which means that data can be written into a file all in one go, and then read out again, but as soon as the data needs to be updated, a copy of the current file needs to be made and written into a shadow file which is then made active.



2.3.10. Ring Bus communication.

The ring bus forms the communication link between the remote master unit and the contactor unit. The ring bus uses two sets of rs232 ports to allow bi directional communication around the bus in failure modes. The ring bus can thus be in one of the following modes:



During normal operation, the data goes along the bus following the first uart. The master initiates most of the messages, but on certain occasions the slaves can also initiate messages. A message is considered complete when it has traveled around the bus once and has come around to the originator of the message. When a break in the line is detected (the message ACK times out), the bus goes into fail mode A, which allows it to bypass this unit and loop back on the device next to it to maintain the integrity of the bus. The break in the line is also identified and a failure message is transmitted up to the server via the remote master. When more than one break is detected, the bus may need to short itself out with a second loop, which removes the devices outside of the break from the bus, but maintains the integrity of the ring bus.

2.3.11. Display Leds

The RJ45 connectors on the ring bus also incorporate two leds for display purposes (Standard ethernet connectors.) for a total of 4 display leds. The leds display the following data, and is the only way to see the status of the device when it is installed. The leds represent the following functionalities in the device

- Led 1: Ring Bus Side A Traffic
- Led 2: Ring Bus Side B Traffic
- Led 3: Contactor Switch status
- Led 4: Wh pulse

3.1. Mechanical Design

The contactor device is installed in the Stubby electrical distribution points, and due to the large amount of variance between different stubby units throughout the whole country, the mounting options for the contactor device need to reflect this. The mechanical shell design has thus been proposed to incorporate a Din Mounting, mini Din mounting as well asn a screw down mounting. The shell is being designed by mr. Jan Olwagen. The Pcb layout needs to fit inside the given mechanical parameters.

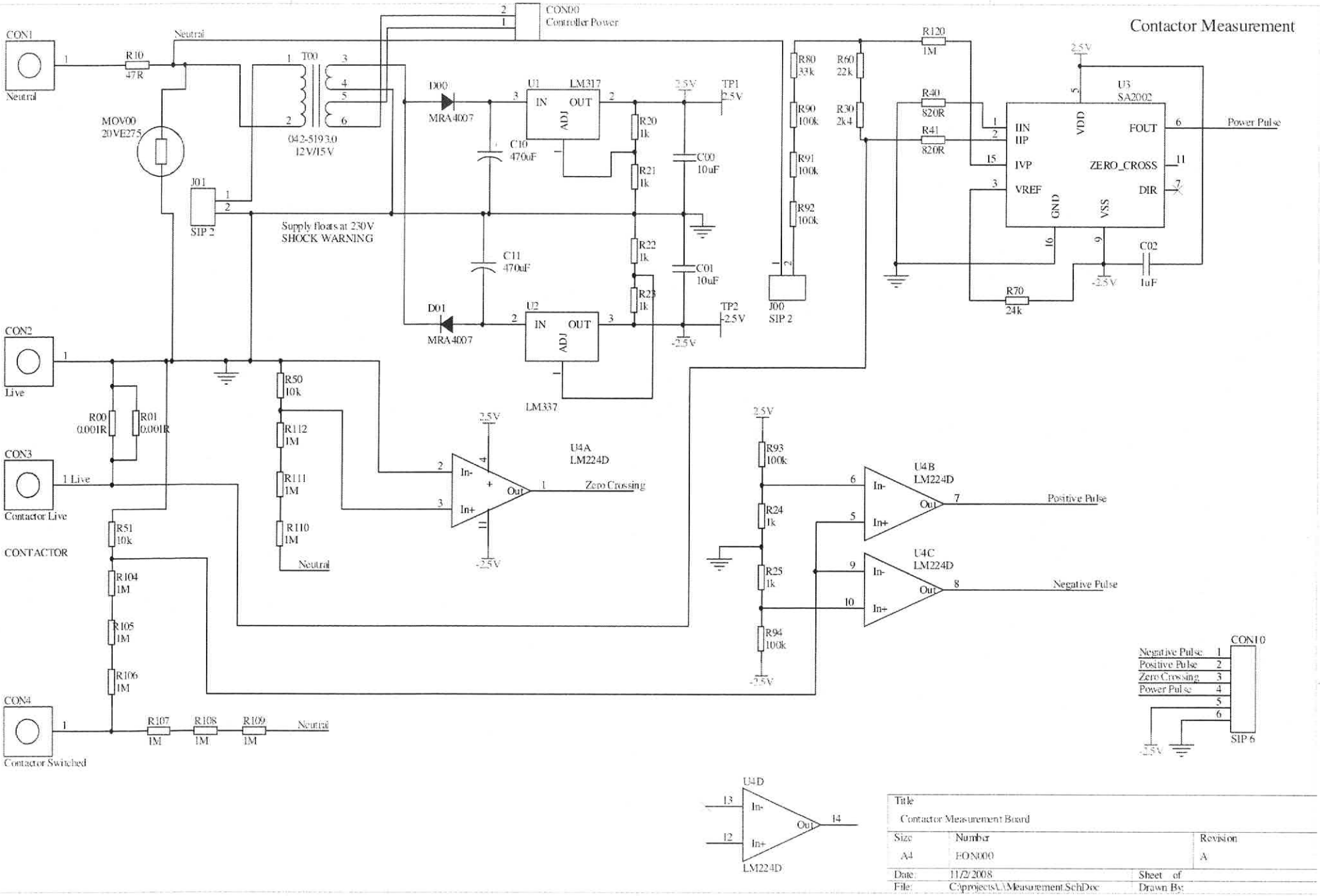
4.1. Design Information – Schematics and Pcb Documents

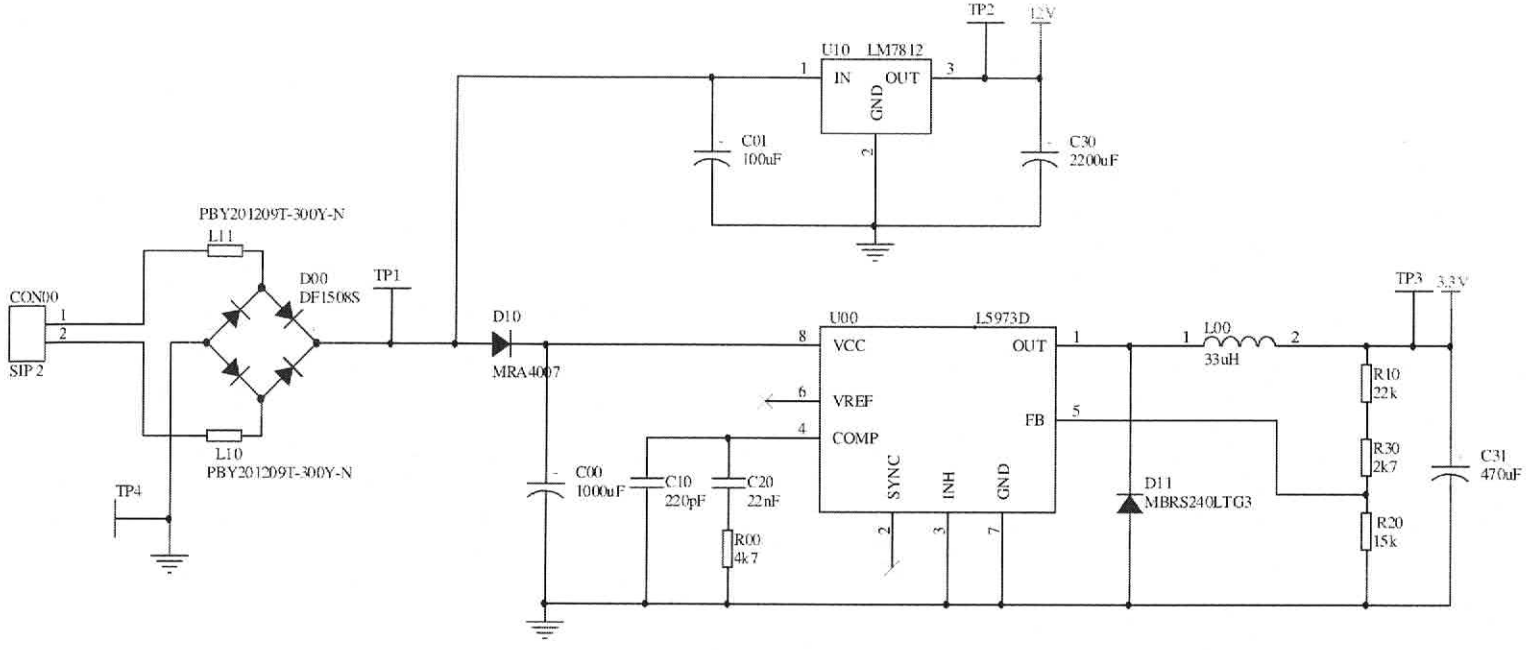
The device has been split into two pc boards to allow it to fit into the mechanical shell easily. The boards are designated as EON000 and EON001. The schematics are on the following pages in the following order:

- 4.2.1 EON000 Measurement Schematic
- 4.2.2 EON001 Power Supply Schematic
- 4.2.3 EON001 IO Schematic
- 4.2.4 EON001 Processor Schematic

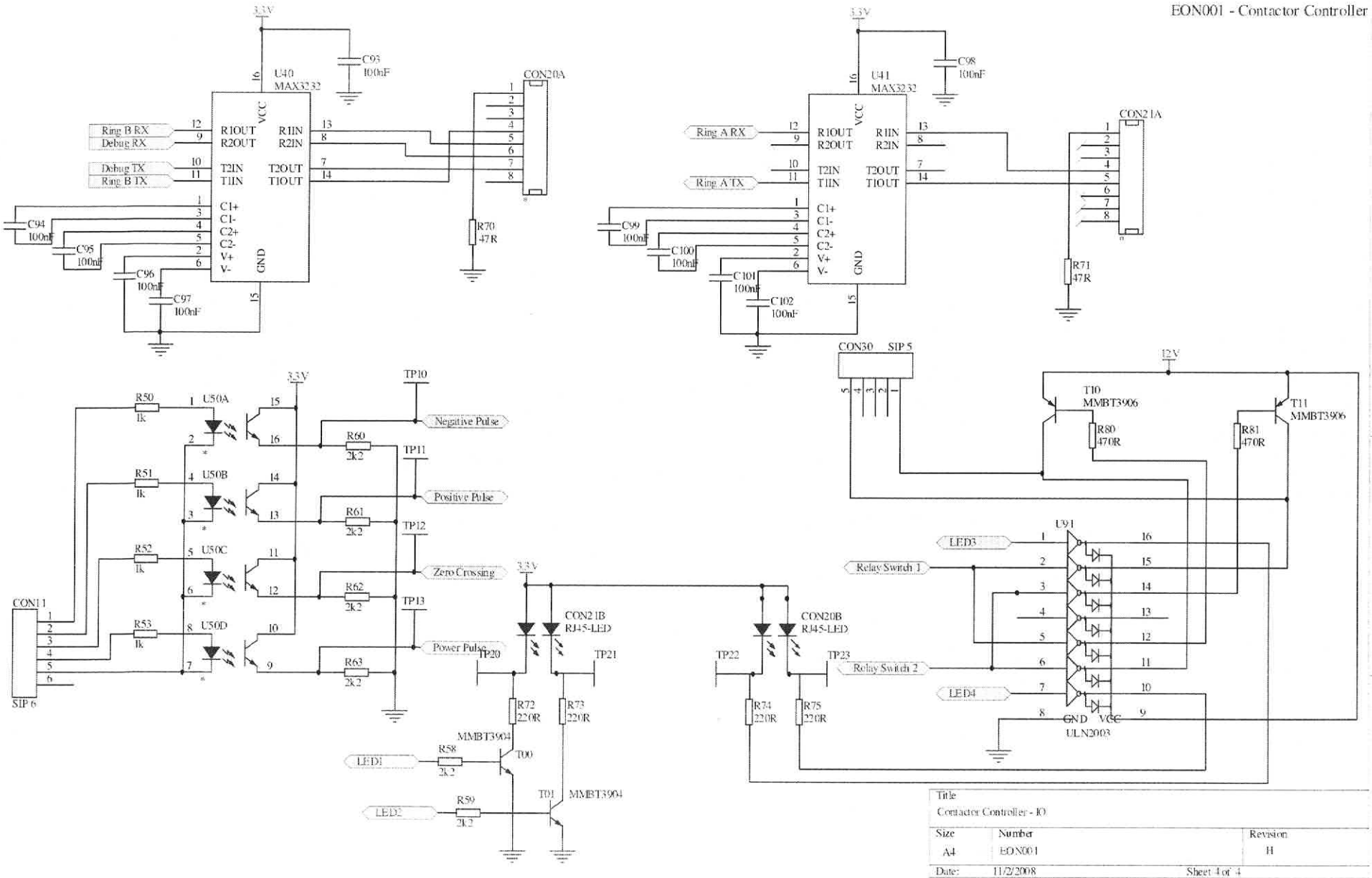
The pcb documents have also been broken up in the following way:

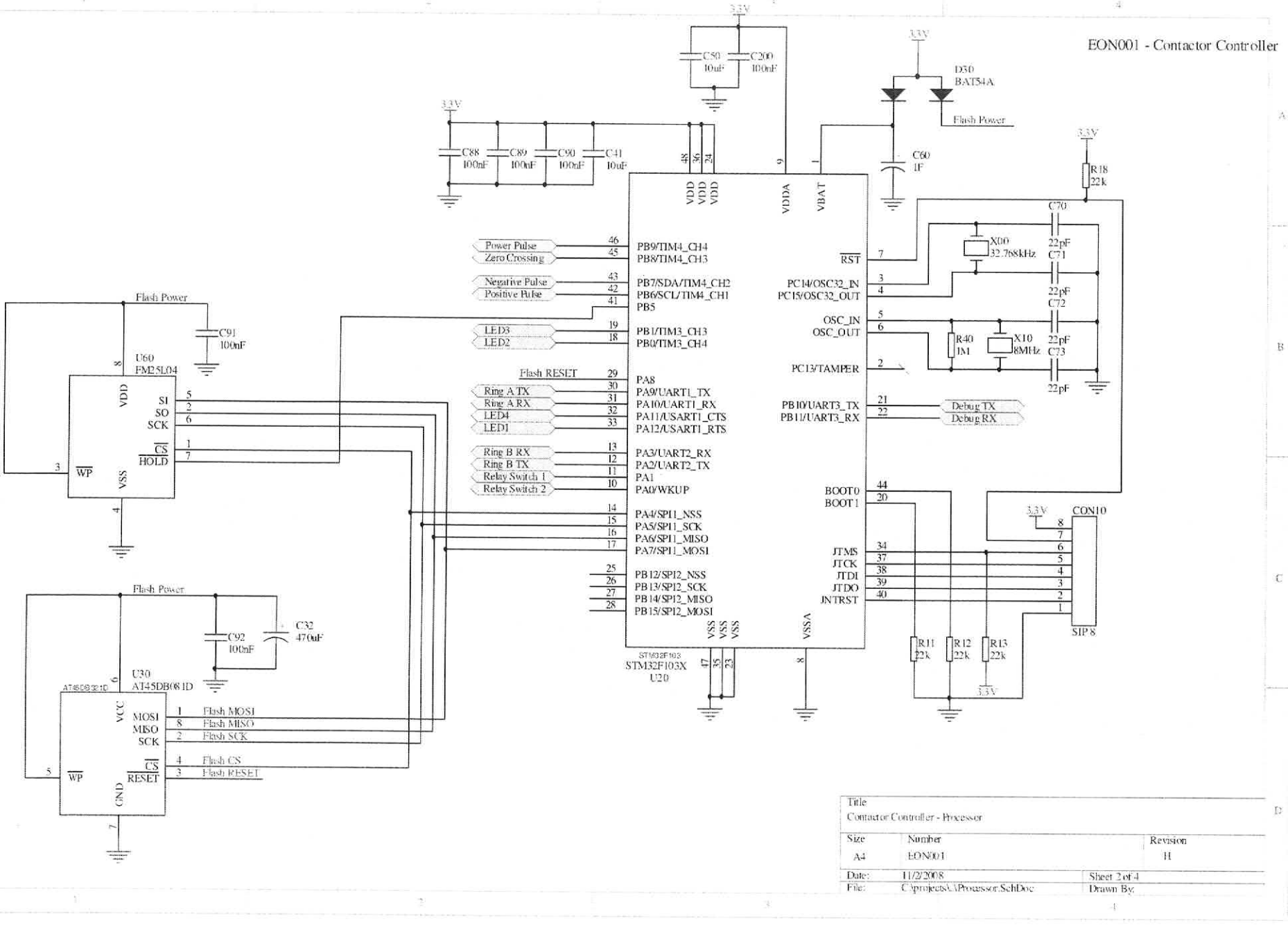
- 4.3.1 EON000 Top Layer
- 4.3.2 EON000 Top Overlay
- 4.3.3 EON000 Bottom Layer
- 4.3.4 EON000 Bottom Overlay
- 4.3.5 EON001 Top Layer
- 4.3.6 EON001 Top Overlay
- 4.3.7 EON001 Bottom Layer
- 4.3.8 EON001 Bottom Overlay

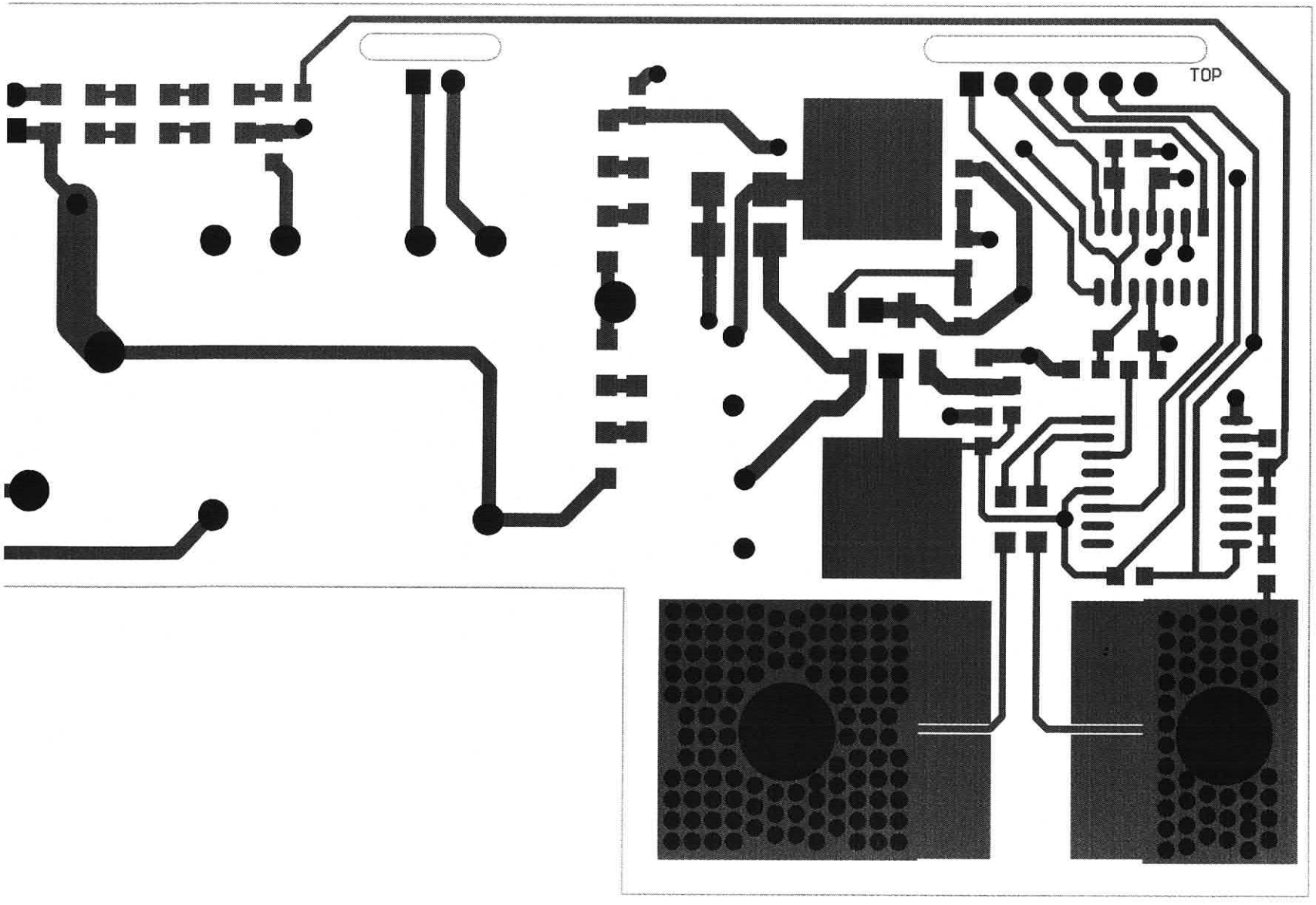

Contactor Measurement
CON00
Controller Power
CON1
Neutral
R10
47R
MOV00
20VE275
T00
042-5193.0
12V/15V
D00
MRA4007
U1
LM317
IN
OUT
ADJ
C10
470uF
R20
1k
R21
1k
C00
10uF
TP1
2.5V
J01
SIP 2
Supply floats at 230V
SHOCK WARNING
C11
470uF
R22
1k
R23
1k
C01
10uF
D01
MRA4007
U2
LM337
TP2
-2.5V
J00
SIP 2
R80
33k
R90
100k
R91
100k
R92
100k
R60
22k
R30
2k4
R120
1M
R40
820R
R41
820R
U3
SA2002
IIN
IIP
IVP
VREF
VDD
VSS
GND
FOUT
ZERO_CROSS
DIR
Power Pulse
C02
1uF
R70
24k
CON2
Live
R00
0.001R
R01
0.001R
CON3
Contactor Live
CONTACTOR
R50
10k
R112
1M
R111
1M
R110
1M
U4A
LM224D
Zero Crossing
R51
10k
R104
1M
R105
1M
R106
1M
CON4
Contactor Switched
R107
1M
R108
1M
R109
1M
R93
100k
R24
1k
R25
1k
R94
100k
U4B
LM224D
Positive Pulse
U4C
LM224D
Negative Pulse
CON10
Negative Pulse
Positive Pulse
Zero Crossing
Power Pulse
SIP 6
U4D
LM224D
Title
Contactor Measurement Board
Size
A4
Number
EON000
Revision
A
Date:
11/2/2008
Sheet of
File:
C:\projects\...\Measurement.SchDoc
Drawn By:



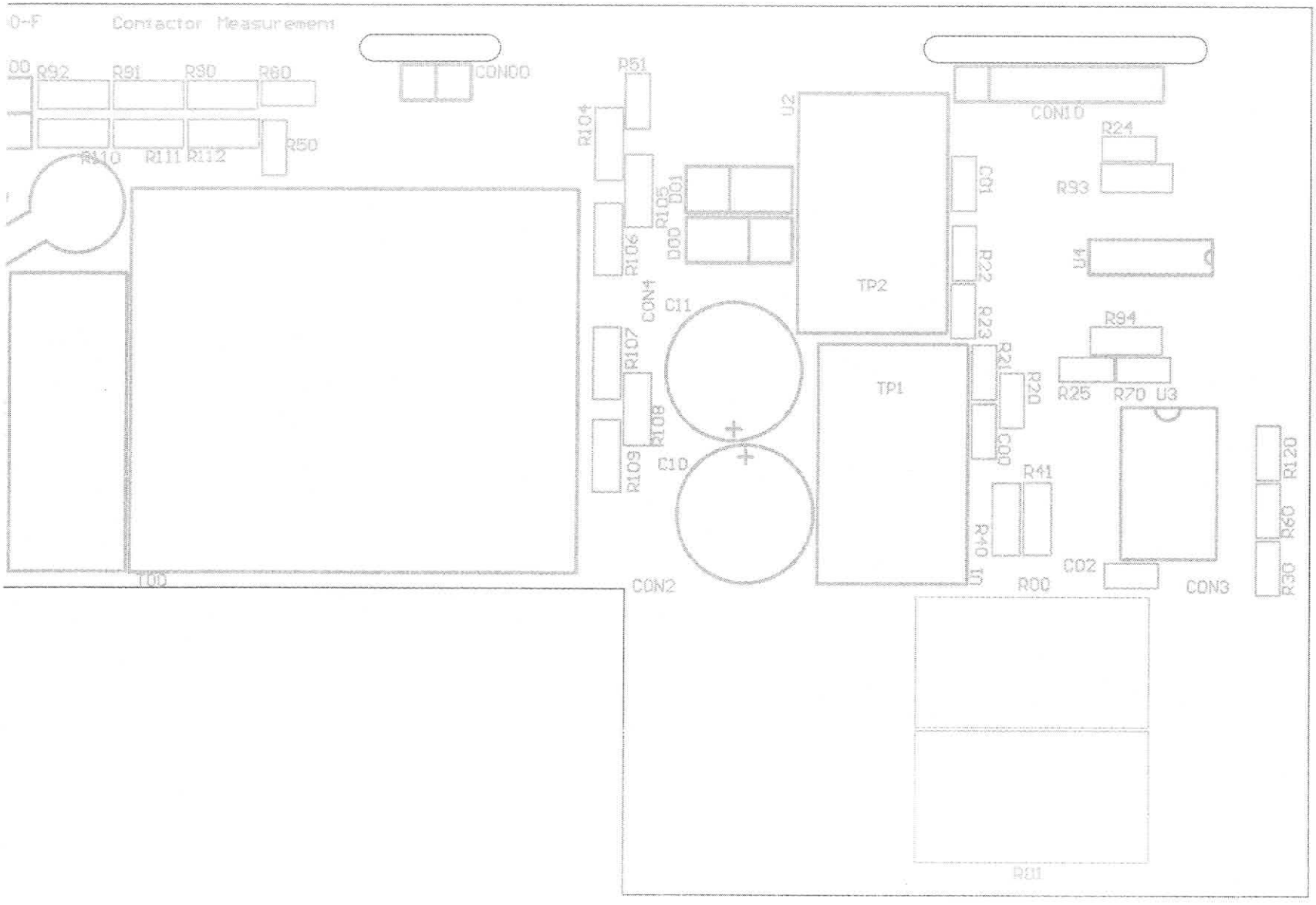
Title: Contactor Controller - Power		
Size: A4	Number: EON001	Revision: H
Date: 9/6/2008	Sheet 3 of 4	
File: C:\projects\...\Power.SchDoc	Drawn By:	


EON001 - Contactor Controller
U40
MAX3232
U41
MAX3232
Ring B RX
Debug RX
Debug TX
Ring B TX
Ring A RX
Ring A TX
R1OUT
R2OUT
R1IN
R2IN
T2IN
T1IN
T2OUT
T1OUT
VCC
GND
C1+
C1-
C2+
C2-
V+
V-
C93 100nF
C94 100nF
C95 100nF
C96 100nF
C97 100nF
C98 100nF
C99 100nF
C100 100nF
C101 100nF
C102 100nF
CON20A
CON21A
R70 47R
R71 47R
CON30 SIP 5
CON11
SIP 6
U50A
U50B
U50C
U50D
R50 1k
R51 1k
R52 1k
R53 1k
R60 2K2
R61 2K2
R62 2K2
R63 2K2
TP10
TP11
TP12
TP13
Negative Pulse
Positive Pulse
Zero Crossing
Power Pulse
CON21B
RJ45-LED
CON20B
RJ45-LED
TP20
TP21
TP22
TP23
R72 220R
R73 220R
R74 220R
R75 220R
MMBT3904
T00
T01
R58 2K2
R59 2K2
LED1
LED2
T10
MMBT3906
T11
MMBT3906
R80 470R
R81 470R
U91
ULN2003
LED3
LED4
Relay Switch 1
Relay Switch 2
GND
VCC
Title
Contactor Controller - IO
Size
A4
Number
EON001
Revision
H
Date:
11/2/2008
Sheet 4 of 4
File:
C:\projects\Mio.SchDoc
Drawn By:

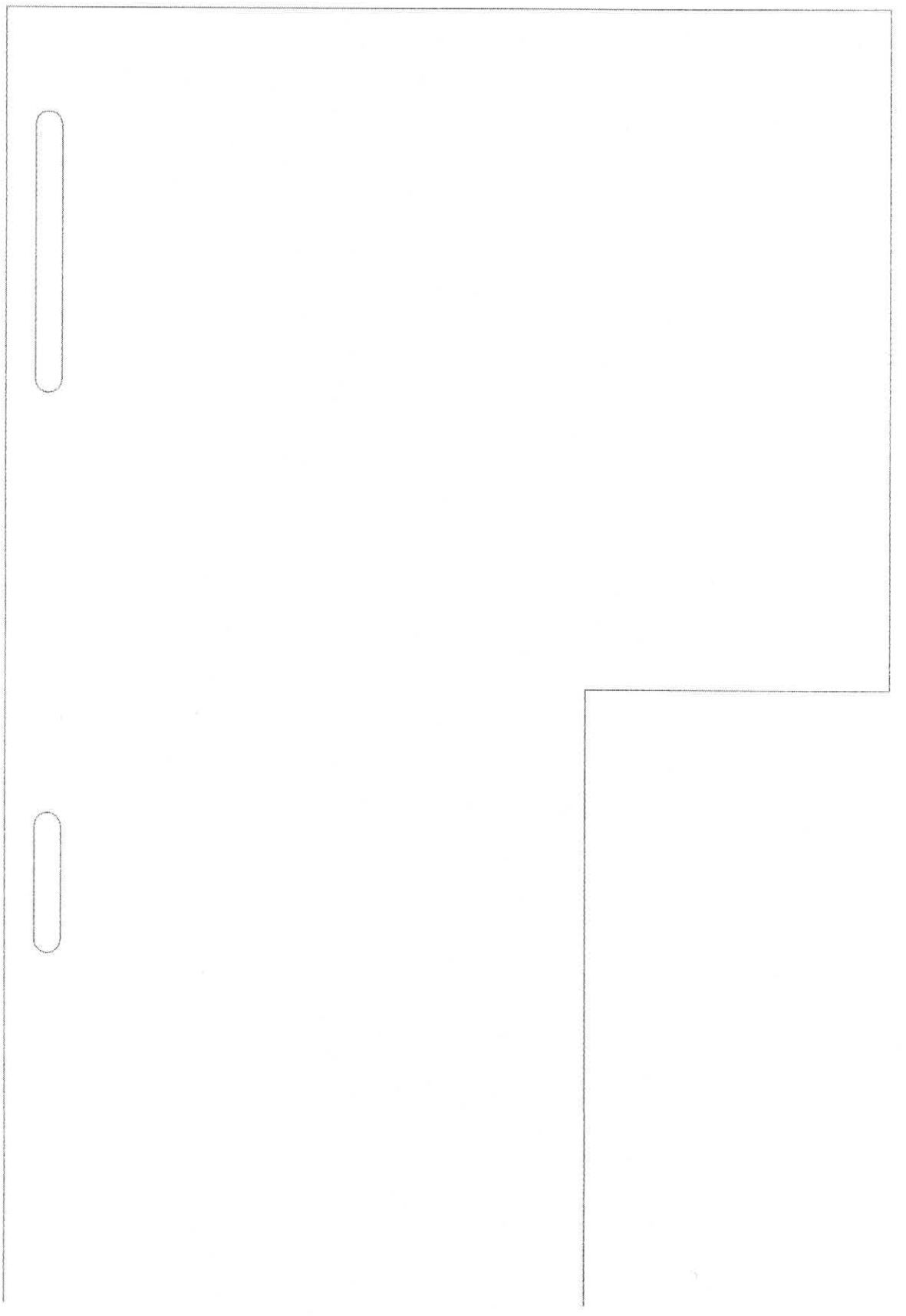

EON001 - Contactor Controller
3.3V
C50 10uF
C200 100nF
C88 100nF
C89 100nF
C90 100nF
C41 10uF
D30 BAT54A
Flash Power
C60 1F
R18 22k
C70 22pF
X00 32.768kHz
C71 22pF
C72 22pF
R40 1M
X10 8MHz
C73 22pF
Power Pulse
Zero Crossing
Negative Pulse
Positive Pulse
LED3
LED2
Flash RESET
Ring A TX
Ring A RX
LED4
LED1
Ring B RX
Ring B TX
Relay Switch 1
Relay Switch 2
PB9/TIM4_CH4
PB8/TIM4_CH3
PB7/SDA/TIM4_CH2
PB6/SCL/TIM4_CH1
PB5
PB1/TIM3_CH3
PB0/TIM3_CH4
PA8
PA9/UART1_TX
PA10/UART1_RX
PA11/USART1_CTS
PA12/USART1_RTS
PA3/UART2_RX
PA2/UART2_TX
PA1
PA0/WKUP
PA4/SPI1_NSS
PA5/SPI1_SCK
PA6/SPI1_MISO
PA7/SPI1_MOSI
PB12/SPI2_NSS
PB13/SPI2_SCK
PB14/SPI2_MISO
PB15/SPI2_MOSI
VDD
VDDA
VBAT
VSS
VSSA
RST
PC14/OSC32_IN
PC15/OSC32_OUT
OSC_IN
OSC_OUT
PC13/TAMPER
PB10/UART3_TX
PB11/UART3_RX
Debug TX
Debug RX
BOOT0
BOOT1
JTMS
JTCK
JTDI
JTDO
JNTRST
STM32F103
STM32F103X
U20
Flash Power
C91 100nF
U60
FM25L04
VDD
SI
SO
SCK
CS
HOLD
WP
VSS
C92 100nF
C32 47uF
U30
AT45DB081D
VCC
MOSI
MISO
SCK
CS
RESET
WP
GND
Flash MOSI
Flash MISO
Flash SCK
Flash CS
Flash RESET
R11 22k
R12 22k
R13 22k
CON10
SIP 8
Title
Contactor Controller - Processor
Size
A4
Number
EON001
Revision
H
Date:
11/2/2008
Sheet 2 of 4
File:
C:\projects\A\Processor.SchDoc
Drawn By:

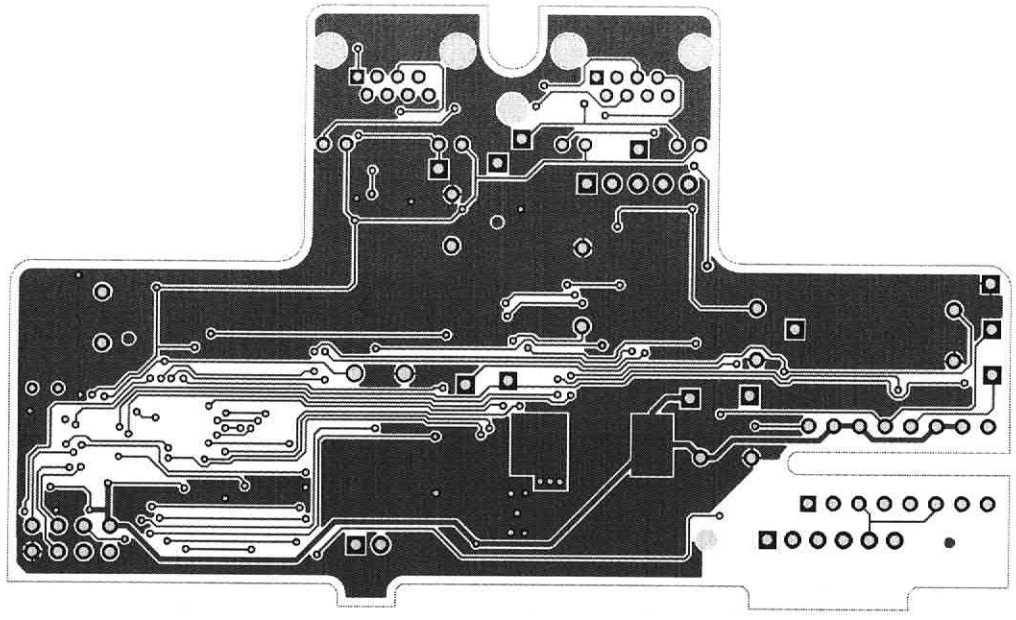


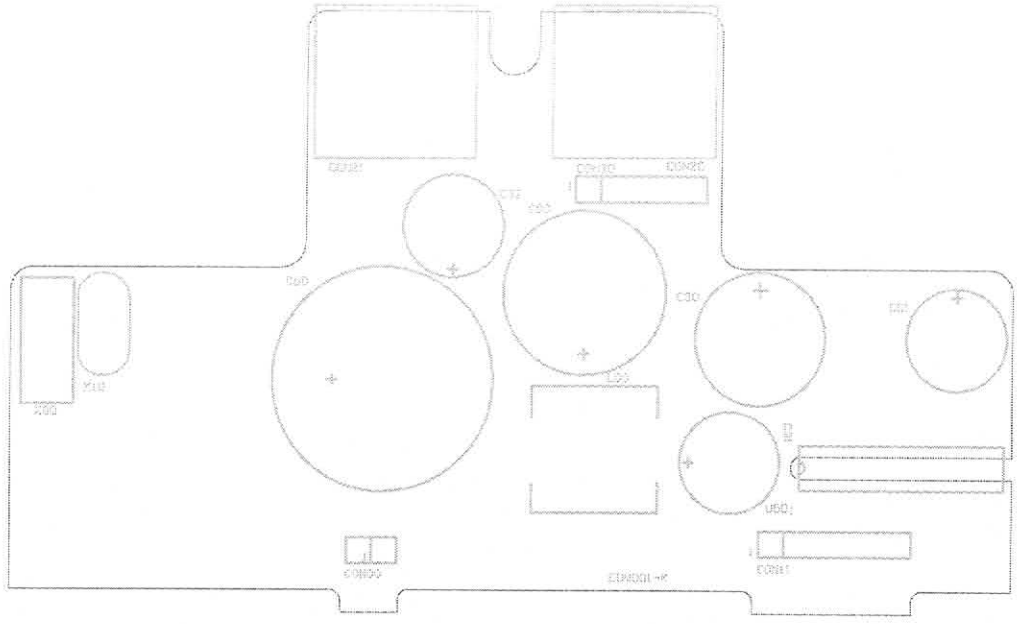
Top La

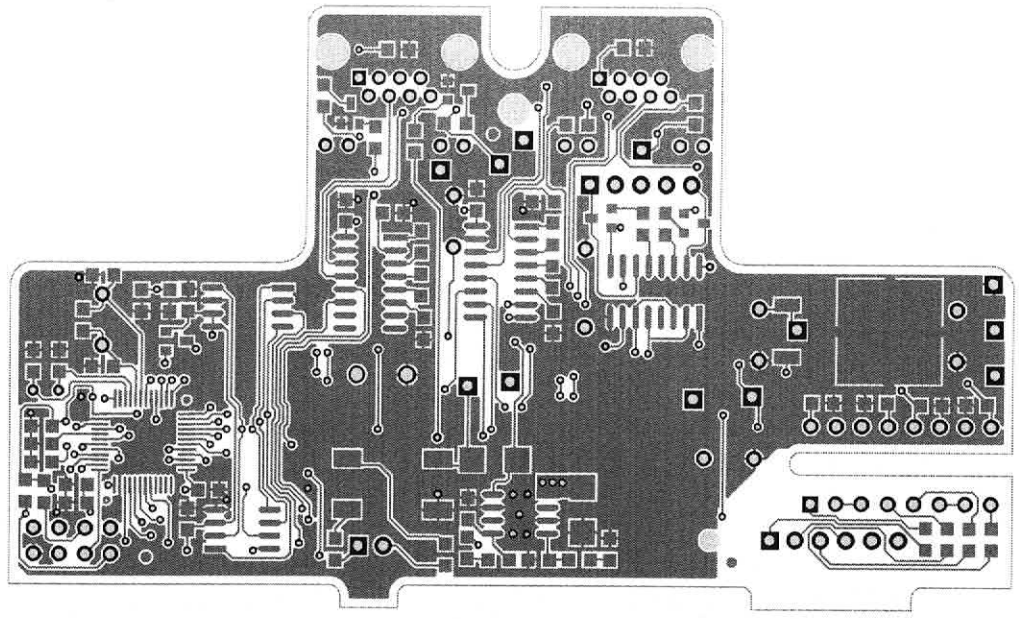

0-F
Contactor Measurement
R92
R91
R90
R60
R110
R111
R112
R50
CON00
R104
P51
R105
R106
D01
D00
CON4
C11
C10
R107
R108
R109
U2
TP2
TP1
CON1
C01
R24
R93
R22
R23
R21
R20
C00
R41
R40
U1
R00
U4
R94
R25
R70
U3
C02
CON3
R120
R60
R30
T00
CON2
R11

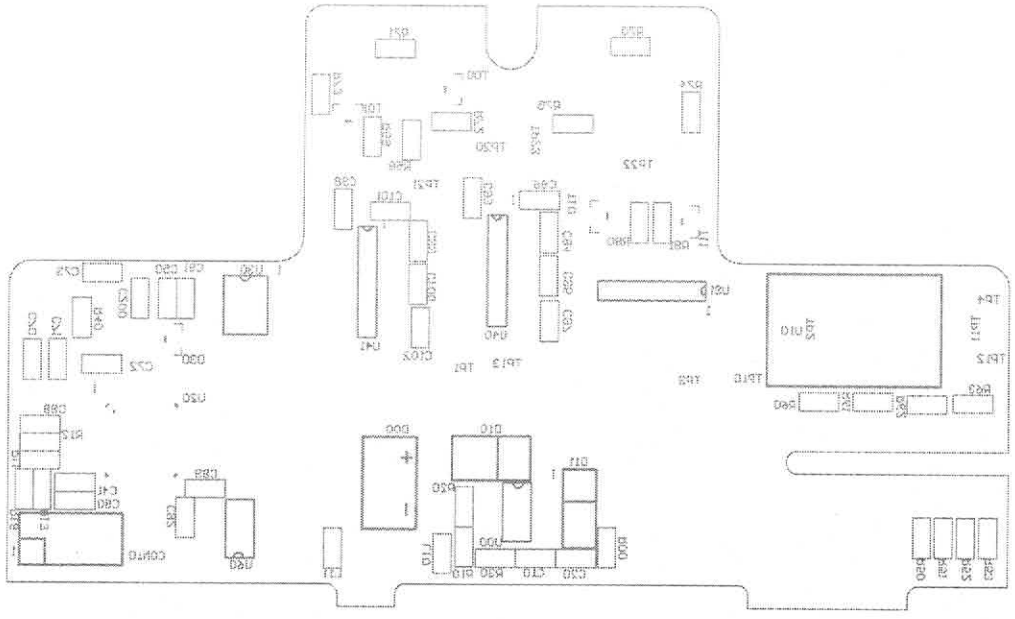
Bottom Layer











5. Contact List

- Jacob Greeff — Engineer on Load limiter project
  email: japieg@gmail.com — Tel : 078 214 6148
- Nicholas Prozesky — Engineer on Load limiter project and my mentor
  email: nicholasp@eon.co.za — Tel : 083 798 7408
- Andrew Goedhart — Development Leader
  email: andrewg@eon.co.za — Tel : 084 556 7711

6. Bibliography

1. Single Phase Bi-directional Power / Energy Metering IC with Instantaneous Pulse Output SA2002H, Sames Semiconductor, [Online] Available www.sames.co.za
2. LM117/LM317A/LM317 3-Terminal Adjustable Regulator, National Semiconductor, [Online] Available www.national.com/ds/LM/LM117.pdf
3. LM124/LM224/LM324/LM2902 Low Power Quad Operational Amplifiers, National Semiconductor, [Online] Available www.national.com/ds/LM/LM124.pdf
4. LM137/LM337 3-Terminal Adjustable Negative Regulators, National Semiconductor, [Online] Available_http://www.national.com/ds.cgi/LM/LM137.pdf
5. ULN2003 High Voltage and high current darlington transistor array, YouWang Electronics corporation, [Online] Available http://www.datasheetcatalog.org/datasheets/120/489337_DS.pdf
6. FM25L04 4Kb FRAM Serial 3V Memory, Ramtron Electronics, [Online] Available www.zlgmcu.com/RAMTRON/pdf/FM25L04.pdf
7. AT45DB – 1-Megabit 2.7-Volt Only Serial DataFlash, Atmel Corporation, [Online] Available http://www.alldatasheet.co.kr/datasheet-pdf/pdf_kor/56158/ATMEL/AT45DB.html
8. CNY74-4 – Multichannel OptoCoupler with phototransistor output, Temic Semiconductors, [Online] Available

   http://www.alldatasheet.com/datasheet-pdf/pdf/81944/TEMIC/CNY74-4.html
9. STM32F103x4 Low-density performance line, ARM-based 32-bit MCU with 16 or 32KB Flashm USB, CAN, 6 timers, 2 ADCs, 6 communication interfaces, ST microelectronics, [Online] Available http://www.st.com/stonline/products/literature/ds/15060/stm32f103c4.pdf
10. K. Billings, Switchmode power supply handbook. New York: McGraw-Hill, 1989, pp.1.89-1.100
11. SANS62051, "1.00 Electricity metering – Glossary of terms," SABS, 1999
12. SANS10142, "The wiring of premises, Part1: Low-voltage installations", SABS, 2003
13. E. Hughes, Electrical Technology. Edinburgh Gate: Addison Wesley Longman Limited, 1998

DPJ391UE

Design Implementation Report As Well As Final Conclusions

By
Jacob Jacobus Greeff (3692-247-1)

In partial fullfulment of the requirements for the Ndip: Electrical Engineering

Table of Contents

1. Update....3
2. Wiring diagrams of test set up....4
3. Ring bus Load test....7
4. Final Conclusions....9
5. Contact List....11
6. Bibliography and References....12

1. Update

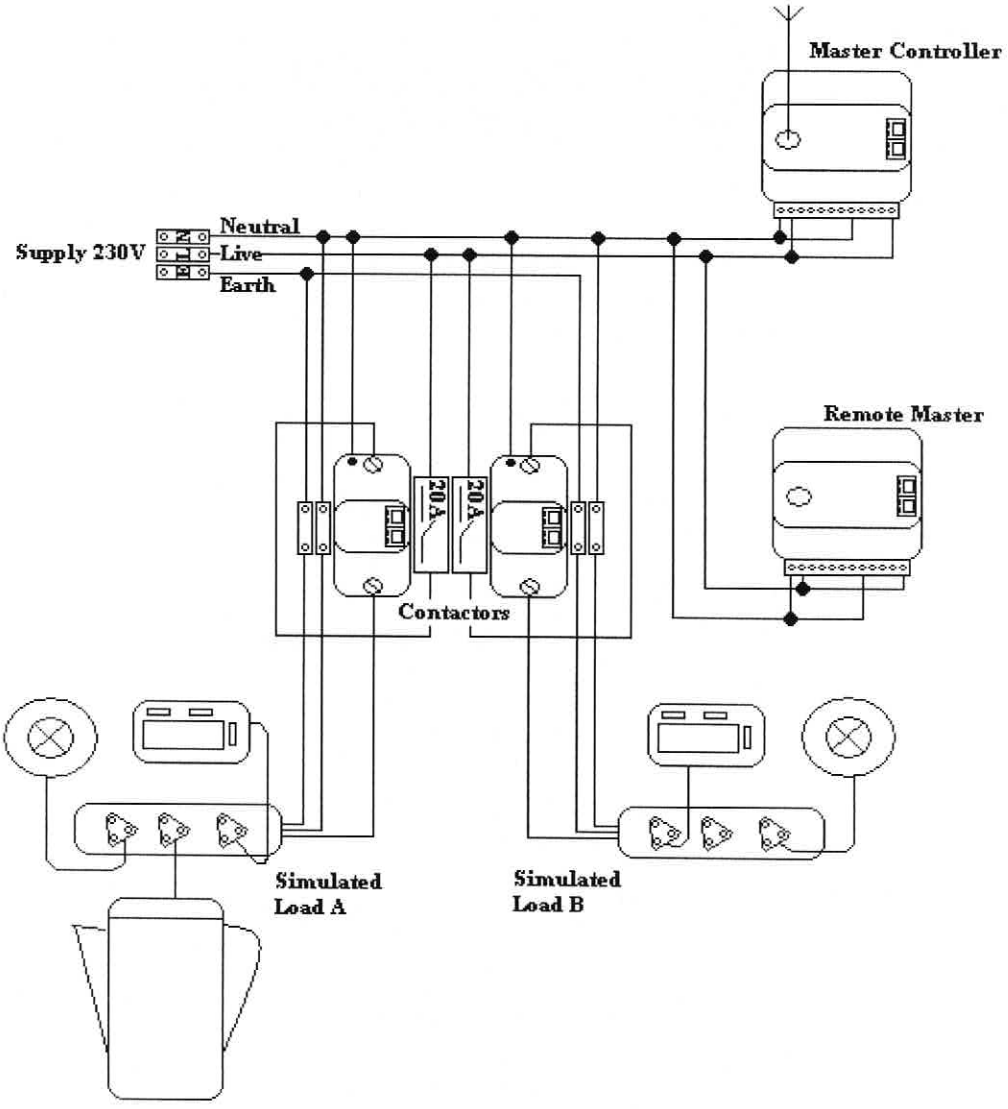
After a number of revisions, a final hardware revision for each of the components in the Load limiter project has been achieved. On the 30th of October 2008 the first dry run of the end to end communication capabilities of the system was implemented as a part of a presentation to delegates of each of the municipalities in south africa. The prototype system was a simulation of the functionalities that the final system can implement once installed into a residential stubby network. The functional tests were run as follows:

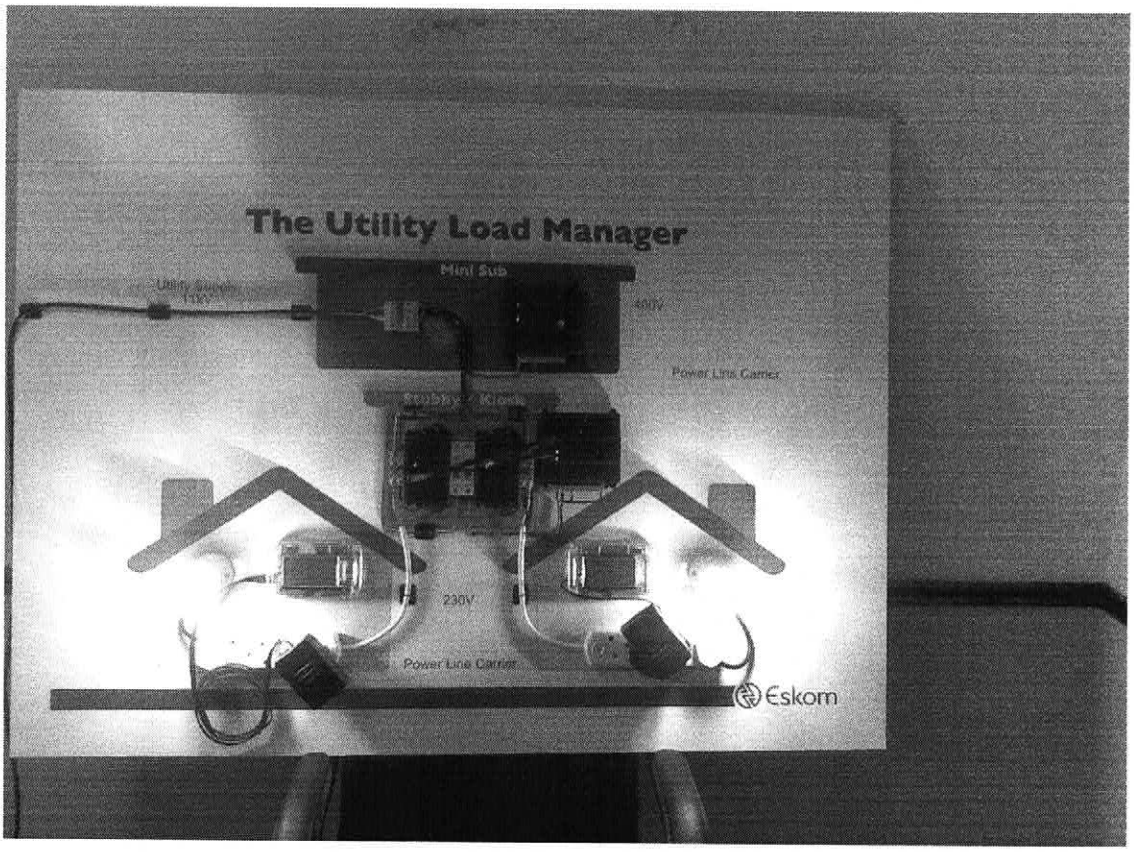
- A connection was made between the Load limiter server and the master controller unit, using a gprs connection. The server sent down load limiting schedules, timing information and messages to the master controller, and the master controller was able to properly collate collected data and send it through to the server which harvested the data and displayed it in the form of graphs and reports.
- The Master Controller unit succesfully built the Plc Mesh network from scratch and propogated the required data to the remote master units, and collected data from them in turn.
- The remote master units succesfully implemented the ring bus network with the two contactor units, that measured the power consumption of the simulated house loads.
- The display terminals successfully captured all the required data and the initial implementation of the menu system and information windows went down as a huge success with the delegates that attended the meeting.

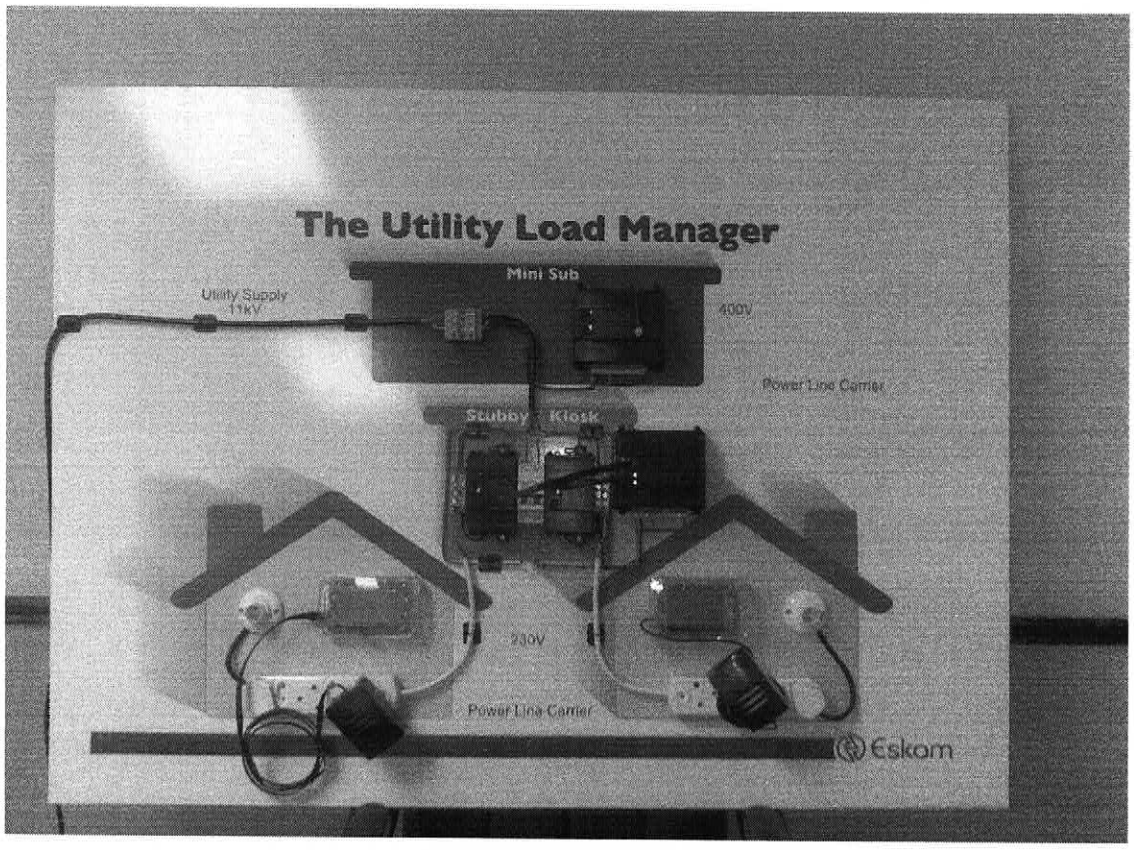
The test involved running the system under normal conditions, and then introducing a 1500W load limit on the house simulations. When the load limit is in place, the house simulators have a base load of 1 60W light bulb which simulates a compliant state in which the utility will guarantee never to cut the user. A larger load (2000W kettle) is then introduced onto one of the simulated load points to sinulate a non compliant state, which then indicates via the display that the user needs to comply or he will be cut off in a set amount of time (This is displayed as a countdown on the Lcd.). Once the user has chosen not to comply (or his load has exceeded the limit set by the utility more than 3 times in the set load limiting period) the user will be cut. The demonstraition that load shedding could be implemented on a customer to customer basis, with full access to information within two minutes of anywhere in the system was greeted with a huge amount of inthusiasm by the delegates that attended. The next test to be run will be implemented into a resedential complex with 40 houses, controlled by 1 master controller and a number of remote masters on the 24th of november 2008.

2. Wiring diagrams of test set up.

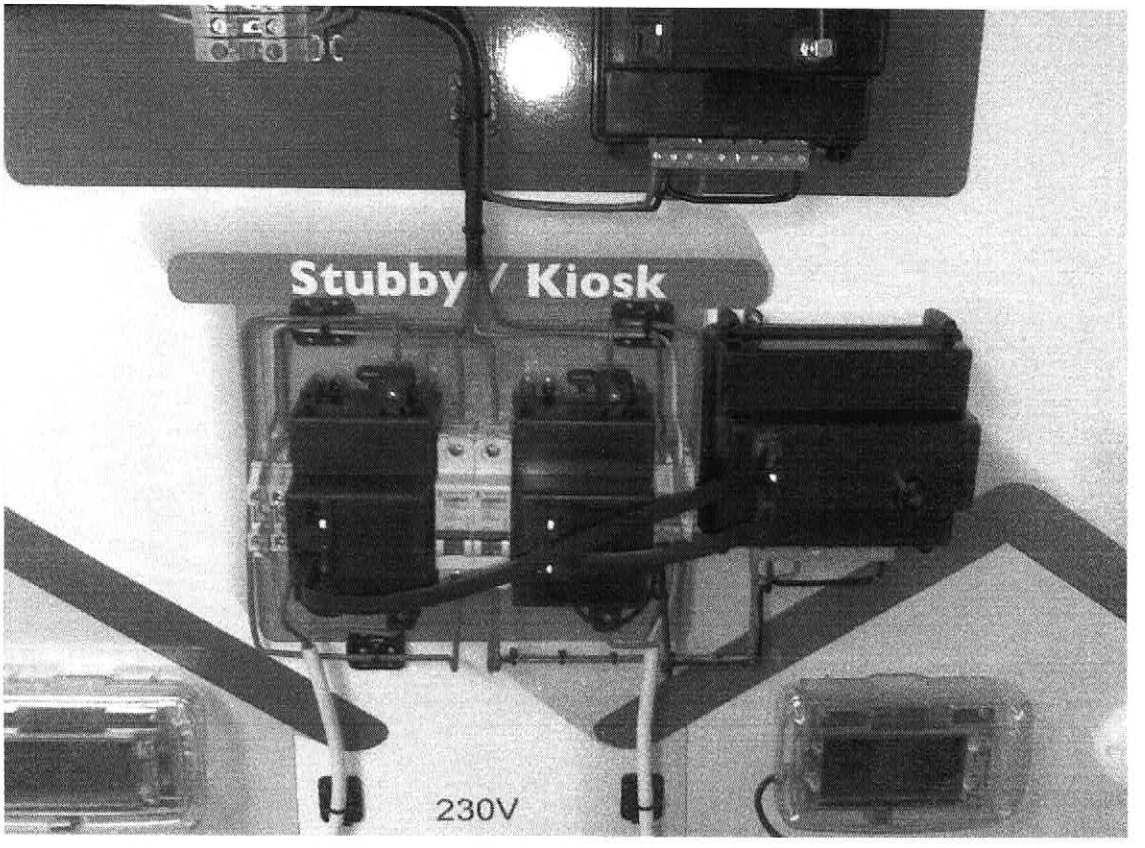
What follows below is the wiring diagram of the test set up, which simulates the way in which the system will be installed into the proposed sites, as well as photos of the test set up while running.




The Utility Load Manager
Mini Sub
Utility Supply
11kV
400V
Power Line Carrier
Stubby Kiosk
230V
Power Line Carrier
Eskom


The Utility Load Manager
Mini Sub
Utility Supply
11kV
400V
Power Line Carrier
Stubby Kiosk
230V
Power Line Carrier
Eskom

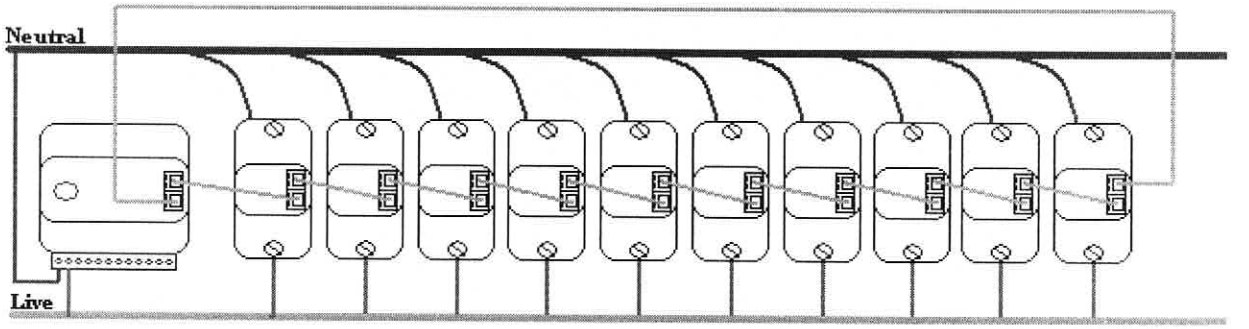

he Utility Load Manager
Mini Sub
400V
Stubby / Kiosk


Stubby / Kiosk
230V

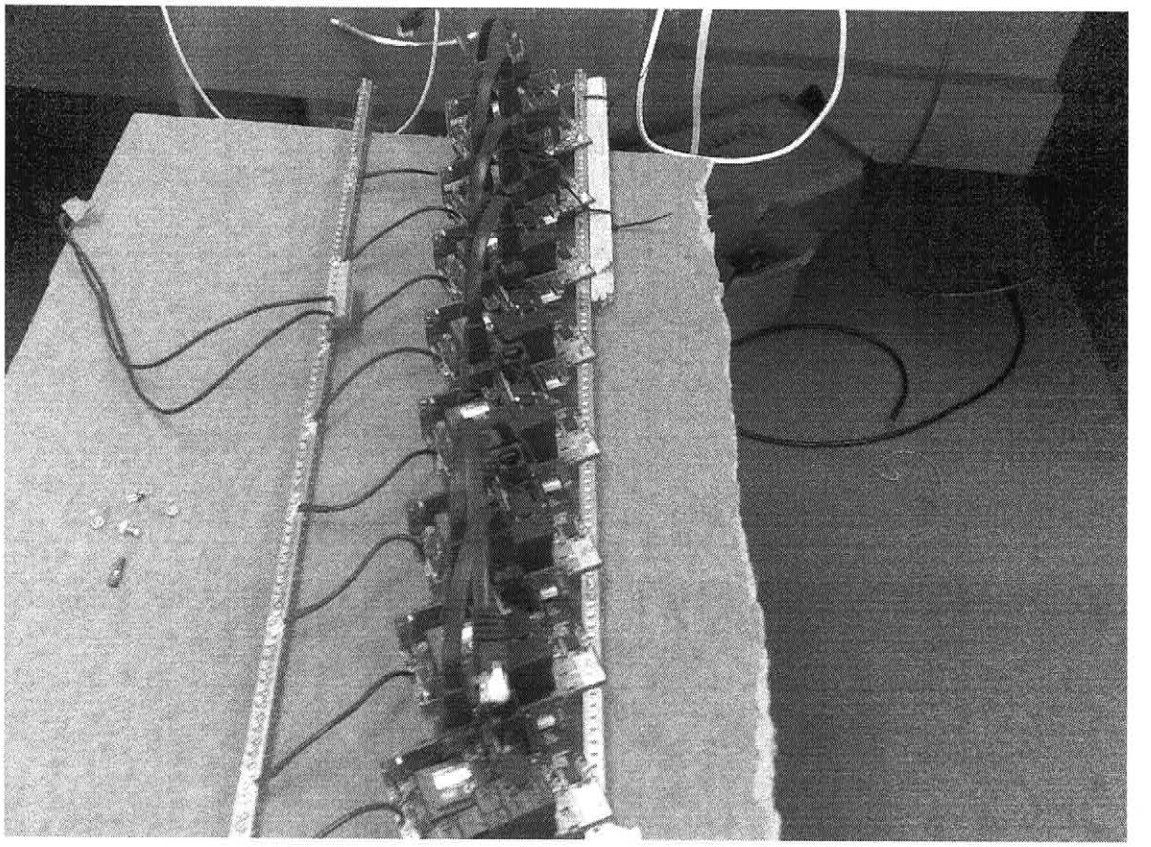
3. Ring bus Load test

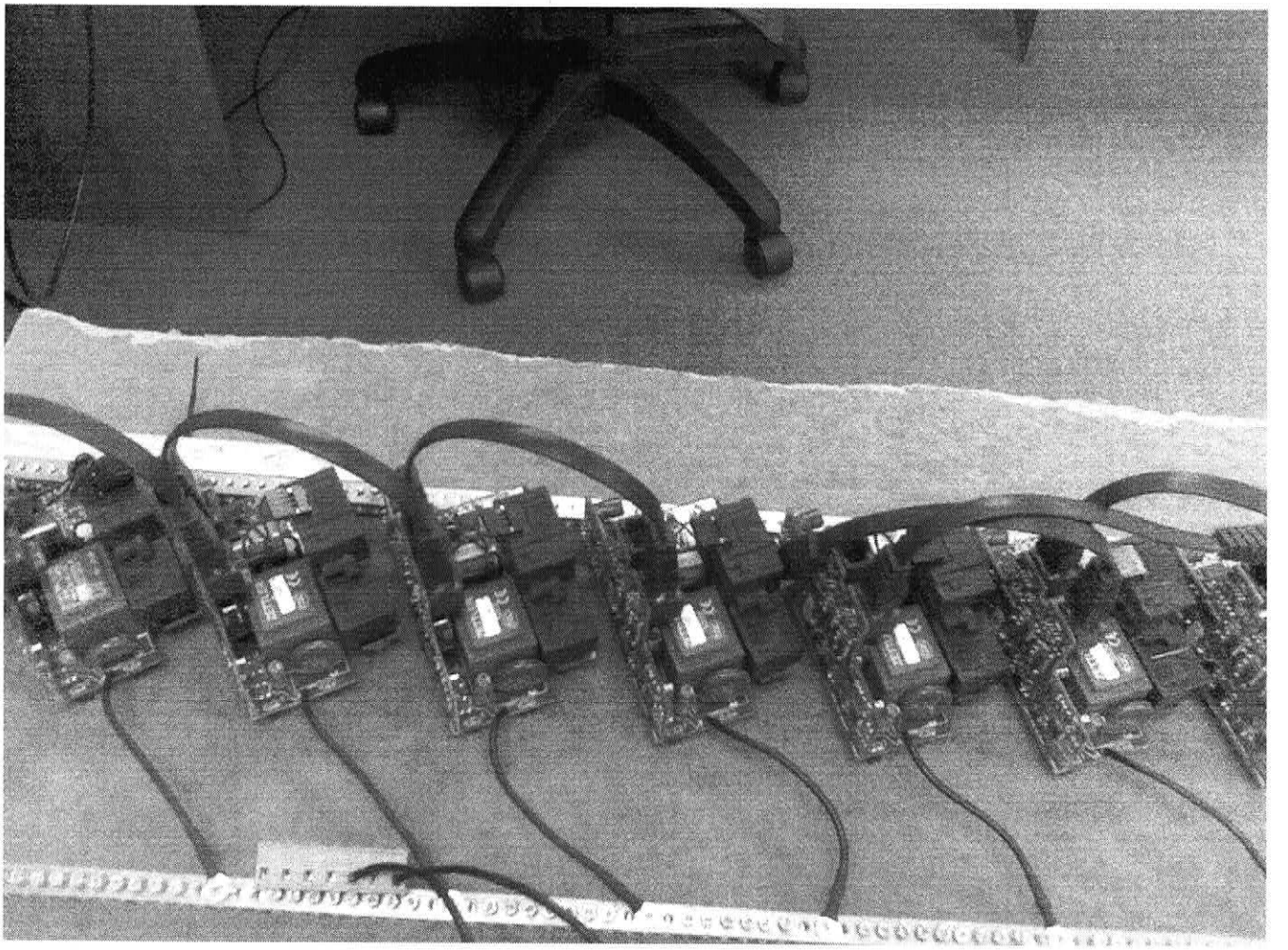
The ring bus can have up to 32 devices on it, and to insure that no messages are lost on the bus, and that minimal collisions occur, a load test was performed using a remote master and 10 contactors. The ring bus propogates a number of test messages around the bus, followed by a "Contactor switch message" which will either switch all of the contactors to open or closed (alternating between the two states.). Below is a wiring diagram of the ring bus test setup, as well as a photo of the set up in action.

Wiring Diagram



Photos





4. Final Conclusions

After the end to end test of the system, as well as load tests run on the server and Plc network, it is my conclusion that the proposed system performs well with reference to the initial spec that was put down by eskom and the proposal committee. The Proposed pilot of first, 40 houses as pilot site, and then the full system prototype consisting of 27000 connection points in march 2009 attests to the rigidity and performance of the system as a whole. A number of changes have however been proposed with reference to the suitability of the system. These are outlined below:

- A second source for the secondary Plc network needs to be found. After much discussion, it has been decided that the Plc network currently implemented using the Yitran It800 module, although perfectly sufficient according to the specification, causes a weakness in the long term maintainability of the project due to the propriatry nature of the module. The following courses of action have been taken :
    - Second sources of Plc modules have been identified from Texas instruments as well as St Microelectronics
    - In the case that the Plc network proves unsuitable during the rollout of the pilot system, a wi-fi and Rf solution has been identified as a suitable (if slightly more expensive) secondary network.
- A second source for the measurement chip needs to be found. Although the Sames chip meets all the required specifications, in order to keep the long term prospects of the system sustainable, second sources of measurement chips have been identified from both St microelectronics as well as atmel. Although these devices were initially investigated at the start of the project, they proved too expensive due to the fact that they require a second transducer to measure power factor (The Sames chip measures power factor by calculating the difference in phase angle between current and voltage, where as the other two devices calculate phase angle by measuring the difference in the current signal in the live wire versus the current in the neutral wire.). Action taken is that we have ordered development kits of these two devices and will implement measurement pc boards to fit the current mechanical assembly as soon as they arrive and are proven suitable.
- The size of the contactor unit is proving difficult to install into a stubby type system. Although the size of the contactor device meets all the specifications initially put down, it has been found on site visits that the size of the contactor makes it difficult to install into the confined space of the stubby when more than 6 houses are connected to the same stubby. Using the current mechanical assembly, a second stubby will to be installed to connect houses that have too many connections in one stubby. To correct this, the size of the contactor will need to be substantially reduced, which will require a major redesign of the system. The following changes have been suggested:
    - Instead of floating the power supply to the metering chip and using opto couplers to feed the signal to the processor, use the same supply, all floating at live, and then connect the communication ports for the ring bus optically. This will save space on the power suply as the floating suppply can then be implemented without the use of a transformer.
    - source a thinner contactor that has the same rating, perhaps at a higher price.
    - Investigate using atmels new range of metering chips on release in the first quarter of

2009 that combine a measurement chip and microprocessor into one cpu. The board space that this saves should allow the entire circuit to fit onto one pcb if the capacitors are placed in such a way as to not interfere with the contactor mechanically.

- By making these changes, the contactor size should be able to shrink from the current 7cm to at least 4 cm which will make a sufficent difference to nullify the space constraint.

- Most of the problems encountered have been in the firmware of the unit, and as with most embedded projects, this is the longest running part of the project. To cut down on this development time, it has been proposed that a cruise control server be set up, that will implement automated testing of the unit tests every time firmware is committed to the central SVN archive. This should minimise the errors currently being seen while the operating system is still in development.

5. Contact List

- Jacob Greeff — Engineer on Load limiter project
  email: japieg@gmail.com — Tel : 078 214 6148
- Nicholas Prozesky — Engineer on Load limiter project and my mentor
  email: nicholasp@eon.co.za — Tel : 083 798 7408
- Andrew Goedhart — Development Leader
  email: andrewg@eon.co.za — Tel : 084 556 7711

6. Bibliography and References

1. Single Phase Bi-directional Power / Energy Metering IC with Instantaneous Pulse Output SA2002H, Sames Semiconductor, [Online] Available www.sames.co.za
2. LM117/LM317A/LM317 3-Terminal Adjustable Regulator, National Semiconductor, [Online] Available www.national.com/ds/LM/LM117.pdf
3. LM124/LM224/LM324/LM2902 Low Power Quad Operational Amplifiers, National Semiconductor, [Online] Available www.national.com/ds/LM/LM124.pdf
4. LM137/LM337 3-Terminal Adjustable Negative Regulators, National Semiconductor, [Online] Available_http://www.national.com/ds.cgi/LM/LM137.pdf
5. ULN2003 High Voltage and high current darlington transistor array, YouWang Electronics corporation, [Online] Available http://www.datasheetcatalog.org/datasheets/120/489337_DS.pdf
6. FM25L04 4Kb FRAM Serial 3V Memory, Ramtron Electronics, [Online] Available www.zlgmcu.com/RAMTRON/pdf/FM25L04.pdf
7. AT45DB – 1-Megabit 2.7-Volt Only Serial DataFlash, Atmel Corporation, [Online] Available http://www.alldatasheet.co.kr/datasheet-pdf/pdf_kor/56158/ATMEL/AT45DB.html
8. CNY74-4 – Multichannel OptoCoupler with phototransistor output, Temic Semiconductors, [Online] Available

   http://www.alldatasheet.com/datasheet-pdf/pdf/81944/TEMIC/CNY74-4.html
9. K. Billings, Switchmode power supply handbook. New York: McGraw-Hill, 1989, pp.1.89-1.100
10. SANS62051, "1.00 Electricity metering – Glossary of terms," SABS, 1999
11. SANS10142, "The wiring of premises, Part1: Low-voltage installations", SABS, 2003
12. E. Hughes, Electrical Technology. Edinburgh Gate: Addison Wesley Longman Limited, 1998

Appendix
Firmware

By
Jacob Jacobus Greeff (3692-247-1)

In partial fulfillment of the requirements
for the Ndip: Electrical Engineering

Table of Contents

1. Trade secrets and confidentiality....3
2. Embedded Operating System Architecture....4
3. Contactor Firmware....6
4. Power Measurement Firmware....7
....7
5. Contact List....8
6. Bibliography....9

1. Trade secrets and confidentiality

Due to the fact that large parts of the firmware embedded operating system (EOS) are currently still under development, but a large part of it is involved in a number of pending patents, it has been requested that I only publish those parts of the system that have direct impact on the system as I have functionally described it in this design report. It is however necessary to understand the core functionality of the system, in order to understand the code that has been implemented to perform functions. The functional components that I am going the publish the code for are:

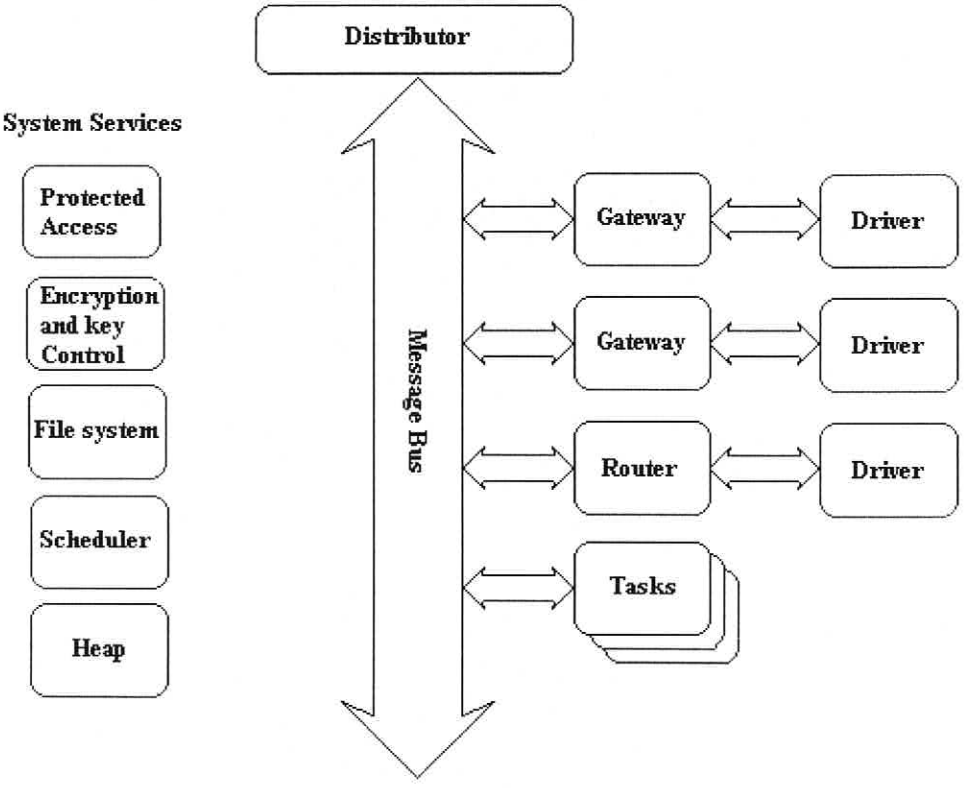
- The actual contactor switching code, which includes the ability to switch on the zero point, as well as the ability to sense alarm events like faulty contactors and bridged out terminals.
- The measurement driver and gateway that handles the measurement of power consumption, as well as calculating the amount of power that has been consumed over time (kWh register.)

the parts of the system that I will however not be able to publish code for are:

- The File System
- The Heap
- The Messaging structure
- The Scheduling and mutex management
- The Ring bus implementation
- The System routing topology

I will however give a brief overview of the system that should allow the code to become more understandable in this context.

2. Embedded Operating System Architecture



The embedded operating system is coded almost entirely in C++ and therefor makes full use of the object orientated advantages that the language offers. The embedded operating system is based quite heavily on the enterprise service bus architecture. Almost all of the firmware components are descendants of the classes in the above diagrams. The operating system makes a clear split between the higher level functions and the hardware abstraction layer. All data drivers are however wrapped as a generic driver to allow the system to access them the same way regardless of the platfrom. Below is a short summary of the functionality of these base classes and system services.

The operating system consists of the following components:

- Distributor: The distributor is responsible for controlling the message routing as well as the expiration of messages and the storing of messages to file on system power down. The distributor, and message queue are implemented as singletons.
- Message bus: The message bus is part of the distributor and gets the messages in the order in which they were queued by the distributor and hands them to the different registered components
- Protected access: allows the storage of those components that need to be stored, and not accessed again outside of the confines of the operating system (Encryption data, etc.)
- Encryption and key control: Implements the encryption of data and messages as well as some of the routing.

- File system: The file system allows data to be stored for later use and retrieval. The file system handles potential intermittent power conditions by being fully transactional, and implementing shadowing on the lower level.
- Scheduler: Allows the programmer to have his task trigger a function at a set period, or once only. Timers are implemented by simply wrapping the descendant class in an abstract base class that implements the callback function as a virtual function.
- Heap: The allows the allocation of objects to memory without the need to worry about fragmentation
- Gateways: Gateways allow tasks to get data to and from drivers using a standardized message format, without breaking the abstraction layer in any way. This allows the same task to get data from any number of sources as messages are always fire and forget type objects.
- Drivers: The drivers are part of the hardware abstraction layer, which means that they are different depending on the processor that is being run.
- Routers: Routers are like gateways, except for the fact that they contain no intelligence of their own, except for a filter that allows messages from outside of the system to enter into it, and messages inside the system to be transmitted out of the system. Encryption is all handled inside the routers, as it is only needed on the boundary between the inside and outside of the firmware operating system.
- Tasks: The tasks contain the actual functionality of the system, and are able to access the rest of the operating system only through messages, except for the system services. Most of the firmware lies in the tasks, and due to the architecture implementation, all of the operating system, except for the tasks is portable between any one of the devices in the system.

3. Contactor Firmware

3.1 gw_contactor.h

```
#ifndef __EOS_GATEWAYS_CONTACTOR__
#define __EOS_GATEWAYS_CONTACTOR__

#include "sys_Definitions.h"
#include "drv_Timer.h"
#include "drv_Capture_Notify.h"
#include "gw_Gateway.h"
#include "sys_CustomTypes.h"
#include "drv_Gpio.h"
#include "sys_Scheduler.h"
#include "msgs_Contactor.h"

namespace eos_gateways {
 using namespace eos_drivers;

 /**
 * ContactorPins
 * -------------
 * Since the design implementation of the contactor is a single coil contactor
 * two output pins are required, one for open and one for closed for either
 * side of the H-Bridge driver
 */
 struct ContactorPins {
  public:

    /**
    * Populate the contactor pins structure
    */
    ContactorPins () :
     openPin (new OutputPin(CONTACTOR_OPEN_CLOSE, CONTACTOR_OPEN_PORT, CONTACTOR_OPEN_PIN, false,
NULL)),
     closePin (new OutputPin(CONTACTOR_OPEN_CLOSE, CONTACTOR_CLOSE_PORT, CONTACTOR_CLOSE_PIN, false,
NULL)) {
    }

    // The output pin that controls switching into the "CLOSED" state
    OutputPin* closePin;
    // the output pin that controls switching into the "OPEN" state
    OutputPin* openPin;
 };

/**
* ContactorGateway
* ----------------
* The contactor gateway is responsible for switching of the external contactor which in turn feeds
* or cuts power to a premisis. The gateway needs to determine the best possible moment to close/open
* the physical contactor based on information it receives from a zero crossing detector on the measurement
* chip and a comparatored input on the other side of the contactor allows us to calibratewhen the switching
* should take place. It should always open and close at a zero crossing in order to extend the life
* of the contactor!
*/
 class ContactorStateMachine;
 class ContactorGateway : public Gateway , public CaptureNotify, public TimerCallback {
  public:

    /**
```

```
    * Create the Contactor Gateway
    */
    ContactorGateway();

    /**
    * Process a message that has been delivered by the distributor
    */
    virtual void processMessage(Message *message);

    /**
    * This gets called when we get a zero crossing on the input power supply
    */
    virtual void notifyCapture(DWord pulseLength, CaptureChannel* channel);

    /*
    * callback from the calibration timer.
    */
    void switchContactor();

    /**
    * The callback function that gets called when a timer expires
    * @param timerId - CONTACTOR_SWITCH : clears the contactor ios when it is switched
    *                  ZERO_CROSSING_CHECK : check if a pulse was received in the alotted time
    */
    virtual void timerExpired(Word timerId);

    /**
    * Switch the contactor into a new state(CONTACTOR_CLOSED or CONTACTOR_OPEN). This is usually called
    * on the next zero crossing pulse of the primary zeroCrossing measurement.
    */
    void triggerContactorSwitch(LogicalState newState);

    /**
    * Sends a message to indicate the new state of the contactor
    */
    void sendContactorStateMessage(ContactorState newState);

    /**
    * Return the logicalState of the contactor
    */
    LogicalState getContactorState();

private:

    /**
    * switches the pulse driver over to thge required polarity on all channels
    */
    void setPulsePolarity(CapturePolarity polarity);

    /**
    * Update our average cycle pulse width
    */
    void updateAverageCycleWidth(DWord pulseWidth);

    /**
    * If the contactor is open, then we get a pulses for the positive and negative side
    * of the cycle along with the reference pulse. Therefor, if closedCount reaches its threshold,
    * we can assume that the contactor is closed, and below the threshold, it is open.
    */
    void updateContactorState(CaptureChannelNo channelNo);

    /**
```

```
    * If a switch has been triggered, do the switch
    */
    void switchContactorIfRequired();

    /**
    * Calibrate the transition that has just happenend.
    * @param contactorstate is the state to calibrate(CONTACTOR_CLOSING or CONTACTOR_OPENING)
    * @param pulseWidth is the pulse received that caused the calibration to be initiated
    */
    void calibrate(Word pulseWidth);

    ContactorPins *contactorPins;
    // The driver for the calibration timer(timer 3)
    DownCounter* switchingTimer;
    // This timer allows us to detect when our reference zero cross pulse disappears.
    PollTimer *zeroCrossFailTimer;
    // Create a contactor state machine to track the state changes of the contactor
    ContactorStateMachine *stateMachine;
    // The averge length of a full AC wave in timer ticks
    DWord aveCycleWidth;
    // This will be true if we haven't recieved a pulse width yet
    bool firstPulse : 1;
    // Indicate whether or not we need to switch during a zero crossing
    bool doSwitch : 1;
    // Indicates whether we have already done a calibration on a pulse
    bool doCalibrate : 1;
    // Indicates whetehr there is a zero crossing failure
    bool zeroCrossFail: 1;
    // Indicates whether we've checked the zero crossing before
    bool firstZeroCrossCheck : 1;
    // If the closedCount reaches a defined value, then we can assume that the contactor is closed
    Byte closedCount : 3;
    // The time in ticks to wait from when we get zero crossing to when we switch open
    DWord openTimeout : 24;
    // The time in ticks to wait from when we get a zero crossing to when we switch closed
    DWord closeTimeout : 24;
};

/**
* ContactorStateMachine
* ---------------------
* The contactor state machine tracks and controls the contactors states depending on incoming
* events received from the contactor gateway.
*/
class ContactorStateMachine {
  public:

    /**
    * Create a contactor state machine that will track the state of, and control the contactor
    * @param gateway is a pointer back to the contactor gateway
    */
    ContactorStateMachine (ContactorGateway *gateway);

    /**
    * Get the current state of the contactor state machine
    * @return the current state of the state machine
    */
    ContactorState getState ();

    /**
    * All state machine events are processed through this guy. The state will be updated depending
    * on what state we are already in.
```

```
    * @param event is an event that has occurred and indicated a possible state change.
    */
    void updateStateWithEvent (ContactorEvent event);

  private:

    /**
    * Work out the state we need to move into for the received event when we
    * have just started up, and don't know what state we are in yet.
    * @param event is an external event that has occured in the contactor gateway
    */
    ContactorState updateIdle (ContactorEvent event);

    /**
    * Work out the state we need to move into for the received event once we
    * have begun to switch the contactor into an open state.
    * @param event is an external event that has occured in the contactor gateway
    */
    ContactorState updateSwitchingOpen (ContactorEvent event);

    /**
    * Work out the state we need to move into for the received event once we have
    * completed switching to an open state, and are awaiting confirmation of the switch.
    * @param event is an external event that has occured in the contactor gateway
    */
    ContactorState updateVerifyOpen (ContactorEvent event);

    /**
    * Work out the state we need to move into for the received event once we
    * have begun to switch the contactor into a closed state,
    * @param event is an external event that has occured in the contactor gateway
    */
    ContactorState updateSwitchingClosed (ContactorEvent event);

    /**
    * Work out the state we need to move into for the received event once we have
    * completed switching to a closed state, and are awaiting confirmation of the switch.
    * @param event is an external event that has occured in the contactor gateway
    */
    ContactorState updateVerifyClosed (ContactorEvent event);

    /**
    * Set the LED state to reflect the current state of the contactor
    */
    void updateLed (LogicalState state);


    // A pointer back to the contactor gateway
    ContactorGateway *gateway;
    // The current state of the contactor state machine
    ContactorState state;
    // Used to time for a period after switching after which if we are in the wrong state, raise alarm
    PollTimer *timer;
    // a counter to stop flooding error messages
    Byte errorCounter;
    // The previous state the contactor was in
    LogicalState expectedState;
};

}

#endif
```

3.2 gw_Contactor.cpp

```
#include "gw_Contactor.h"
#include "plt_System.h"
#include "sys_CriticalSection.h"
#include "sys_Error.h"
#include "sys_Logger.h"
#include "msg_Distributor.h"
#include "msgs_Gpio.h"
#include "msgs_Contactor.h"
#include "msgs_EventManager.h"


namespace eos_gateways {
  using namespace eos_system;

//timer definitions for the contactorGateway
#define TIMER_CONTACTOR_STATE_CHECK 0
#define TIMER_CONTACTOR_PROCESS_IO  1
#define TIMER_CONTACTOR_SWITCH      2
#define TIMER_ZERO_CROSS_CHECK      3
// Define the number of pulses of the reference without pulses on the other side before deciding state
#define CLOSED_COUNT_THRESHOLD      4
// The percentage tolerance to calibrate on
#define TOLERANCE                   0

/**
* ContactorGateway
* ----------------
* The contactor gateway is responsible for switching of the external contactor which in turn feeds
* or cuts power to a premisis. The gateway needs to determine the best possible moment to close/open
* the physical contactor based on information it receives from a zero crossing detector and an analogue
* waveform of the AC supply that it constructs from the ADC. It should always open and close at a zero
* crossing in order to extend the life of the contactor!
*/
  /**
  * Create the Contactor Gateway
  */
  ContactorGateway::ContactorGateway() :
    Gateway(GATEWAY_CONTACTOR),
    contactorPins(new ContactorPins()),
    zeroCrossFailTimer(new PollTimer(1000)),
    switchingTimer(new DownCounter(TIMER_TIM3, TIMER_CONTACTOR_SWITCH)),
    firstPulse(true),
    doSwitch(false),
    doCalibrate(false),
    zeroCrossFail(false),
    firstZeroCrossCheck(true),
    aveCycleWidth(0),
    closedCount(0),
    openTimeout(0x32),
    closeTimeout(0x64) {
    // Can't do this in the initialization block as "this" doesn't exist
    stateMachine = new ContactorStateMachine(this);
    // Set up the switching timer
    switchingTimer->initialise(this);
    // Set up the capture channels for the gateway
    CaptureDriver::getInstance()->enableChannel(POSITIVE_CAPTURE_CHANNEL, this);
    CaptureDriver::getInstance()->enableChannel(NEGATIVE_CAPTURE_CHANNEL, this);
    CaptureDriver::getInstance()->enableChannel(REFERENCE_CAPTURE_CHANNEL, this);
    // Schedule a timer to check the state of our contactor
    Scheduler::getInstance()->schedulePeriodic(100, this, LOW_PRIORITY_TIMER, TIMER_CONTACTOR_STATE_CHECK);
```

```
  Scheduler::getInstance()->schedulePeriodic(5000, this, LOW_PRIORITY_TIMER, TIMER_ZERO_CROSS_CHECK);
}

/**
* Process a message that has been delivered by the distributor
*/
void ContactorGateway::processMessage(Message *message){
 switch (message->getMessageId()) {
  case MSG_ID_CONTACTOR_CRC_REQUEST :
    sendMessage(ContactorMessages::createContactorStateCRCMessage(getContactorState()));
   break;

  case MSG_ID_CONTACTOR_SWITCH: {
    // Iterate through the list of serial numbers in the message and see if one matches ours...
    Byte numSerials = message->readByte(0);
    QWord serialNumber = Credentials::getInstance()->getSerialNumber();
    for (Byte i = 0; i < numSerials; i++) {
     QWord switchSerial = message->readQWord(1 +( i * 8));
     // If one of the serial numbers match, then read the state we should be in and attempt to switch
     if (switchSerial == serialNumber) {
      LogicalState newState = (LogicalState)message->readWord(1 +(numSerials*8));
      if (newState == CONTACTOR_OPEN) {
       Logger::getInstance()->logInfo(SYSTEM, "CONTACTOR: got a open req");
       stateMachine->updateStateWithEvent(CONTACTOR_EVENT_OPEN_REQUEST);
      }
      else if (newState == CONTACTOR_CLOSED) {
       Logger::getInstance()->logInfo(SYSTEM, "CONTACTOR: got a close req");
       stateMachine->updateStateWithEvent(CONTACTOR_EVENT_CLOSE_REQUEST);
      }
      break;
     }
    }
   }
   break;
  case MSG_ID_SET_STATE:
    LogicalId logicalId =(LogicalId)message->readWord(0);
    LogicalState state =(LogicalState)message->readWord(2);
    if (state == CONTACTOR_OPEN) {
     Logger::getInstance()->logInfo(SYSTEM, "CONTACTOR: got a open req");
     stateMachine->updateStateWithEvent(CONTACTOR_EVENT_OPEN_REQUEST);
    }
    else if (state == CONTACTOR_CLOSED) {
     Logger::getInstance()->logInfo(SYSTEM, "CONTACTOR: got a close req");
     stateMachine->updateStateWithEvent(CONTACTOR_EVENT_CLOSE_REQUEST);
    }
   break;
 }
}

/**
* This gets called when we get a zero crossing on the input power supply. The pulse is checked
* to see if an uncalibrated transition has occured in the pulse, and if it has calibrate it. the
* calibration goal is to get the contactor to switch on/off in the last 5% of the half cycle
*/
void ContactorGateway::notifyCapture(DWord pulseWidth, CaptureChannel* channel) {
 CaptureChannelNo channelNo = channel->getChannelNo();
 updateContactorState(channelNo);
 // If this is for our reference channel, we measure cycle width and switch if neccessary
 if (channelNo == REFERENCE_CAPTURE_CHANNEL) {
  updateAverageCycleWidth(pulseWidth);
  switchContactorIfRequired();
  zeroCrossFailTimer->startTimer();
```

```
    }
    // Else if this is either the positive or the negative half cycle indication pulses
    else {
      // If we get a cycle that is between half and a whole cycle(within some tolerance), while we are opening or
      // closing the contactor, then a switch has occurred, and we should update our calibration values.
      ContactorState contactorState = stateMachine->getState();
      if ((doCalibrate == true) &&(contactorState == CONTACTOR_STATE_VERIFY_OPEN) ||(contactorState ==
CONTACTOR_STATE_VERIFY_CLOSED)) {
        Word halfCycle = aveCycleWidth / 2;
        Word tolerance =(aveCycleWidth * TOLERANCE) / 100;
        if ((pulseWidth >(halfCycle + tolerance)) &&(pulseWidth <(aveCycleWidth - tolerance)) &&(pulseWidth >= tolerance)) {
          calibrate(pulseWidth);
        }
      }
    }
  }

  /**
  * Callback from the calibration timer. this means the calibration has been implemented and the contactor can now
  * switch safely as the mechanical contacts will open/close on the zero point +- 5%
  */
  void ContactorGateway::switchContactor() {
    // Disable the output lines and timers in case we want to switch before we complete switching
    contactorPins->openPin->processTimeout();
    contactorPins->closePin->processTimeout();
    Scheduler::getInstance()->disableTimer(this, TIMER_CONTACTOR_PROCESS_IO);
    // Pulse the contactors open pin and notify the state machine that we have attempted a switch
    if (stateMachine->getState() == CONTACTOR_STATE_SWITCHING_OPEN) {
      contactorPins->closePin->setIoState(GPIO_INACTIVE);
      contactorPins->openPin->setIoState(GPIO_ACTIVE, 1, 1, 1);
      stateMachine->updateStateWithEvent(CONTACTOR_EVENT_SWITCHED_OPEN);
    }
    // Pulse the contactors close pin and notify the state machine that we have attempted a switch
    else if (stateMachine->getState() == CONTACTOR_STATE_SWITCHING_CLOSED) {
      contactorPins->openPin->setIoState(GPIO_INACTIVE);
      contactorPins->closePin->setIoState(GPIO_ACTIVE, 1, 1, 1);
      stateMachine->updateStateWithEvent(CONTACTOR_EVENT_SWITCHED_CLOSED);
    }
    // Schedule the deactivation of the active contactor output pins
    Scheduler::getInstance()->scheduleOneShot(50, this, LOW_PRIORITY_TIMER, TIMER_CONTACTOR_PROCESS_IO);
  }

  /**
  * The callback function that gets called when a timer expires
  * @param timerId is used to identify the timer that expired
  */
  void ContactorGateway::timerExpired(Word timerId) {
    switch (timerId) {
      case TIMER_CONTACTOR_STATE_CHECK :
        // Heartbeat to the state machine
        stateMachine->updateStateWithEvent(CONTACTOR_EVENT_TIMER);
        break;
      case TIMER_ZERO_CROSS_CHECK:
        // If the timer never starts, or if the it times out, then we have a zero crossing failure
        if (((zeroCrossFailTimer->stopped() == true) ||(zeroCrossFailTimer->timedOut() == true)) &&(zeroCrossFail == false)) {
          sendMessage(EventManagerMessages::createAddNewEventMessage(EVENT_ZERO_CROSS_FAIL, 0));
          zeroCrossFailTimer->stopTimer();
          zeroCrossFail = true;
          firstZeroCrossCheck = false;
        }
        // If the timer starts running after a failure, or its running the first time we're checking, send a restored event
        else if (((zeroCrossFailTimer->stopped() == false) &&(zeroCrossFailTimer->timedOut() == false)) &&
```

```
            ((zeroCrossFail == true) ||(firstZeroCrossCheck == true))) {
          sendMessage(EventManagerMessages::createAddNewEventMessage(EVENT_ZERO_CROSS_RESTORED, 0));
          zeroCrossFail = false;
          firstZeroCrossCheck = false;
        }
      break;
    case TIMER_CONTACTOR_PROCESS_IO :
      // Clear both the IOs for the contactor by calling their timerCallback functions(immitating the GPIO 20ms timeout.)
      contactorPins->openPin->processTimeout();
      contactorPins->closePin->processTimeout();
      break;
    case TIMER_CONTACTOR_SWITCH :
      switchContactor();
      break;
  }
}

/**
* triggers a sequence which will switch the cintactor into the desired state on the next zero cross.
*/
void ContactorGateway::triggerContactorSwitch(LogicalState newState) {
  ASSERT((newState == CONTACTOR_OPEN) ||(newState == CONTACTOR_CLOSED));
  doSwitch = true;
  doCalibrate = true;
  if (newState == CONTACTOR_CLOSED)
    setPulsePolarity(CAPTURE_NON_INVERTED);
  else if (newState == CONTACTOR_OPEN)
    setPulsePolarity(CAPTURE_INVERTED);
}

/**
* Switches the pulse driver over to the required polarity on all channels
*/
void ContactorGateway::setPulsePolarity(CapturePolarity polarity) {
  CRITICAL {
    CaptureDriver::getInstance()->setPolarity(polarity, POSITIVE_CAPTURE_CHANNEL);
    CaptureDriver::getInstance()->setPolarity(polarity, NEGATIVE_CAPTURE_CHANNEL);
  }
}

/**
* Update our average cycle pulse width
*/
void ContactorGateway::updateAverageCycleWidth(DWord pulseWidth) {
  // If this is the first pulse we have gotten, then capture the initial length, else average
  if (firstPulse == true) {
    aveCycleWidth = pulseWidth;
    firstPulse = false;
  }
  else {
    aveCycleWidth =(aveCycleWidth + pulseWidth) / 2;
  }
}

/**
* If the contactor is open, then we get a pulses for the positive and negative side
* of the cycle along with the reference pulse. Therefor, if closedCount reaches its threshold,
* we can assume that the contactor is closed, and below the threshold, it is open.
*/
void ContactorGateway::updateContactorState(CaptureChannelNo channelNo) {
  if (channelNo == REFERENCE_CAPTURE_CHANNEL) {
    // If closedCount gets to its set threshld, then the contactor is considered to be closed
```

```
    if(++closedCount > CLOSED_COUNT_THRESHOLD) {
      closedCount = CLOSED_COUNT_THRESHOLD;
    }
   }
   else {
    closedCount = 0;
   }
}

/**
* Return the logicalState of the contactor
*/
LogicalState ContactorGateway::getContactorState() {
  if (closedCount >= CLOSED_COUNT_THRESHOLD) {
    return CONTACTOR_CLOSED;
  }
  else {
    return CONTACTOR_OPEN;
  }
}

/**
* If a switch has been triggered, do the switch
*/
void ContactorGateway::switchContactorIfRequired() {
  if (doSwitch == true) {
    if(stateMachine->getState() == CONTACTOR_STATE_SWITCHING_CLOSED) {
      Logger::getInstance()->logInfo(SYSTEM, String("Close ")+closeTimeout);
      switchingTimer->setTimeout(closeTimeout);
      switchingTimer->startTimer();
      doSwitch = false;
    }
    if (stateMachine->getState() == CONTACTOR_STATE_SWITCHING_OPEN) {
      Logger::getInstance()->logInfo(SYSTEM, String("Open ")+openTimeout);
      switchingTimer->setTimeout(openTimeout);
      switchingTimer->startTimer();
      doSwitch = false;
    }
  }
}

/*
* calibrate the transition that has just happenend.
*/
void ContactorGateway::calibrate(Word pulseWidth) {
  ContactorState contactorState = stateMachine->getState();
  Word halfCycle = aveCycleWidth / 2;
  Word tolerance =(aveCycleWidth * TOLERANCE) / 100;
  if (contactorState == CONTACTOR_STATE_VERIFY_OPEN) {
    openTimeout =(openTimeout +(pulseWidth - halfCycle -(tolerance / 2))) % halfCycle;
    if (openTimeout == 0)
      openTimeout = 5;
  }
  else if (contactorState == CONTACTOR_STATE_VERIFY_CLOSED) {
    closeTimeout =(closeTimeout +(aveCycleWidth - pulseWidth -(tolerance / 2))) % halfCycle;
  }
}

/**
* Sends a message to indicate the new state of the contactor
*/
void ContactorGateway::sendContactorStateMessage(ContactorState newState) {
```

```
}

/**
* ContactorStateMachine
* ---------------------
* The contactor state machine tracks and controls the contactors states depending on incoming
* events received from the contactor gateway.
*/
 /**
 * Create a contactor state machine that will track the state of, and control the contactor
 * @param gateway is a pointer back to the contactor gateway
 */
ContactorStateMachine::ContactorStateMachine(ContactorGateway *gateway_) :
  gateway(gateway_),
  timer(new PollTimer(200)),
  state(CONTACTOR_STATE_IDLE),
  expectedState(INVALID_STATE),
  errorCounter(0) {
}

 /**
 * Get the current state of the contactor state machine
 * @return the current state of the state machine
 */
ContactorState ContactorStateMachine::getState() {
  return state;
}

 /**
 * All state machine events are processed through this guy. The state will be updated depending
 * on what state we are already in.
 * @param event is an event that has occurred and indicated a possible state change.
 */
void ContactorStateMachine::updateStateWithEvent(ContactorEvent event) {
  ContactorState newState;
  switch (state) {
    case CONTACTOR_STATE_IDLE :
       newState = updateIdle(event);
      break;
    case CONTACTOR_STATE_SWITCHING_OPEN :
       newState = updateSwitchingOpen(event);
      break;
    case CONTACTOR_STATE_VERIFY_OPEN :
       newState = updateVerifyOpen(event);
      break;
    case CONTACTOR_STATE_SWITCHING_CLOSED :
       newState = updateSwitchingClosed(event);
      break;
    case CONTACTOR_STATE_VERIFY_CLOSED :
       newState = updateVerifyClosed(event);
      break;
  }
  // Update to the new state
  state = newState;
}

 /**
 * Work out the state we need to move into for the received event when we
 * have just started up, and don't know what state we are in yet.
 * @param event is an external event that has occured in the contactor gateway
 */
```

```
ContactorState ContactorStateMachine::updateIdle(ContactorEvent event) {
 switch (event) {
  case CONTACTOR_EVENT_TIMER:
    // If the timer is stopped, start it so we can check the contactor state every 500mS
    if (timer->stopped() == true) {
     timer->setTimeout(500);
     timer->startTimer();
    }
    //
    if (timer->timedOut() == true) {
     LogicalState contactorState = gateway->getContactorState();
     if (expectedState == INVALID_STATE) {
      expectedState = contactorState;
      updateLed(contactorState);
     }
     // If the contactor unexpectedly changes state, notify that there's a problem
     if (contactorState != expectedState) {
      if (expectedState == CONTACTOR_OPEN) {
       timer->stopTimer();
       return CONTACTOR_STATE_SWITCHING_OPEN;
      }
      else if(expectedState == CONTACTOR_CLOSED) {
       timer->stopTimer();
       return CONTACTOR_STATE_SWITCHING_CLOSED;
      }
     }
    }
    break;
  case CONTACTOR_EVENT_OPEN_REQUEST:
    timer->stopTimer();
    return CONTACTOR_STATE_SWITCHING_OPEN;
  case CONTACTOR_EVENT_CLOSE_REQUEST:
    timer->stopTimer();
    return CONTACTOR_STATE_SWITCHING_CLOSED;
 }
 return state;
}

/**
* Work out the state we need to move into for the received event once we
* have begun to switch the contactor into an open state.
* @param event is an external event that has occured in the contactor gateway
*/
ContactorState ContactorStateMachine::updateSwitchingOpen(ContactorEvent event) {
 switch(event) {
  case CONTACTOR_EVENT_TIMER:
    // Trigger a switch of the contactor until we get a switched event
    if((timer->stopped() == true) ||(timer->timedOut() == true)) {
     gateway->triggerContactorSwitch(CONTACTOR_OPEN);
     timer->setTimeout(100);
     timer->startTimer();
    }
    break;
  case CONTACTOR_EVENT_SWITCHED_OPEN:
    // When we get told that the gateway tried to switch...
    timer->stopTimer();
    return CONTACTOR_STATE_VERIFY_OPEN;
    break;
  case CONTACTOR_EVENT_CLOSE_REQUEST:
    // If we get interrupted with a request to close, abort the open
    timer->stopTimer();
    return CONTACTOR_STATE_SWITCHING_CLOSED;
```

```
      break;
    }
    return state;
  }

  /**
  * Work out the state we need to move into for the received event once we have
  * completed switching to an open state, and are awaiting confirmation of the switch.
  * @param event is an external event that has occured in the contactor gateway
  */
  ContactorState ContactorStateMachine::updateVerifyOpen(ContactorEvent event) {
    switch(event) {
      case CONTACTOR_EVENT_TIMER:
        // When we enter this state, set the timer to
        if(timer->stopped() == true) {
          timer->setTimeout(200);
          timer->startTimer();
        }
        if(timer->timedOut() == true) {
          timer->stopTimer();
          LogicalState state = gateway->getContactorState();
          // If the contactor successfully changed state, we're happy, so update the state on the front LED!
          if(state == CONTACTOR_OPEN) {
            updateLed(state);
            gateway->sendMessage(EventManagerMessages::createAddNewEventMessage(EVENT_OPEN_RESTORED, 0));
            expectedState = CONTACTOR_OPEN;
            Distributor::getInstance()->queueMessage(ContactorMessages::createContactorStateMessage(CONTACTOR_OPEN));
            return CONTACTOR_STATE_IDLE;
          }
          else if(state == CONTACTOR_CLOSED) {
            Logger::getInstance()->logError(GATEWAY_CONTACTOR, "Contactor failed to open!");
            gateway->sendMessage(EventManagerMessages::createAddNewEventMessage(EVENT_OPEN_FAIL, 0));
            Distributor::getInstance()-
>queueMessage(ContactorMessages::createContactorErrorMessage(CONTACTOR_ERROR_FAILED_TO_OPEN));
            return CONTACTOR_STATE_SWITCHING_OPEN;
          }
        }
        break;
      case CONTACTOR_EVENT_CLOSE_REQUEST:
        // If we get interrupted with a request to close, abort the open
        timer->stopTimer();
        return CONTACTOR_STATE_SWITCHING_CLOSED;
        break;
    }
    return state;
  }

  /**
  * Work out the state we need to move into for the received event once we
  * have begun to switch the contactor into a closed state,
  * @param event is an external event that has occured in the contactor gateway
  */
  ContactorState ContactorStateMachine::updateSwitchingClosed(ContactorEvent event) {
    switch(event) {
      case CONTACTOR_EVENT_TIMER:
        // Trigger a switch of the contactor until we get a switched event
        if((timer->stopped() == true) ||(timer->timedOut() == true)) {
          gateway->triggerContactorSwitch(CONTACTOR_CLOSED);
          timer->setTimeout(100);
          timer->startTimer();
        }
        break;
```

```
    case CONTACTOR_EVENT_SWITCHED_CLOSED:
      // When we get told that the gateway tried to switch...
      timer->stopTimer();
      return CONTACTOR_STATE_VERIFY_CLOSED;
     break;
    case CONTACTOR_EVENT_OPEN_REQUEST:
      // If we get interrupted with a request to open, abort the close
      timer->stopTimer();
      return CONTACTOR_STATE_SWITCHING_OPEN;
     break;
   }
  return state;
}

/**
* Work out the state we need to move into for the received event once we have
* completed switching to a closed state, and are awaiting confirmation of the switch.
* @param event is an external event that has occured in the contactor gateway
*/
ContactorState ContactorStateMachine::updateVerifyClosed(ContactorEvent event) {
  switch(event) {
    case CONTACTOR_EVENT_TIMER:
      // When we enter this state, set the timer to
      if(timer->stopped() == true) {
       timer->setTimeout(200);
       timer->startTimer();
      }
      if(timer->timedOut() == true) {
       timer->stopTimer();
       LogicalState state = gateway->getContactorState();
       // If the contactor successfully changed state, we're happy, so update the state on the front LED!
       if(state == CONTACTOR_CLOSED) {
         updateLed(state);
         gateway->sendMessage(EventManagerMessages::createAddNewEventMessage(EVENT_CLOSE_RESTORED, 0));
         Distributor::getInstance()->queueMessage(ContactorMessages::createContactorStateMessage(CONTACTOR_CLOSED));
         return CONTACTOR_STATE_IDLE;
       }
       else if(state == CONTACTOR_OPEN) {
         Logger::getInstance()->logError(GATEWAY_CONTACTOR, "Contactor failed to close!");
         gateway->sendMessage(EventManagerMessages::createAddNewEventMessage(EVENT_CLOSE_FAIL, 0));
         Distributor::getInstance()-
>queueMessage(ContactorMessages::createContactorErrorMessage(CONTACTOR_ERROR_FAILED_TO_CLOSE));
         return CONTACTOR_STATE_SWITCHING_CLOSED;
       }
      }
     break;
    case CONTACTOR_EVENT_OPEN_REQUEST:
      // If we get interrupted with a request to open, abort the close
      timer->stopTimer();
      return CONTACTOR_STATE_SWITCHING_OPEN;
     break;
   }
  return state;
}

/**
* Set the LED state to reflect the current state of the contactor
*/
void ContactorStateMachine::updateLed(LogicalState state) {
  if(state == CONTACTOR_OPEN)
    gateway->sendMessage(GpioMessages::createSetLogicalStateMessage(LED_PLC_STATUS, LED_OFF));
  else if(state == CONTACTOR_CLOSED)
```

```
    gateway->sendMessage(GpioMessages::createSetLogicalStateMessage(LED_PLC_STATUS, LED_ON));
  expectedState = state;
 }

}
```

3.3 gw_Metering.h

```
#ifndef __EOS_GATEWAYS_METERING__
#define __EOS_GATEWAYS_METERING__

#include "gw_Gateway.h"
#include "drv_Timer.h"
#include "sys_Scheduler.h"
#include "drv_Capture_Notify.h"
#include "drv_Fram.h"
#include "msg_Message.h"
#include "msg_Distributor.h"

// the number of watts per pulse per second in milliwatt. Calculated from the hardware.
#define MILLIWATT_PER_PULSE 15860

namespace eos_gateways {

 struct CalibrationValues {
  // The value of totalPulses when we start a calibration cycle
  QWord startPulses;
  // The value of totalPulses when we stop a calibration cycle
  QWord stopPulses;
 };

/**
* MeteringGateway
* ---------------
* The metering gateway is responsible for collecting and storing both instantaneous as well as
* total power usage from the metering chip and broadcasting it to the rest of the system every second.
*/
 class MeteringGateway : public Gateway, public CaptureNotify, public TimerCallback {
  public:

   /**
   * Create the metering gateway
   */
   MeteringGateway ();

   /**
   * Process a message that has been delivered by the distributor
   */
   virtual void processMessage (Message *message);

   /**
   * This gets called when we get a zero crossing on the input power supply
   */
   virtual void notifyCapture (DWord pulseLength, CaptureChannel* channel);

   /**
   * The callback function that gets called when a timer expires
   * @param timerId - AVE_WATT_PERIOD: a 100ms check to see if a pulse has been received, if it
   * has, detirmine the current average Wattage of the system
   */
   virtual void timerExpired (Word timerId);

  private:

   /**
   * Calculates our current consuption given the number of pulses over the last second
   * @param pulses is a count of the number of pulses we have recieved in a second
```

```
    * @param timeSinceLast is the time in milliseconds since the last calculation
    * @return our current current consumption in kW/h
    */
    QWord calculatePowerConsumption (QWord pulses, Milliseconds timeSinceLast);

    /**
    * Start calibrating the device...
    */
    void startCalibration ();

    /**
    * Stop calibration.
    * @param expectedPowerConsumed is the total amount of power that we should have read...
    */
    void stopCalibration (QWord expectedPowerConsumed);

    // The calibration factor to use when calculating power consumption
    QWord calibrationFactor;
    // The current number of pulses we have received
    QWord totalPulses;
    // The number of pulses received over the last second
    DWord pulsesPerSecond;
    // The last time in milliceconds when the timer fired (used for calibration)
    Milliseconds lastSystemTime;
    // Structure used to store calibration start and end pulses during calibration
    CalibrationValues *calibrationValues;
};

}

#endif
```

3.4 gw_Metering.cpp

```
#include "gw_Metering.h"
#include "sys_CriticalSection.h"
#include "msgs_Metering.h"
#include "msgs_ReadingManager.h"
#include "sys_Logger.h"
#include "sys_Memory.h"

namespace eos_gateways {
 using namespace eos_utils;

// Timer definitions used by the metering gateway
#define TIMER_AVE_WATT_CHECK 1
#define TIMER_STORAGE 2
// The number of Watt seconds per pulse
#define WS_PER_PULSE  8.85
//
#define ASSUMED_VOLTAGE_VALUE 230
//
#define CALIBRATION_FACTOR_POSITION 0

/**
* MeteringGateway
* ---------------
* The metering gateway is responsible for collecting and storing both instantaneous as well as
* total power usage from the metering chip and broadcasting it to the rest of the system every second.
*/
 /**
 * Create the metering gateway
 */
 MeteringGateway::MeteringGateway () :
  Gateway (GATEWAY_METERING),
  calibrationValues (new CalibrationValues ()),
  totalPulses (0),
  pulsesPerSecond (0),
  calibrationFactor (1) {
  // Set up our pulse capture channel
  CaptureDriver::getInstance()->enableChannel (POWER_CAPTURE_CHANNEL, this);
  CaptureDriver::getInstance()->setPolarity (CAPTURE_NON_INVERTED, POWER_CAPTURE_CHANNEL);
  // Schedule our timer to do power calculations
  Scheduler::getInstance()->schedulePeriodic (950, this, HIGH_PRIORITY_TIMER, TIMER_AVE_WATT_CHECK);
  Scheduler::getInstance()->schedulePeriodic (10000, this, LOW_PRIORITY_TIMER, TIMER_STORAGE);
  lastSystemTime = Scheduler::getInstance()->getSystemTime ();
  // Update our conversionFactor and totalPulse registers from FRAM
  FramDriver *fram = FramDriver::getInstance ();
  Buffer *data = fram->slowRead (FRAM_MEASUREMENT_CALIBRATION_FACTOR, 16);
  calibrationFactor = data->readQWord ();
  totalPulses = data->readQWord ();
  deleteIfNotNull (data);
  if (calibrationFactor == 0)
   calibrationFactor = 88500;
  Logger::getInstance()->logInfo (GATEWAY_METERING, (String)("Calibration Factor ") + calibrationFactor);
 }

 /**
 * Process a message that has been delivered by the distributor
 */
 void MeteringGateway::processMessage (Message *message) {
  switch(message->getMessageId()) {
   case MSG_ID_POWER_START_CALIBRATION :
     startCalibration ();
```

```
      break;
    case MSG_ID_POWER_STOP_CALIBRATION :
      stopCalibration (message->readQWord (0));
      break;
    case MSG_ID_CLEAR_POWER_REGISTERS :
      // TODO clear the power registers.
      break;
   }
  }

  /**
  * This gets called whenever we get a pulse output from the metering chip. It signifies
  * that WS_PER_PULSE watt seconds of power have been used.
  */
  void MeteringGateway::notifyCapture (DWord pulseLength, CaptureChannel* channel) {
   totalPulses++;
   pulsesPerSecond++;
  }

  /**
  * The callback function that gets called when a timer expires
  * @param timerId is used to determine which time has expired
  */
  void MeteringGateway::timerExpired (Word timerId) {
   if (timerId == TIMER_AVE_WATT_CHECK) {
    Milliseconds timeSinceLastTimer = Scheduler::getInstance()->getTimeDifference (lastSystemTime);
    // Calculate our average power consumption, and current used in the last second
    QWord wattSeconds = 0;
    Word pulses = pulsesPerSecond;
    CRITICAL {
     wattSeconds = calculatePowerConsumption (pulsesPerSecond, timeSinceLastTimer);
     lastSystemTime = Scheduler::getInstance()->getSystemTime ();
     pulsesPerSecond = 0;
    }
    QWord amperes = wattSeconds / ASSUMED_VOLTAGE_VALUE;
    QWord totalConsumption = calculatePowerConsumption (totalPulses, 1000) / 3600000;
    Distributor::getInstance()->queueMessage (MeteringMessages::createPowerConsumptionMessage (wattSeconds, amperes,
totalConsumption));
    Logger::getInstance()->logInfo (GATEWAY_METERING, String("Consumption -> ") + wattSeconds + " : " + amperes + " : " +
totalConsumption + " : " + pulses);
   }
   else if (timerId == TIMER_STORAGE) {
    // Write our running total of the power consumption into the FRAM
    Buffer *data = new Buffer ();
    CRITICAL {
     data->writeQWord (totalPulses);
    }
    data->flip ();
    FramDriver *fram = FramDriver::getInstance ();
    fram->write (FRAM_MEASUREMENT_POWER_REGISTER, data);
   }
  }

  /**
  * Calculates our current consuption given the number of pulses over the last second
  * @param pulses is a count of the number of pulses we have recieved in a second
  * @param timeSinceLast is the time in milliseconds since the last calculation
  * @return our current current consumption in W/s
  */
  QWord MeteringGateway::calculatePowerConsumption (QWord pulses, Milliseconds timeSinceLast) {
   QWord powerInWattSeconds = (pulses * calibrationFactor * 1000) / timeSinceLast;
   return powerInWattSeconds;
```

```
  }

  /**
  * Start calibrating the device...
  */
  void MeteringGateway::startCalibration () {
   CRITICAL {
    calibrationValues->startPulses = totalPulses;
   }
  }

  /**
  * Stop calibration.
  * @param expectedPowerConsumed is the total amount of power that we should have read in watts
  */
  void MeteringGateway::stopCalibration (QWord expectedPowerConsumed) {
   CRITICAL {
    calibrationValues->stopPulses = totalPulses;
    // Calculate the calibration factor
    calibrationFactor = expectedPowerConsumed / (calibrationValues->stopPulses - calibrationValues->startPulses);
   }
   // Store it into FRAM
   Buffer *calibration = new Buffer ();
   calibration->writeQWord (calibrationFactor);
   calibration->flip ();
   FramDriver *fram = FramDriver::getInstance ();
   fram->write (FRAM_MEASUREMENT_CALIBRATION_FACTOR, calibration);
   Logger::getInstance()->logInfo (GATEWAY_METERING, (String)("Calibration Factor ") + calibrationFactor + " : " +
calibrationValues->startPulses + " : " + calibrationValues->stopPulses);
  }

}
```