UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETROTÉCNICA
CURSO DE ESPECIALIZAÇÃO EM ENERGIAS RENOVÁVEIS

DANIEL LINSINGEN PIAZZETTA

MONTAGEM DE UM SISTEMA DE AQUISIÇÃO DE DADOS PARA UMA INSTALAÇÃO SOLAR FOTOVOLTAICA CONECTADA À REDE ELÉTRICA

CURITIBA
2015

DANIEL LINSINGEN PIAZZETTA

MONTAGEM DE UM SISTEMA DE AQUISIÇÃO DE DADOS PARA UMA INSTALAÇÃO SOLAR FOTOVOLTAICA CONECTADA À REDE ELÉTRICA

Monografia, apresentada como requisito parcial para obtenção do título de especialista no curso de Pós-Graduação em Energias Renováveis, Departamento Acadêmico de Eletrotécnica, DAELT, Universidade Tecnológica Federal do Paraná, UTFPR.

Orientador: Prof. Dr. Jair Urbanetz Jr.

CURITIBA
2015

TERMO DE APROVAÇÃO

DANIEL LINSINGEN PIAZZETTA

MONTAGEM DE UM SISTEMA DE AQUISIÇÃO DE DADOS PARA UMA INSTALAÇÃO SOLAR FOTOVOLTAICA CONECTADA À REDE ELÉTRICA

Esta Monografia de Especialização foi apresentada no dia 08 de dezembro de 2015, como requisito parcial para obtenção do título de Especialista em Energia Renováveis – Departamento Acadêmico de Eletrotécnica – Universidade Tecnológica Federal do Paraná. O aluno foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____

Prof. Dr. Jair Urbanetz Junior
Coordenador de Curso de Especialização em Energias Renováveis

_____

Prof. Dr. Paulo Cícero Fritzen
Chefe do Departamento Acadêmico de Eletrotécnica

BANCA EXAMINADORA

_____

Prof. Dr. Jair Urbanetz Junior
Orientador - UTFPR

_____

Prof. Dr. Gerson Máximo Tiepolo
UTFPR

_____

Prof. Esp. Carlos Henrique Karam Salata
UTFPR

"O termo de aprovação assinado encontra-se na Coordenação do Curso"

RESUMO

Este trabalho objetiva verificar, na forma de uma pesquisa aplicada, a viabilidade técnica para elaboração de um sistema destinado a aquisitar as medidas elétricas de um sistema fotovoltaico conectado à rede elétrica de distribuição de energia. É notável a popularização dos sistemas de geração de energia autônoma – eólica e fotovoltaica – e, junto com a popularização dos sistemas, existe uma necessidade de se monitorar e verificar a eficácia destes sistemas. A medição de grandezas em corrente alternada é bastante difundida e existem diversos equipamentos comerciais, muitos com custo bastante atrativo, disponíveis no mercado varejista. A medição de grandezas em corrente contínua, ao contrário, não possui tão vasta disponibilidade, desta forma foi construído e ensaiado um dispositivo eletrônico para coleta das grandezas elétricas do sistema fotovoltaico.

Palavras chave: Sistema de Medição, Sistema Fotovoltaico Conectado à Rede.

ABSTRACT

This study target is to verify, in a applied research, the technical viability to develop a system to acquire electrical measures from a grid tied photovoltaic system. With the increasing of the availability of autonomous energy generation systems – Wind Power and photovoltaic – and, tied to the popularizations of such kind of systems, there will have a need to monitor and measure the efficiency of the installed systems. The alternate current acquisition system is preety common and easy to find in the retail marked with a good cost to benefit relation. The measured variables in DC, in contrast, does not have as wide availability in this way was built and tested an electronic device to collect the electrical parameters of the PV system.

Key Words: Measurement system, Grid-Tie Photovoltaic System.

SUMÁRIO

ÍNDICE DE FIGURAS

ÍNDICE DE GRÁFICOS

1. INTRODUÇÃO

A demanda mudial por energia cresce em um ritmo bastante acelerado, e a busca por fontes de energia menos poluentes e com menor impacto ambiental também torna-se relevante. Em relação à energia elétrica também se observa essa mesma tendência, sendo que no cenário mundial as principais fontes de energia elétrica são dependentes de combustíveis fosseis e em menor proporção provenientes de fontes renováveis. Este cenário, porém, vem se alterando gradativamente de forma que as energias renováveis começam a ter uma maior participação.

No Brasil, esta situação é inversa a que está ocorrendo no cenário mundial, ou seja, a matriz elétrica nacional é predominantemente renovável, (Figura 1) onde se destaca a participação das usinas hidrelétricas com 65,2% da produção de energia elétrica. As demais fontes renováveis estão em ritmo ascendente, porém ainda em menor escala (BEN, 2015). Devido à crise hídrica observada nos últimos anos, a participação das usinas que se utilizam de combustíveis fosséis aumentou.

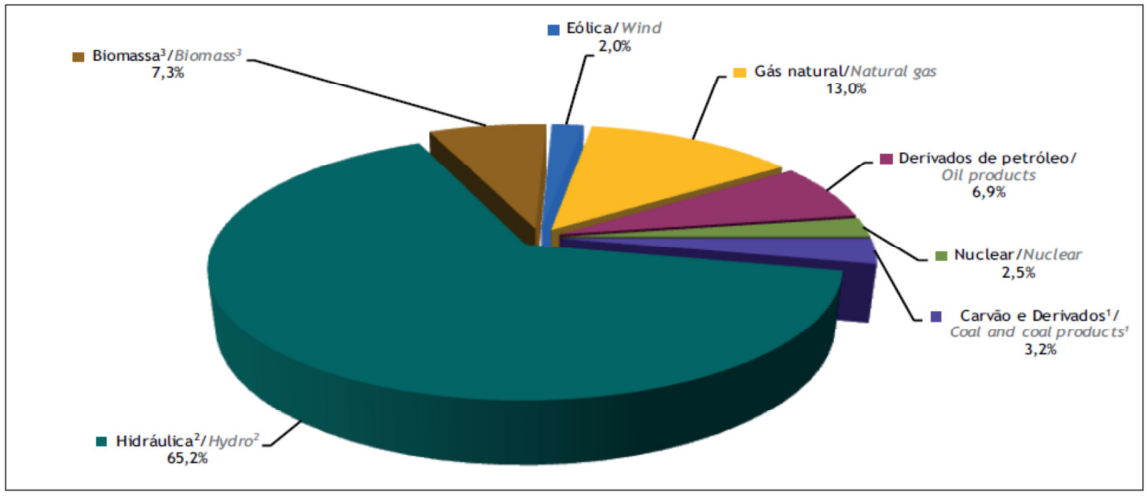


Figura 1 – Oferta Interna de Energia Elétrica por Fonte

Fonte: BEN, 2015

Entre as renováveis não-hidro, apresenta-se em destaque no cenário mundial a energia solar fotovoltaica, que tem tido uma adição na sua capacidade instalada significativa. No Brasil a partir de 2012, com a regumentação por parte da Agência

Nacional de Energia Elétrica (ANEEL), através da Resolução Normativa 482/2012 possibilitou-se a inserção deste tipo de fonte pelo consumidor final, ou seja, passa a ser permitida a instalação de sistemas fotovoltaicos para produzir energia elétrica diretamente no ponto de consumo (ANEEL, 2012).

A possibilidade de gerar energia elétrica de forma distribuída implica em postergar investimentos com geração e transmissão, além de que a geração fotovoltaica é não poluente, e quando instalada junto as edificações não necessita de área adicional, reduzindo, portanto os impactos provenientes da implementação de grandes usinas.

Dada a importância da energia elétrica no mundo atual, surgiu também a preocupação em compreender o funcionamento dos diversos tipos de cargas, bem como o comportamento da rede elétrica perante esses diferentes tipos. A grande quantidade de eletrônicos conectados na rede elétrica pode afetar a qualidade da energia elétrica e, portanto, reitera a importância de sistemas de medição capazes de monitorar as grandezas elétricas (EPRI - 2015).

Diante do exposto, as gerações distribuídas a partir dos sistemas fotovoltaicos podem apresentar benefícios ao terem suas grandezas elétricas monitoradas.

1.1 TEMA

Desenvolvimento de um sistema de medição e armazenamento de dados para aplicação em instalações fotovoltaicas conectadas à rede elétrica.

1.2 DELIMITAÇÃO DA PESQUISA

Este trabalho se utiliza de um microcontrolador da linha PIC de 8 bits para a elaboração de um circuito eletrônico capaz de medir e armazenar grandezas elétricas provenientes no lado de corrente contínua. Além disso, o sistema é composto de um multimedidor comercial para aquisição das grandezas elétricas no lado de corrente alternada. Ambos os sistemas de medição são aplicados a um sistema fotovoltaico, de baixo custo, conectado à rede elétrica.

1.3 PROBLEMAS E PREMISSAS

Com a crescente oferta de sistemas fotovoltaicos e a queda no preço dos mesmos, existe a tendência desses sistemas serem bastante populares nas casas dos brasileiros nos próximos anos. Embora a maioria dos usuários deste tipo de sistema não esteja preocupada com a monitoração das grandezas dos seus sistemas, sempre existirão aqules que desejam ter total controle e conhecimento sobre a atuação do mesmo. Um sistema para este monitoramento pode custar alguns milhares de reais, face a escassez de oferta de alguns tipos de equipamentos, em especial os de aquisição de medidas em corrente contínua. Aliado a este fator, outra informação interessante de se conhecer é a eficiência da conversão de energia realizada pelo inversor. Para poder calcular esta eficiência é necessário que estejam disponíveis informações sobre a energia disponível no lado de captação - lado de corrente contínua – e a energia efetivamente entregue ao sistema de distribuição, que poderá ser consumida pelas cargas anexas à rede de distribuição.

Face a isto, este trabalho procurou responder a seguinte questão:

- O sistema proposto é capaz de coletar e armazenar os dados pertinentes para elaboração de uma análise técnica do comportamento do inversor?

1.4 OBJETIVOS

1.4.1 Objetivo Geral

Implementar um sistema de medição capaz de aquisitar e armazenar grandezas elétricas de corrente alternada e contínua de um sistema de geração de energia fotovoltaica conectado à rede.

1.4.2 Objetivos Específicos

- Desenvolvimento de um sistema de medição de grandezas elétricas de corrente contínua;

- Prover armazenamento para as grandezas elétricas coletadas;
- Definição de uma forma de medição de grandezas elétricas de corrente alternada;
- Sincronizar as medições de corrente contínua e alternada para que possam ser analisadas.

1.5 JUSTIFICATIVA

Devido a pouca oferta de equipamentos direcionados para medição e aquisição de grandezas elétricas em corrente continua aplicados aos sistemas de geração de energia autônoma – fotovoltaico ou eólico – que pudessem ser instalados em uma aplicação de pequeno porte, em especial que possuísse um custo competitivo. Via de regra, os sistemas disponíveis estão prontos para a aquisição de grandezas em corrente alternada exclusivamente, impossibilitando a verificação da eficiência da conversão de energia realizada pelo inversor. Diante disso, a implementação de um sistema capaz de realizar esta função, torna-se interessante.

1.6 PROCEDIMENTOS METODOLÓGICOS

Este trabalho caracteriza-se como uma pesquisa aplicada, de campo e experimental, com um caso de estudo específico, um sistema de medição e armazenamento de grandezas elétricas de corrente contínua e alternada. De acordo com Gil (1995, p.58), “a pesquisa experimental consiste em determinar um objeto de estudo, selecionar as variáveis que seriam capazes de influenciá-lo, definir as formas de controle e de observação dos efeitos que a variável produz no objeto”.

As principais etapas desse trabalho são:

a) Revisão da literatura referente aos conceitos de medição de grandezas elétricas; uso de dispositivos de armazenamento destas grandezas; e sistemas fotovoltaicos conectados à rede de distribuição de energia;

b) Construção de um sistema de medição e armazenamento de grandezas elétricas em corrente contínua;
c) Escolha e aplicação de um equipamento comercial de medição de grandezas elétricas em corrente alternada capaz de armazenar ou disponibilizar para armazenamento externo;
d) Instalação dos sistemas de medição junto a um sistema fotovoltaico conectado à rede;
e) Análise dos dados coletados através da composição das grandezas medidas em corrente contínua e alternada em uma mesma estampa de tempo.

1.7 ESTRUTURA DO TRABALHO

Este trabalho é constituído de quatro capítulos:

No capítulo 1 é apresentada uma introdução, o tema de pesquisa, sua delimitação, objetivos geral e específicos, além de justificativas, problemas e premissas.

No capítulo 2 é feita uma revisão da literatura sobre os temas medição e armazenamento de grandezas elétricas e sistemas fotovoltaicos enfatizando os conectados à rede elétrica.

No capítulo 3 é apresentado o projeto do circuito de medição de corrente contínua bem como a instalação do equipamento de medição de corrente alternada e configuração do sistema de medição como um todo, considerando também a geração fotovoltaica e o inversor.

No capítulo 4 são apresentadas as conclusões e as considerações finais referentes ao sistema de medição, bem como as limitações encontradas e sugestões para próximos desenvolvimentos.

2. REVISÃO DA LITERATURA

2.1 MEDIÇÃO DE GRANDEZAS ELÉTRICAS

Um dos principais processos contidos no ciclo de fornecimento de energia elétrica é a medição das grandezas elétricas (ITO, 2003).

O papel exercido pelos equipamentos de medição é fundamental, pois permite não só o monitoramento e controle das grandezas elétricas (a fim de que as mesmas se encontrem em estado de funcionamento seguro), mas também permitem que a tarifação seja baseada no consumo real da energia por unidade consumidora (MIRANDA, 1987).

Para garantir a coerência e assertividade nas medições de energia em corrente contínua e alternada, têm-se estabelecidas normas brasileiras que trazem detalhes e diretrizes para a confecção de sistemas de medição de energia, instalação de sistemas de medição e até padrões de protocoloes de comunicação para medidores de grandezas elétricas. Um exemplo é a norma NBR 14519 (Medidores eletrônicos de energia elétrica — Especificação), que estabelece requisitos técnicos específicos para medidores de energia em corrente alternada (ABNT, 2000).

2.2 ARMAZENAMENTO DE DADOS

De acordo com Conceição (2015), os sistemas geralmente utilizados para armazenamento de dados são os *dataloggers,* que são equipamentos auxiliares para a coleta e armazenagem de dados de outros instrumentos. O funcionamento destes dispositivos se baseia em um sistema de contagem de pulsos eletrônicos, emitidos pelo instrumento ao qual se adaptou um *datalogge*r, sempre que um evento ou medição se repete. O Datalogger apresenta uma unidade de memória permitindo a retenção dessas informações e pode se comunicar com um computador através de programas adequados para este fim. As principais vantagens da aplicação desses dispositivos são:

- Expandem as possibilidades de monitoramentos ambientais com objetivos específicos;

- Os processos de obtenção e armazenamento de dados são mais rápidos e muitas vezes mais acurados do que os manuais;
- Obtenção de dados com maior frequência, o que melhora a qualidade dos dados obtidos.
- Possibilidade de transferência dos dados para um computador (CONCEIÇÃO, 2015).

Esser (2008) afirma que estes equipamentos têm a função de coletar dados em campo e armazená-los para posterior leitura e verificação, para conhecer o real funcionamento do equipamento que terá suas grandezas físicas medidas. O *dattalogger* possui entradas digitais, que tem a função de ler dados, por exemplo, ligado ou desligado, porta aberta ou fechada, válvula aberta ou fechada, isto é, qualquer sinal em forma digital "0" ou "1". Já suas entradas analógicas, possuem a função de ler sinais de tensão de 0 a 10V ou de corrente de 4 a 20mA, que podem representar grandezas físicas como tensão, pressão, vazão, nível, temperatura, umidade, etc.

Neste sentido, além da coleta de dados, é possível, através de duas saídas a relé, controlar algum equipamento. Esta função foi implementada para criar uma rotina que, caso ocorra alguma falha, permita a programação do coletor de dados proteger o equipamento monitorado caso ocorra um aumento súbito em alguma variável que esteja sendo monitorada (ESSER, 2008).

Conforme Esser (2008), o dispositivo permite a instalação de sensores capazes de realizar leitura de temperatura e umidade do ambiente onde estará instalado o coletor de dados, assim as entradas analógicas ficam livres para monitorar outras grandezas. Da mesma forma, é possivel programar a data e a hora exata em que a medição de uma determinada grandeza é realizada.

2.3 **SISTEMAS FOTOVOLTAICOS**

Os sistemas fotovoltaicos podem ser divididos em dois grupos: os sistemas isolados e os sistemas conectados à rede elétrica, conforme ilustrado na Figura 2.

Figura 2: Tipos de sistemas fotovoltaicos

Fonte: URBANETZ, 2010

2.3.1 SISTEMAS FOTOVOLTACOS ISOLADOS

Os sistemas fotovoltaicos isolados não possuem conexão com a rede elétrica das concessionárias de energia. Há diversas aplicações para sistemas isolados, elas variam desde pequenos equipamentos ou pequenos circuitos alimentados em corrente contínua até o fornecimento de energia elétrica para populações afastadas das redes elétricas das concessionárias. Módulos fotovoltaicos são apenas capazes de fornecer energia elétrica quando há irradiação solar incidindo sobre eles. Para alimentar cargas em momentos onde não há irradiação (por exemplo, durante a noite), utilizam-se acumuladores de energia, conforme mostrado na Figura 3 que representa a configuração de um sistema fotovoltaico isolado. Neste sentido, durante o dia, o módulo alimenta as cargas e carrega um banco de baterias. Durante a noite, as cargas são alimentadas pelo banco de baterias (GOETZBERGER e HOFFMANN, 2005).

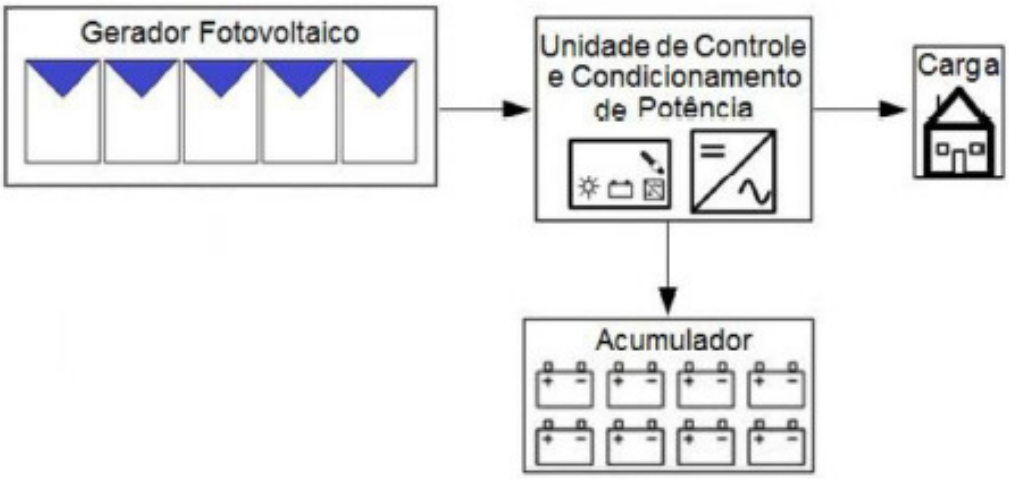


Figura 3: Configuração básica de um sistema fotovoltaico isolado

Fonte: PINHO e GALDINO, 2014

Para alimentar cargas convencionais independentemente da existência momentânea de irradiação solar sobre os módulos, um sistema isolado normalmente é composto por:

- Módulos ou Gerador Fotovoltaicos: os módulos devem ser instalados, preferencialmente, voltados na direção norte (se o sistema for instalado no hemisfério sul), com uma inclinação calculada para que a incidência de irradiação solar seja otimizada para os meses de menor irradiação solar e devem ser evitadas quaisquer sombras sobre os módulos;
- Inversor ou Condicionamento de Potência: é o equipamento que converte as grandezas contínuas fornecidas pelos módulos ou pelo banco de baterias em grandezas alternadas, necessárias para a alimentação da maioria das cargas;
- Banco de baterias ou Acumuladores: deve ter capacidade suficiente para suprir a demanda de energia durante os períodos onde não há irradiação solar suficiente para o funcionamento dos módulos;
- Controlador de carga ou Unidade de Controle: equipamento que monitora a tensão das baterias, garantindo que o carregamento seja feito conforme a curva de carga da bateria utilizada.

A principal vantagem da utilização de sistemas isolados é que, quando corretamente dimensionados e instalados, fornecem energia elétrica com uma alta confiabilidade, pois não estão sujeitos às falhas das redes de transmissão e distribuição da concessionária. O maior problema desse sistema é a incompatibilidade entre as vidas úteis de cada um dos equipamentos, principalmente entre os módulos fotovoltaicos (vida útil acima de 25 anos) e as baterias (cerca de 2 anos para baterias de chumbo ácido).

2.3.2 SISTEMAS FOTOVOLTAICOS CONECTADOS À REDE

A constituição básica de um sistema fotovoltaico conectado a rede é mais simples do que a de um sistema isolado. A confiabilidade do sistema está vinculada à confiabilidade da rede da concessionária, portanto não são necessários o banco de baterias nem o controlador de carga conforme mostrado na Figura 4.

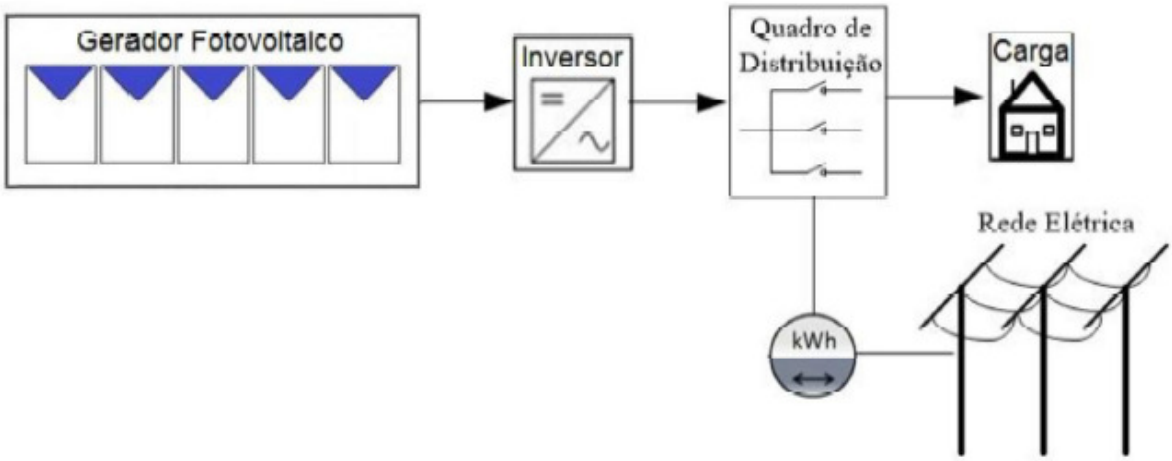


Figura 4: Configuração básica de um sistema fotovoltaico conectado à rede
Fonte: PINHO e GALDINO, 2014

Os sistemas conectados alimentam a rede com toda a energia disponibilizada pelos mesmos, garantindo uma alta produtividade. Outra vantagem desse tipo de sistema é a vida útil elevada devido à ausência de baterias (URBANETZ, 2010).

O fornecimento de energia para a rede elétrica e a proximidade entre a geração e o consumo, caracterizam esses sistemas como geração distribuída. Esse tipo de geração já é

muito utilizado em vários países, principalmente na Europa. Em alguns países já existem legislações específicas para este tema e há um incentivo para que esta seja feita com a utilização de fontes renováveis. Na Alemanha, por exemplo, foi estabelecida a *EEG - Erneuerbare Energie Gesetzt* (Lei de Energias Renováveis). Essa lei regulamenta e estabelece incentivos para o uso de energias renováveis. O proprietário de uma fonte alternativa de energia pode vender a energia gerada à concessionária. O preço é tabelado e varia de acordo com a fonte alternativa utilizada.

No Brasil a geração distribuída foi legalmente reconhecida pela primeira vez no dia 15 de março de 2004 com a Lei nº 10.848. Antes disso a geração distribuída não possuía regulamentação, sendo que esta foi definida primeiramente com o Decreto nº 5.163 de 30 de julho de 2004.

Sistemas de geração distribuída que utilizam fontes alternativas de energia apresentam diversas vantagens em relação ao sistema convencional de geração, entre elas:

- Economia com custos de transmissão e distribuição;
- O tempo de instalação é reduzido;
- Redução de impactos ambientais;
- Diversificação da matriz energética, minimizando a dependência de uma única fonte de energia;
- Viabilização do emprego em fontes renováveis de energia elétrica;
- Viabilização de unidades geradoras de pequeno porte.

No Brasil, a regulamentação mais recente referente aos sistemas de geração distribuida de micro e minigeração é a Resolução Normativa Aneel Nº 482/2012, e devem atender aos Procedimentos de Distribuição (PRODIST), Módulo 3, e às normas de acesso das distribuidoras locais. A Resolução 482 estabelece as condições gerais para o acesso de microgeração e minigeração distribuídas aos sistemas de distribuição de energia elétrica e o sistema de compensação de energia elétrica, cujas definições são:

- Microgeração distribuída: central geradora de energia elétrica, com potência instalada menorou igual a 100 kW e que utilize fontes com base em energia hidráulica, solar, eólica, biomassa ou cogeração qualificada,

conforme regulamentação da Aneel, conectada na redede distribuição por meio de instalações de unidades consumidoras.

- Minigeração distribuída: central geradora de energia elétrica, com potência instalada superiora 100 kW e menor ou igual a 1 MW para fontes com base em energia hidráulica, solar,eólica, biomassa ou cogeração qualificada, conforme regulamentação da Aneel, conectada na rede de distribuição por meio de instalações de unidades consumidoras.
- Sistema de compensação de energia elétrica: sistema no qual a energia ativa injetada por unidade consumidora com microgeração distribuída ou minigeração distribuída é cedida, pormeio de empréstimo gratuito, à distribuidora local e posteriormente compensada com o consumo de energia elétrica ativa dessa mesma unidade consumidora ou de outra unidade consumidora de mesma titularidade da unidade consumidora onde os créditos foram gerados, desde que possua o mesmo Cadastro de Pessoa Física (CPF) ou Cadastro de Pessoa Jurídica (CNPJ) junto ao Ministério da Fazenda.

Porém, para que sejam implantados sistemas desse tipo, algumas questões técnicas devem ser contempladas. Um sistema de geradores conectados à rede elétrica da concessionária não deve apresentar riscos à segurança da rede e dos demais consumidores, deve garantir a qualidade da energia elétrica fornecida, não deve comprometer o sistema de proteção da rede da concessionária e deve proteger a si mesmo em caso de falhas no funcionamento ou falhas externas.

Neste sentido, o equipamento que confere a parametrização de grandezas elétricas à rede elétrica é o inversor, equipamento responsável pela conversão da energia de corrente contínua (CC) dos módulos fotovoltaicos em energia de corrente alternada (CA) para alimentação de cargas. A energia CA na sua saída é colocada diretamente em paralelo com a rede elétrica convencional. Dessa forma, as duas ondas de tensão devem estar em fase e ter características elétricas bastante similares, de modo a possibilitar o paralelismo de geradores. Assim, a senoide produzida pelo inversor utiliza a onda da rede elétrica como referência, e havendo a referência, há geração FV. No caso de um desligamento da rede

elétrica convencional, o inversor do sistema FV também é desligado automaticamente, evitando o efeito do ilhamento (URBANETZ 2010).

O fenômeno de ilhamento ocorre quando há uma falta por parte da concessionária e o gerador distribuído continua a alimentar um ramo do sistema. Isso pode comprometer a segurança da equipe responsável pela manutenção da rede, uma vez que algumas áreas permanecem energizadas sem o conhecimento da concessionária. A qualidade da energia do ramo ilhado também estará fora do controle da concessionária e as proteções contra sobrecorrente da concessionária podem não atuar corretamente devido à mudança no nível de curto-circuito nesse ponto (PITOMBO, 2010).

Outro aspecto relevante em sistemas FV é o ponto de operação a que estão sujeitos os módulos FV. Eles possuem uma região de operação denominada de ponto de máxima potência (MPP), onde a potência fotogerada, que é o produto tensão x corrente, apresenta seu máximo valor. Esse valor varia continuamente em função da irradiância e da temperatura. Portanto, sistemas que também de modo contínuo busquem colocar os módulos FV para operar em MPP melhoram o desempenho do sistema FV. Nos SFVCR, o inversor normalmente já possui essa função incorporada (URBANETZ, 2010).

No que se refere à segurança, instalam-se dispositivos de proteção integrados aos equipamentos, a instalação de outros dispositivos de proteção deve ser prevista, como disjuntores, dispositivos de proteção contra surtos (DPS), sistemas de aterramento e sistemas de proteção contra descargas atmosféricas (SPDA) (PINHO; GALDINO, 2014, p. 246).

3. IMPLEMENTAÇÃO DO SISTEMA DE MEDIÇÃO DAS GRANDEZAS ELÉTRICAS

O sistema de medição foi instalado junto a um sistema fotovoltaico conectado à rede, a fim de ensaiar o sistema fotovoltaico e ao mesmo tempo comprovar a eficácia do sistema de medição proposto tanto para o lado de corrente contínua quanto para o lado de corrente alternada. A Figura 5 ilustra o esquema elétrico do sistema fotovoltaico com os dispositivos de medição.

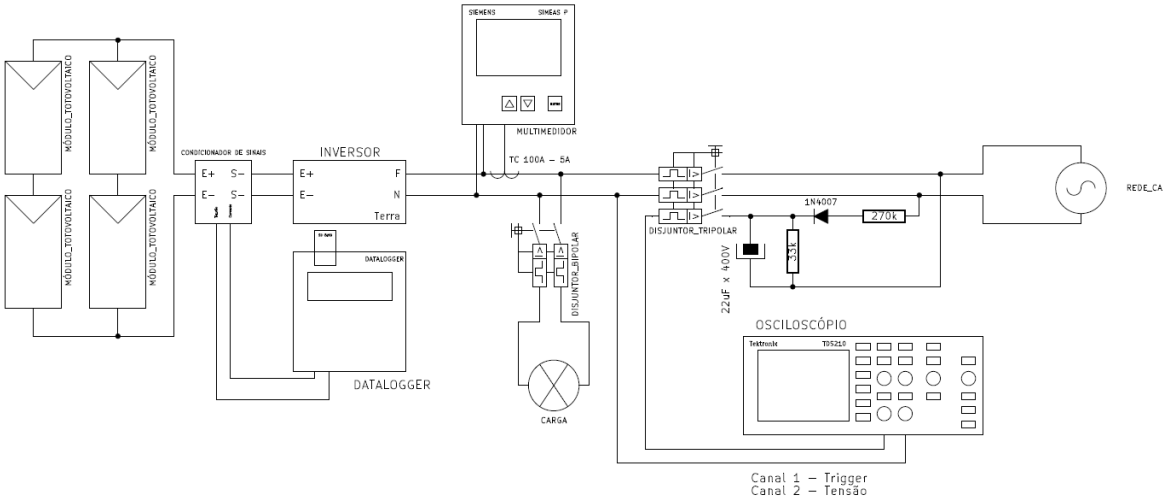


Figura 5: Esquema elétrico do sistema em teste

FONTE: LIPPMANN, 2015

3.1 SISTEMA DE AQUISIÇÃO E ARMAZENAMENTO DE DADOS.

O sistema de aquisição de dados pode ser dividido em três partes distintas:

- Medição do lado de corrente contínua;
- Medição do lado de corrente alternada;
- Aquisição de formas de onda do lado de corrente alternada.

Para as medições do lado de corrente contínua se faz necessário o uso de um dispositivo capaz de aquisitar e armazenar dados em memória não volátil, das grandezas presentes na parte de corrente contínua do sistema. Comercialmente são encontrados

diversos equipamentos aptos a realizar medições em corrente alternada, porém sistemas capazes de aquisitar grandezas em corrente contínua não são tão comuns.

Por este motivo, depois de buscas por equipamentos comerciais capazes de efetuar estas medições, por um custo acessível, optou-se pela construção de um sistema capaz de efetuar tais medições. Assim, para aquisitar os dados do lado de corrente contínua foi utilizados um *datalogger* de construção própria. A Figura 6 ilustra o diagrama eletrônico do sistema de aquisição de dados.

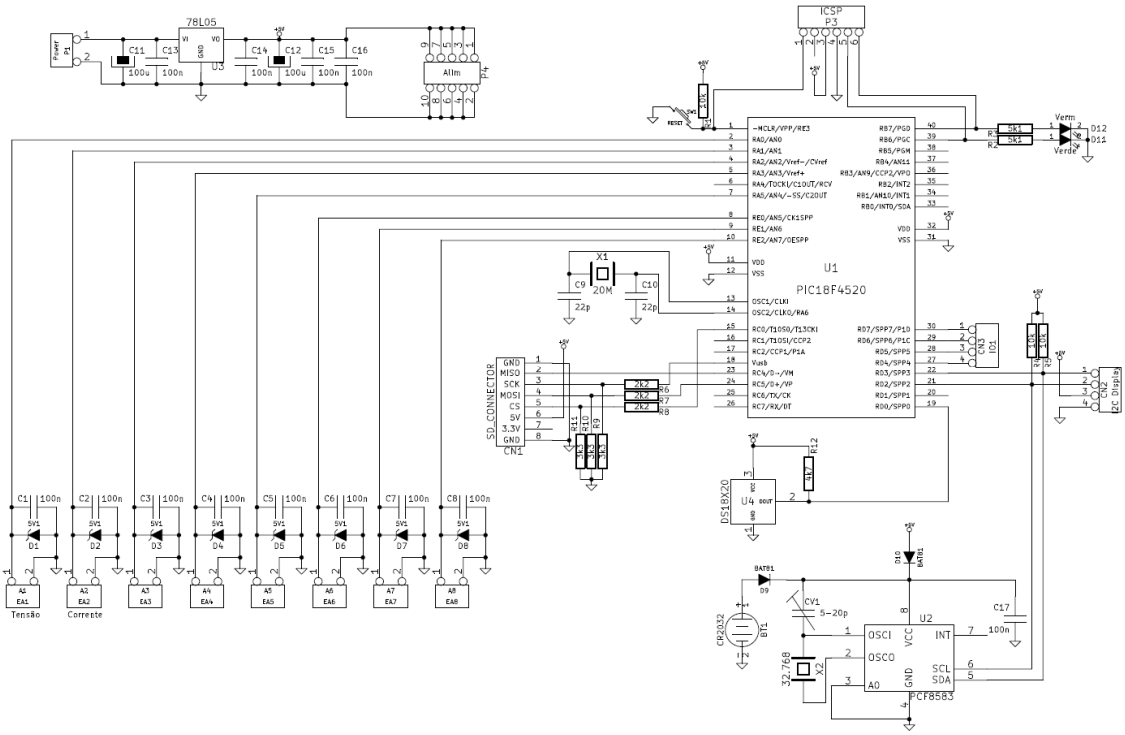


Figura 6: Esquema eletrônico do *datalogger*
FONTE: AUTORIA PRÓPRIA

Este *datalogger* foi baseado em um microcontrolador da família PIC, fabricado pela *Microchip Technology Inc.*, é de fácil aquisição no mercado nacional e bastante simples de ser manipulado. O modelo escolhido para esta função foi o PIC18F4520, que possui 36 portas de entrada/saída programáveis, das quais 13 podem ser utilizadas como entradas analógicas, utilizando o conversor analógico/digital interno de 10 *bits*. As demais portas podem ser utilizadas como entradas ou saídas digitais, livremente programadas.

Outra grande vantagem deste microcontrolador é sua memória de 32*kbytes*, permitindo realizar funções mais avançadas diretamente no microcontrolador.

Como o sistema deve armazenar os dados em memória não volátil, o *datalogger* está equipado com um cartão de memória tipo *Secure Digital*, bastante comum no mercado e de padrão aberto, o que facilita a sua integração. O cartão utilizado de 1*Gigabyte* tem capacidade de armazenamento para mais de 5.000 dias de aquisição de dados. Uma atenção especial precisou ser tomada com relação aos níveis de tensão do cartão de memória. Por padrão, os cartões de memória *Secure Digital* operam com tensão de 3,3V, enquanto o microcontrolador utilizado no *datalogger* opera em 5V. Por este motivo foi necessário utilizar divisores resistivos para adequar a tensão no cartão de memória, evitando que este pudesse ser definitavamente danificado. O módulo para cartão de memória, onde são armazenados os dados, é um modelo comercial, normalmente utilizado para sistemas da plataforma Arduino, porém pode ser utilizado com outros microcontroladores por ser apenas um módulo passivo.

Os dados não podem ser aleatoriamente aquisitados, pois deverão ser confrontados com os dados da medição do lado de corrente alternada, por isso se faz necessário uma referência temporal. Para esta referência temporal o *datalogger* está equipado com um relógio de tempo real baseado no circuito integrado PCF8583P, fabricado pela *NXP Semiconductors*, capaz de fornecer uma referência temporal completa, com dia, mês, ano, hora, minuto e segundo com extrema precisão, devido ao uso de um gerador de relógio interno, e com a vantagem de poder ser equipado com uma bateria para *backup* de força em caso de falta de alimentação no circuito do *datalogger*. Além disso, a comunicação entre o microcontrolador e o relógio de tempo real se dá por meio de um barramento com protocolo I^2C que utiliza apenas duas portas de entrada/saída do microcontrolador para interface entre o mestre e o escravo. Este protocolo utilizado pelo relógio de tempo real é bastante simples e possui fácil implementação.

Para realizar a função de interface homem-máquina o *datalogger* foi equipado com um *display* de cristal liquido de 4 linhas e 20 colunas. Normalmente a interface deste *display* com o microcontrolador é realizada por 11 ou 7 pinos de entrada/saída do microcontrolador, no modo paralelo. Existe, porém, uma forma de fazer a interface deste *display* utilizando um expansor de barramentos PCF8574, também fabricado pela *NXP*

Semiconductors, de modo que, através do protocolo de comunicação I^2C, apenas duas portas de entrada/saída do microcontrolador são necessárias para realizar esta interface. Como o *datalogger* já possui outro equipamento periférico se comunicando através deste mesmo protocolo I^2C, as mesmas portas de entrada/saída podem sem utilizadas para o *display*, otimizando os recursos disponíveis no microcontrolador. Neste display serão exibidas informações como a hora atual, o dia, mês e ano e também as tensões e correntes medidas no lado de corrente contínua, bem como a potência instantânea calculada a partir destas medições.

Como exitem portas disponíveis o datalogger foi equipado com um sensor de temperatura one-wire modelo DS18B20, fabricado pela *Maxim Integrated Products, Inc.* Este sensor é extremamente versátil e totalmente digital. Pode medir temperaturas entre -55°C e +125°C com exatidão de +- 0,5°C e possui resolução de 12 *bits*. Sendo totalmente digital são dispensadas calibrações e componentes externos para seu perfeito funcionamento. Outra vantagem é a necessidade de apenas uma porta de entrada/saída do microcontrolador para a interface entre o dispositivo mestre e o dispositivo escravo. O fato deste sensor ser encontrado em encapsulamento TO-92 facilita a elaboração de um sensor remoto para a aquisição de temperatura e, por não haverem sinais analógicos envolvidos na medição, a distância entre o dispositivo mestre e o dispositivo escravo pode ser razoavelmente longa, da ordem de 5 a 10 metros, sem a preocupação com interferências. A Figura 7 apresenta a imagem do protótipo de aquisição de dados implentado.

Figura 7: *Datalogger* implementado
FONTE: AUTORIA PRÓPRIA

O código fonte do *firmware* do *datalogger* encontra-se no anexo 1.

Como o microcontrolador utilizado para a confecção do *datalogger* consegue manusear tensão de, no máximo, 5V, foi necessário construir um condicionador de sinais, de forma a adequar as medidas de tensão e corrente para os níveis necessários ao microcontrolador responsável pela coleta dos dados.

O condicionador de sinais possui 3 partes disitintas, a primeira parte é um gerador de tensão negativa. Esta tensão negativa é um requisito básico para o perfeito fucnionamento dos amplificadores operacionais e de instrumentação que serão utilizados nas medições de tensão e corrente. Este gerador de tensão negativa foi baseado no circuito integrado ICL7660, também fabricado pela *Maxim Integrated Products, Inc.*, e facilmente encontrado no comercio local.

Para medição da corrente fornecida pelo sistema fotovoltaico, no lado de corrente contínua, será utilizado um resistor *shunt* que apresentará uma queda de tensão proporcional a corrente que circula por seu elemento resistivo. Como as correntes envolvidas são altas, da ordem de dezenas de Ampères, a resistência deste resistor *shunt* deve ser a menor posível para evitar perdas por dissipação de calor e também para não introduzir quedas de tensão indesejáveis ao sistema. Por esta característica, pela lei de Ohm, tendo-se uma baixa resistência haverá uma baixa diferença de potencial. Embora o microcontrolador possa ler valores baixos de tensão a precisão será muito baixa, pois estará operando apenas no começo da escala. O ideal é que seja fornecido ao microcontrolador uma tensão próxima ao fundo de escala, onde é possível fazer a medição com melhor precisão.

Para fazer esta adaptação o ideal é utilizar um amplificador de instrumentação, devido ao baixo nível de ruído e ao mesmo apresentar entrada em modo diferencial, o que dispensa a necessidade de referência à terra, além de necessitar apenas de um único reisistor externo para a operação. O modelo escolhido foi o AD620, já disponível pelo autor, e fabricado pela *Analog Devices, Inc.*

O *shunt* disponível possui a relação 50A – 75mV, ou seja, apresenta uma queda de tensão de 75mV quando submetido a uma corrente de 50A. Para que seja aproveitada toda a escala de medição, o amplificador de instrumentação deverá ser configurado de forma a entregar 5V em sua saída quando aplicados 75mV em sua entrada.

A equação de ganho do amplificador de instrumentação em questão é dada por:

$$G = \frac{49{,}4k\Omega}{R_G} + 1 \text{ ou } R_G = \frac{49{,}4k\Omega}{G-1}$$

Sabe-se que o ganho deverá ser de 66,66667 unidades (5/0.075), logo tem-se:

$$R_G = \frac{49{,}4k\Omega}{66{,}6667 - 1} = \frac{49{,}4k\Omega}{65{,}6667} = 0{,}752282\ k\Omega$$

Ou o valor comercial mais próximo, 750Ω.

A tensão recebida dos módulos fotovoltaicos normalmente é superior a 5V, de forma que não será necessário amplificá-la, mas sim, reduzi-la a níveis que possam ser aquisitados corretamente pelo microcontrolador. Esta operação pode ser feita por um simples divisor resistivo. Considerando uma tensão máxima na entrada de 50V e a tensão máxima que pode ser manipulada pelo microcontroaldor, de 5V, tem-se que o divisor resistivo deverá ter uma proporção de 10 vezes.

$$\frac{V_{Ent}}{R_1 + R_2} = \frac{V_{Sai}}{R_2} \rightarrow \frac{V_{Ent}}{V_{Sai}} = \frac{R_1 + R_2}{R_2} \rightarrow \frac{50}{5} = \frac{R_1 + R_2}{R_2} \rightarrow 10 = \frac{R_1 + R_2}{R_2} \rightarrow R_1 = 9R_2$$

Utilizando um resistor de valor comercial para R_2 pode-se chegar ao valor de R_1 por associações de valores comerciais. Definido o valor de 10kΩ para R_2, tem-se que R_1 deverá ter 90kΩ, o que pode ser conseguido pela associação em paralelo de dois resistores de 180kΩ.

Para evitar perda de carga e alteração da impedância do divisor resistivo foi colocado um amplificador operacional tipo OP07, fabricado pela *Texas Instruments*, apenas para prover a isolação entre a medição e o microcontrolador. Este amplificador operacional está montado na configuração amplificador não inversor com ganho = 1. A Figura 8 ilustra o diagrama eletrônico do condicionador de sinais.

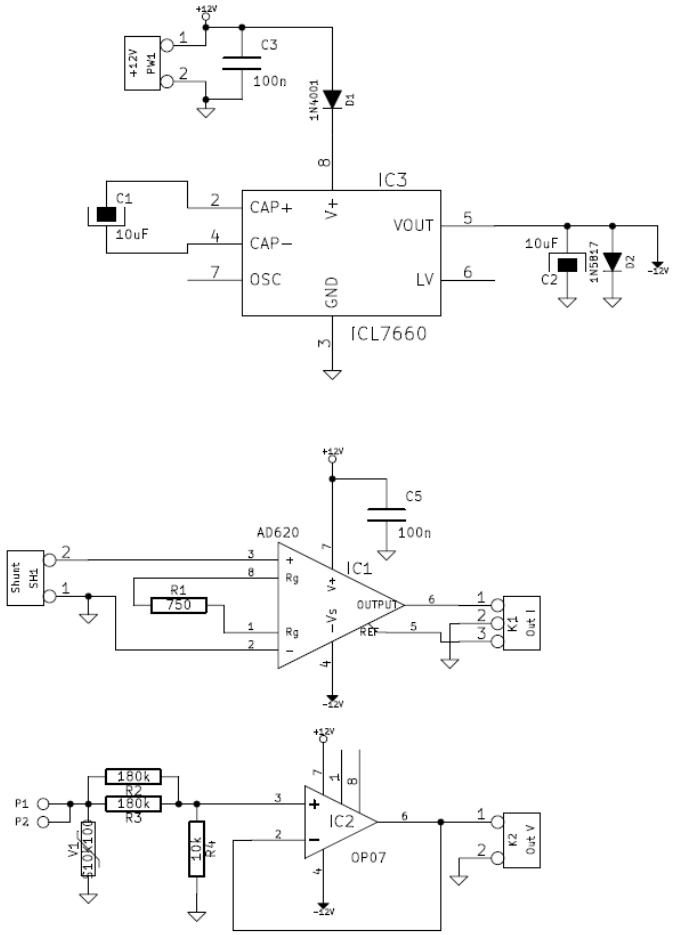


Figura 8: Esquema eletrônico do condicionador de sinais

FONTE: AUTORIA PRÓPRIA

A opção de montar o condicionador de sinais em uma placa separada foi para universalizar suas aplicações futuras. Caso o mesmo fosse integrado o *datalogger* serviria apenas para esta aplicação.

O *datalogger* possui um relógio de tempo real alimentado por bateria, o que mantém seu horário mesmo quando desligado. A Figura 9 apresenta a imagem do condicionador de sinais implementado.

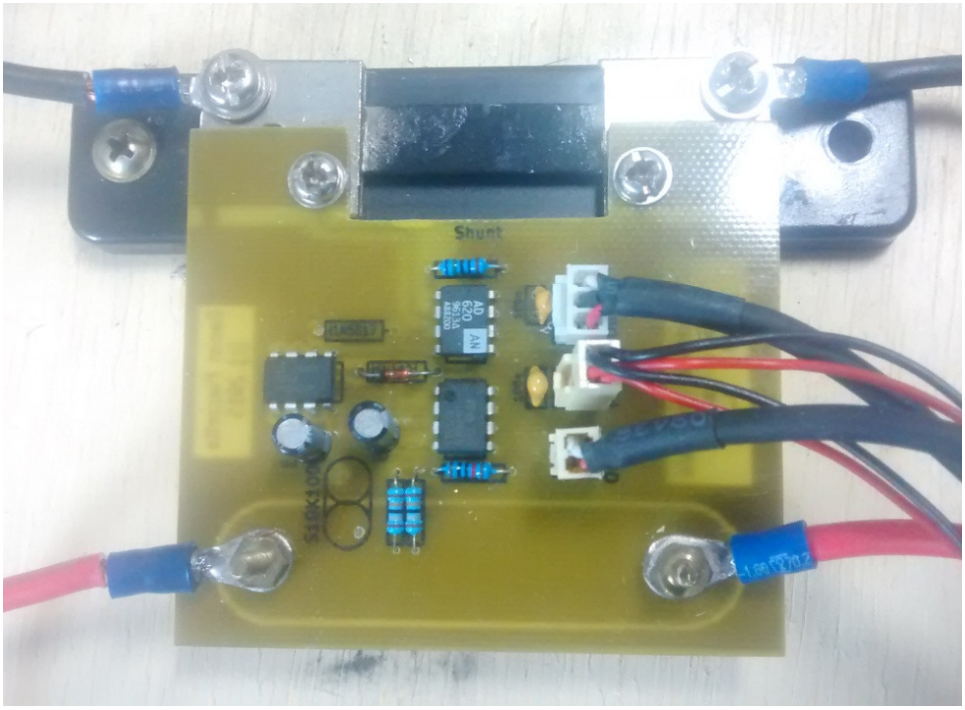


Figura 9: Montagem do condicionador de sinais
FONTE: AUTORIA PRÓPRIA

A placa do condicionador de sinais foi pensada de forma tal que o resistor *shunt* para medida de corrente ficasse integrado a mesma, evitando possíveis perdas numa fiação entre este resistor e a placa. Além disso, o fato do resistor *shunt* estar integrado a placa facilita a sua fixação, pois o resistor *shunt* possui base de montagem.

Para as medições do lado de corrente alternada – lado conectado à rede – foi utilizado um equipamento comercial, pois devido a disponibilidade e custo do mesmo, o desenvolvimento de um sistema para aquisição destes dados seria economicamente inviável. Devido a disponibilidade de um equipamento SIMEAS P600, fabricado pela *Siemens*, o mesmo foi utilizado. Este equipamento é um multimedidor de grandezas em corrente alternada, desenvolvido especificamente para medição de tensão e correntes em sistemas trifásicos, mas que pode ser configurado para operar em sistemas monofásicos. Este equipamento pode medir tensões alternadas até um fundo de escala configurável de 480V e correntes até 6A. Entretanto, ele não possui armazenamento em memória não volátil, o que obriga a dispor de um meio de coleta e armazenamento dos dados medidos.

O medidor disponível possui uma interface de comunicação serial em padrão RS485 que disponibiliza os dados medidos através do protocolo de comunicação MODBUS. Como a corrente teórica máxima que pode ser fornecida pelo inversor é próxima ao fundo de escala do medidor, é prudente prover o sistema com um transformador de corrente de forma a, além de isolar galvanicamente os circuitos, permitir manusear correntes de maior magnitude do que o fundo de escala permitido pelo equipamento de medição.

Para aquisição dos dados medidos e posterior armazenamento utilizou-se do *software* ScadaBR, um servidor SCADA gratuito capaz de coletar os dados no protocolo MODBUS e armazená-los em um banco de dados para serem exportados posteriormente para análise, tratamento e comparações.

Para facilitar os testes o sistema foi montado em uma *work board*, de forma que todos os componentes ficassem próximos e fáceis de serem acessados. A Figura 10 ilustra a disposição dos equipamentos de medição e aquisição de dados junto ao inversor do sistemas fotovoltaico.

Figura 10: Montagem do sistema

FONTE: LIPPMANN, 2015

Além do *datalogger* e do multimedidor, o sistema também é composto de disjuntores de proteção e manobra, cargas de teste e um osciloscópio, para a coleta das formas de onda no lado de corrente alternada.

Para a aquisição das grandezas elétricas no lado de corrente alternada utilizou-se um multimedidor SICAM SIMEAS P600 de fabricação da Siemens. Este multimedidor disponibiliza os dados em uma porta serial RS485 em protocolo MODBUS. Para a aquisição destes dados foi utilizado um computador com o aplicativo gratuito ScadaBR, que é um servidor SCADA capaz de adquirir os dados através de uma porta serial e guardá-los em um banco de dados para posteriormente gerar relatórios das medições.

3.2 **RESULTADOS OBTIDOS**

3.2.1 Medições dos valores de geração lado CC

O Gráfico 1 mostra o resultado de medições de corrente efetuadas no lado de corrente contínua do sistema.

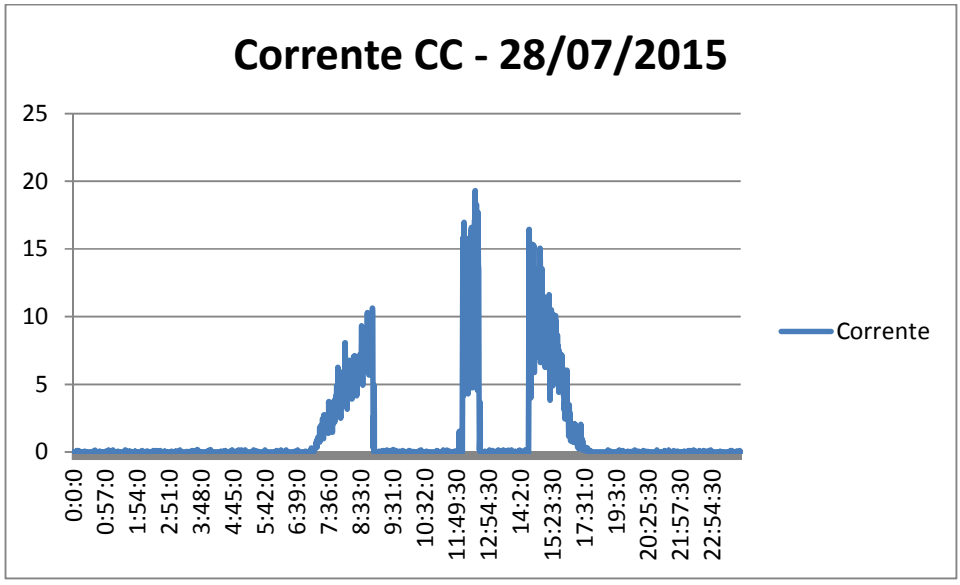


Gráfico 1: Corrente no Lado CC medida no dia 28/07/2015

FONTE: AUTORIA PRÓPRIA

Verifica-se no Gráfico 1 descontinuides na corrente drenada dos painéis fotovoltaicos devido a desligamentos no inversor ocorridos neste dia. Estes desligamentos não tiveram suas causas determinadas visto que a rede elétrica no lado de corrente alternada manteve-se normal.

O Gráfico 2 mostra a tensão medida no lado de corrente contínua. Os dados, bem como as estampas de tempo são armazenados no datalogger e podem ser exportados para análise.

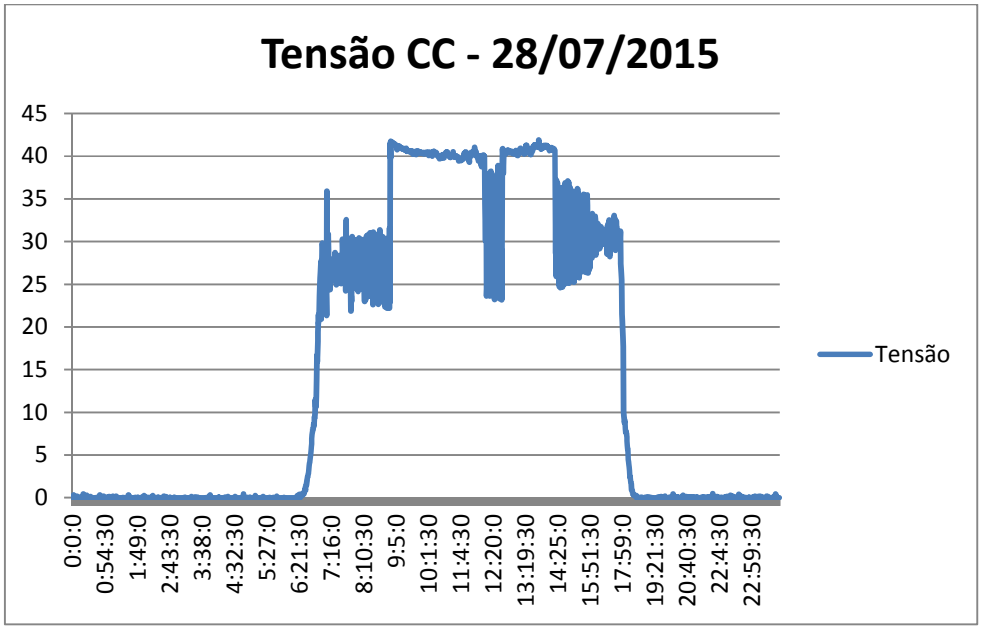


Gráfico 2: Tensão no Lado CC medida no dia 28/07/2015

FONTE: AUTORIA PRÓPRIA

Nos momentos em que houve o desligamento por parte do inversor observa-se que a tensão no lado de corrente contínua é a tensão de circuito aberto do painel fotovoltaico.

O Gráfico 3 apresenta a composição da potência no lado de corrente contínua, obtida a partir do produto da corrente CC com a tensão CC apresentados nos gráficos anteriores.

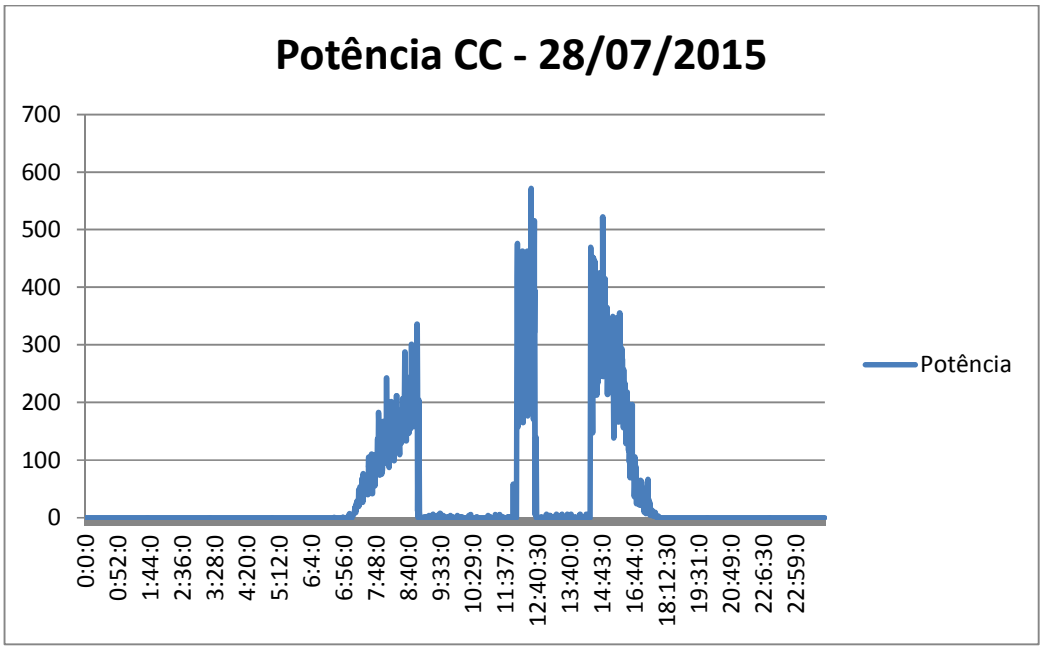


Gráfico 3: Potência no Lado CC medida no dia 28/07/2015

FONTE: AUTORIA PRÓPRIA

Como a potência é a resultante do produto tensão x corrente, nos momentos em que não houve corrente drenada do painel fotovoltaico, oberva-se também a inexistência de potência nestes intervalos.

Um sensor de temperatura também foi utilizado no sistema de medição, junto ao inversor, e teve seus dados armazenados pelo circuito do *datalogger* conforme Gráfico 4.

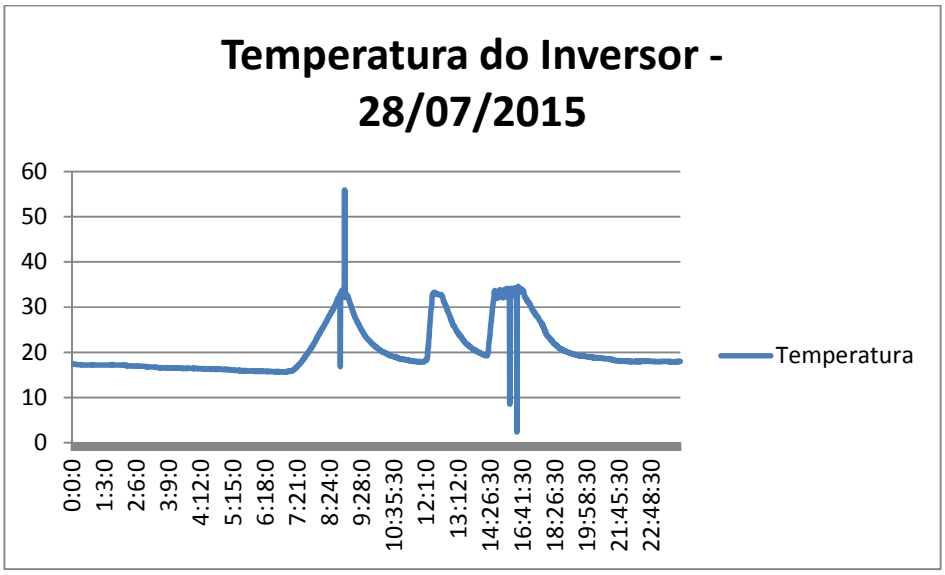


Gráfico 4: Temperatura do inversor medida no dia 28/07/2015

FONTE: AUTORIA PRÓPRIA

A temperatura aumenta gradativamente e, nos momentos onde há o desligamento do sistema, ela é reduzida. Existem alguns pontos com erros de medição causados, provavelmente, por problemas de temporização na conversão analógico-digital do sensor de temperatura.

Foram realizadas novas medições visando obter resultados sem a interferência dos desligamentos observados nos gráficos anteriores. Assim, foram realizadas medições no dia 29/07/2015 e apresentadas nos gráficos a seguir. O Gráfico 5 ilustra o comportamento da corrente no lado CC drenada do painel fotovoltaico.

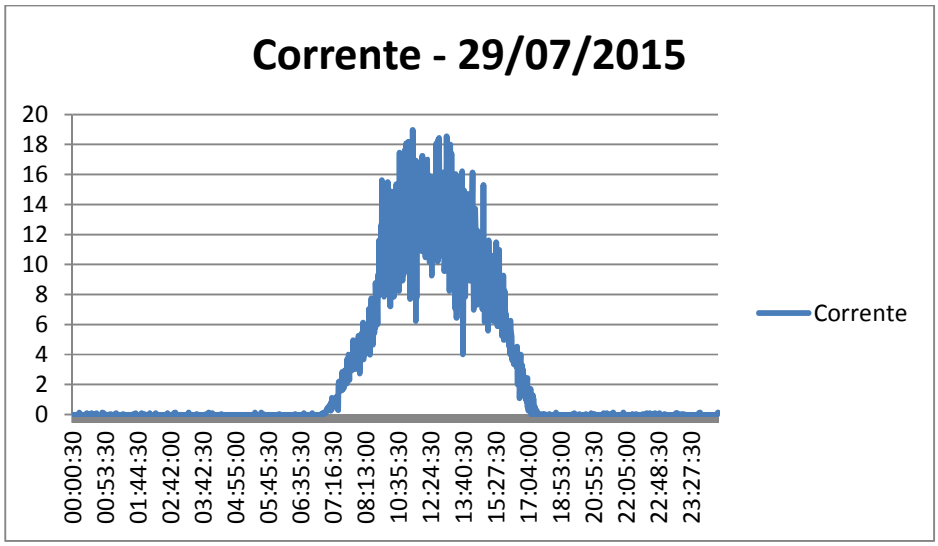


Gráfico 5: Corrente no Lado CC medida no dia 29/07/2015

FONTE: AUTORIA PRÓPRIA

Neste dia, devido a incidência de nuvens, houve uma sensível variação na corrente entregue pelo painel fotovoltaico, porém não foram observadas descontinuides quanto ao funcionamento do inversor. O Gráfico 6 mostra a tensão no lado CC.

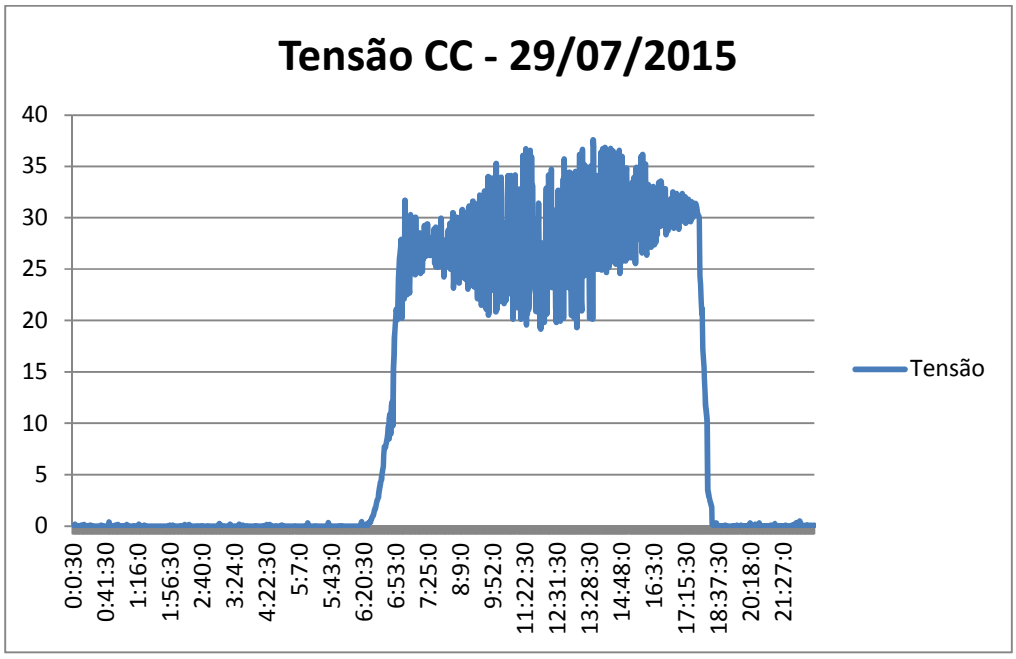


Gráfico 6: Tensão no Lado CC medida no dia 29/07/2015

FONTE: AUTORIA PRÓPRIA

Observa-se nas tensões medidas no lado CC que o painel fotovoltaico permaneceu conectado ao inversor e fornecendo energia.

O Gráfico 7 ilustra a potência drenada do painel fotovoltaico no lado de corrente contínua.

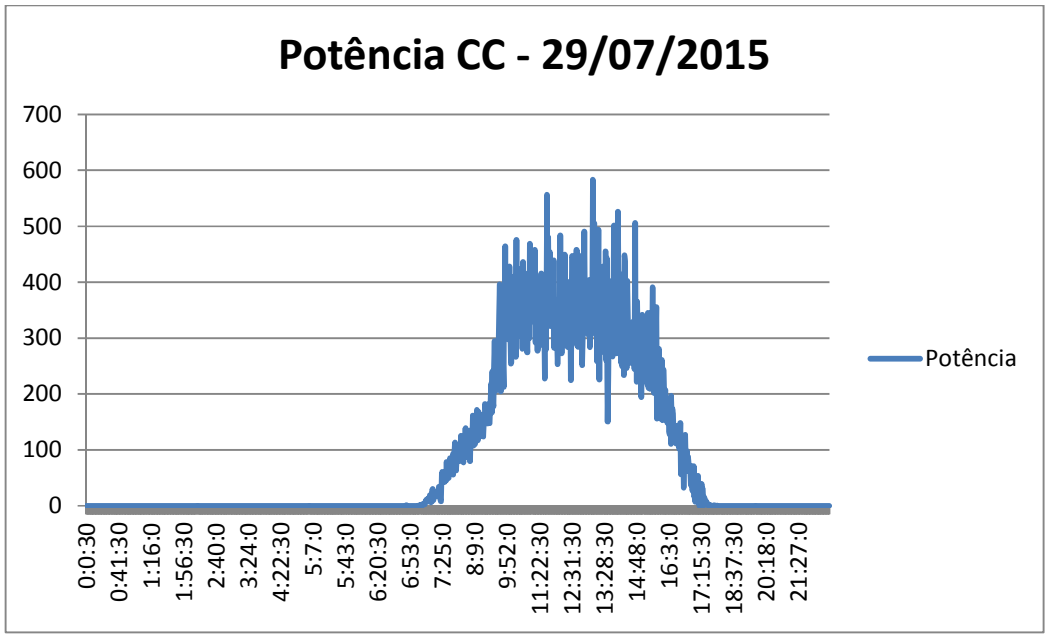


Gráfico 7: Potência no Lado CC medida no dia 29/07/2015

FONTE: AUTORIA PRÓPRIA

Como já mencionado anteriormente o produto da tensão e corrente fornecidos pelo painel fotovoltaico representam a potência elétrica disponibilizada pelo mesmo neste dia.

O Gráfico 8 ilustra o comportamento da temperatura no inversor ao longo do dia.

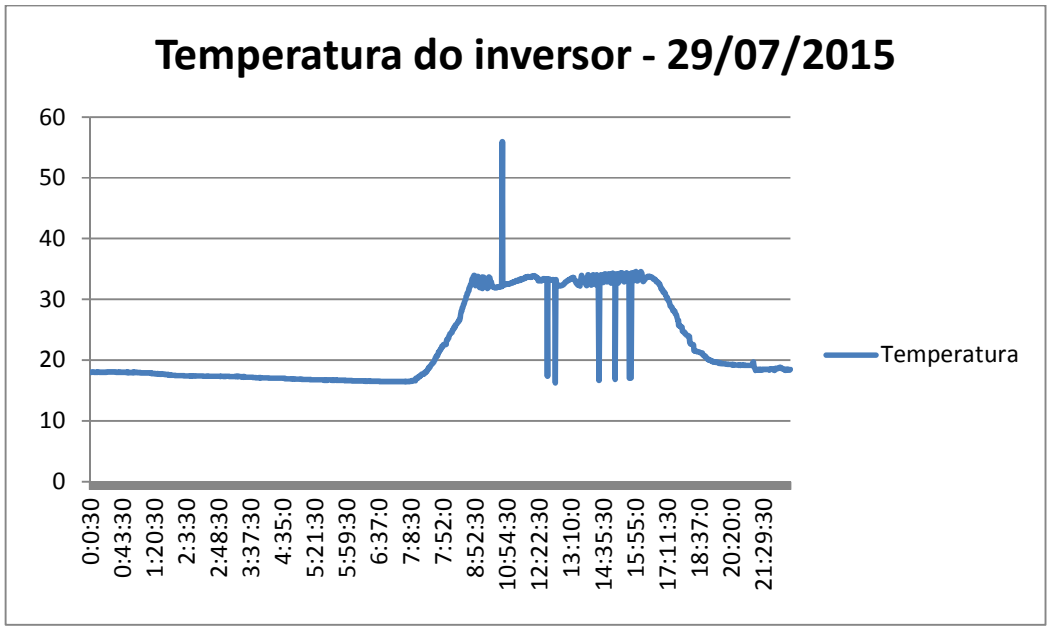


Gráfico 8: Temperatura do Inversor medida no dia 29/07/2015

FONTE: AUTORIA PRÓPRIA

Percebe-se um aumento gradual da temperatura no início da geração e sua estabilização ao longo do dia, com pequenas variações devido a atuação do ventilador interno. Ao final da geração pode-se notar a redução gradativa da mesma até atingir a temperatura ambiente.

4. **CONCLUSÕES FINAIS**

O sistema apresentado se mostrou bastante robusto e capaz de aquisitar todas as grandezas necessárias as análises de desenpenho de um sistema de geração autônoma.

Um ponto negativo do sistema apresentado foi o fato dos dados ficarem armazenados em locais diferentes – Cartão de memória para o *datalogger* de medidas do lado de corrente contínua e disco rígido do supervisório ScadaBR para as medidas do lado de corrente alternada. Embora ambas as medidas fossem aquisitadas e armazenadas em conjunto com a estampa de tempo com que foram aquisitadas, pode haver um deslocamento nos relógiso de tempo real, além do que os mesmos precisam estar em perfeito ajuste para garantir uma precisão maior nas análises.

Outra situação verificada foi a diferença das medições em corrente contínua entre duas amostras consecutivas, por algumas ocasiões. Esta variação pode estar relacionada a ruídos introduzidos no sistema pelo ambiente onde estava instalado, em especial ruídos provenientes do chaveamento realizado pelo inversor. Um filtro passa baixas na interface entre o condicionador de sinais pode amenizar sensivemente esta situação, bem como a implementação de rotinas de média aritmética das últimas medições para evitar valores muito discrepantes.

O desenvolvimento do *datalloger* previa a gravação de um arquivo, em formato texto, por dia de aquisição de dados. O nome do arquivo era tomado pelo dia, mês e ano vigente, recebidos do relógio de tempo real e, na ocorrência da mudança de dia – 0h – o novo arquivo deveria ser criado. Em algumas situações esta criação não aconteceu. Acredita-se que, pelas rotinas de gravação no cartão estarem muito demoradas, o processador não percebeu o horário 00:00:00 que dispararia a criação do novo arquivo, passando a armazenar os dados do novo dia no arquivo do dia anterior. Este fato não constitui um problema crítico, pois as medidas são armazenadas com a indicação do dia e horário de aquisição.

A utilização de um sistema de aquisição e armazenamento das grandezas elétricas no lado de corrente contínua permitiu identificar momentos de desligamento do inversor sem causa aparente. Estes desligamentos afetam diretamente o desempenho global do

sistema fotovoltaico e sua observação só foi possível devido a presença do sistema de monitoramento implementado.

São sugestões para futuros trabalhos:

Aprimoramento do *firmware* do *datalogger*, pois o mesmo apresentou instabilidade no funcionamento, em especial o atraso na gravação de dados no cartão de memória;

Melhoria nas rotinas de medição de tensão e corrente no lado de corrente contínua, pois hove muita variação entre uma medida e a medida anterior. Um algoritmo de suavização através das médias das últimas 10 medidas pode ser uma boa solução.

Implementação de um *driver* MODBUS escravo para disponibilzação dos dados do *datalloger* ao mesmo equipamento que está aquisitando os dados do lado de corrente alternada.

Implementação de um *driver* MODBUS mestre para aquisitar os dados do lado de corrente alternada e fazer o armazenamento em um local único, com a mesma estampa de tempo.

5. **REFERÊNCIAS**

ANEEL Agência Nacional de Energia Elétrica. **Resolução Normativa nº 482/2012**. 17 de abril de 2012.

ABNT. Associação Brasileira de Normas Técnicas. **NBR 14519: Medidores Eletrônicos de Energia Elétrica - Especificação**. Rio de Janeiro, 2000.

BEN - **Balanço Energético Nacional,** 2015 (Ano base 2014) – Disponível em < https://ben.epe.gov.br/downloads/Relatorio_Final_BEN_2015.pdf>. Acesso em: 6 jul. 2015

CONCEIÇÃO, M. N. da. **Tecnologia X Precisão**. Disponivel em: < www.leb.esalq.usp.br/aulas/lce5702/tecnologiaxprecisao.ppt>. Acesso em 14 de setembro de 2015.

EPRI – Electric Power Research Institute; **Electricity Use and Delivery**. Disponível em: www.epri.com. Aceso em 25 de julho de 2015.

ESSER, J. C.. **Data Logger – Coletor de Dados**. 2008 (Monografia de Especialização). Programa de Pós-graduação em Desenvolvimento de Produtos Eletrônicos. IFSC, Florianópolis.

GIL, A. C. 1995. **Como Elaborar Projetos de Pesquisa**. São Paulo: Editora Atlas.

GOETZBERGER, A.; HOFFMANN V. U. **Photovoltaic Solar Energy Generation**. Berlin: Springer, 2005.

ITO, H.T. **Energia Elétrica: Apuração da Qualidade dos Dados de Consumo**. 2003. Dissertação (Mestrado) - Unicamp, Campinas.

LIPPMANN, F. C. **Análise do Comportamento de um Inversor de Baixo Custo para Sistemas Conectados à Rede** (Trabalho de Conclusão de Curso). Programa de Pós-graduação em Energias Renováveis. Universidade Tecnológica Federal do Paraná – UTFPR. Curitiba, 2015.

MIRANDA, M.D.; **Um medidor digital de energia elétrica.** 1987. Dissertação (Mestrado) - Universidade Federal de Santa Catarina, Florianópolis.

PINHO, João T.; GALDINO, Marco A.; Grupo de Trabalho de Energia Solar (GTES). CEPEL - GTES. **Manual de engenharia para sistemas fotovoltaicos**. Rio de Janeiro, 2014, 529 p.

PITOMBO, S.O. **Proteção Adaptativa Anti-Ilhamento de Geradores Síncronos Distribuídos** (Mestrado em Ciências). Programa de Engenharia Elétrica: Sistemas Elétricos de Potência, Universidade de São Paulo, São Carlos – SP, 2010.

URBANETZ JR, J. **Sistemas Fotovoltaicos Conectados a Redes de Distribuição Urbanas: Sua Influência na Qualidade da Energia Elétrica e Análise dos Parâmetros que possam afetar a Conectividade** (Tese de Doutorado). Programa de Pós Graduação em Engenharia Civil. Universidade Federal de Santa Catarina - UFSC. Florianópolis, 2010.

Anexo 1 – Código Fonte do *Datalogger*

```
//firmware.C

#define _LCD_FIRST_LINE 0x80
#define _LCD_SECOND_LINE 0xC0
#define _LCD_THIRD_LINE 0x94
#define _LCD_FOURTH_LINE 0xD4

#define RED_LED   RB2_bit
#define GREEN_LED RB3_bit

#include "Hw_Def.h"
#include "LCD_I2C_DRIVER.h"
#include "PCF8583.h"
#include "SD_Card.h"


unsigned long Vp, Ip, Pp, EA3, EA4, EA5, EA6, EA7, EA8; // Variáveis para leitura
das grandezas analógicas
unsigned Tp=0, Ti=0;                        // Variáveis para leitura de temperatura
float Ppt;

void Display_Time()
{
char txt[11];
   // output values to LCD display
   txt[0] = (TimeRead.day / 10) + 48;
   txt[1] = (TimeRead.day % 10) + 48;
   txt[2] = '/';
   txt[3] = (TimeRead.month / 10) + 48;
   txt[4] = (TimeRead.month % 10) + 48;
   txt[5] = '/';
   txt[6] = ((TimeRead.year % 100) / 10) + 48;
   txt[7] = (TimeRead.year % 10)  + 48;
   txt[8] = 0;  // null to terminate the string
   LCD_I2C_Send_Byte(0x00,_LCD_FIRST_LINE+1);   // Cursor OFF
   LCD_I2C_Write(txt);

   txt[0] = (TimeRead.hours / 10) + 48;
   txt[1] = (TimeRead.hours % 10) + 48;
   txt[2] = ':';
   txt[3] = (TimeRead.minutes / 10) + 48;
   txt[4] = (TimeRead.minutes % 10) + 48;
   txt[5] = ':';
   txt[6] = (TimeRead.seconds / 10) + 48;
   txt[7] = (TimeRead.seconds % 10) + 48;
```

```
   txt[8] = 0;  // null to terminate the string
   LCD_I2C_Send_Byte(0x00,_LCD_FIRST_LINE+10);   // Cursor OFF
   LCD_I2C_Write(txt);
}

void Display_Temperature(void)
{
     unsigned short RES_SHIFT = 4;
     char temp_whole;
     unsigned int temp_fraction;

     char texto[20];

     // escreve no display a mascara   "Tp  .  °C Ti  .  °C"
     LCD_I2C_Send_Byte(0x00,_LCD_SECOND_LINE);   // Posiciona na segunda linha
     texto[0] = 'T';
     texto[1] = 'p';
     texto[2] = ' ';
     texto[3] = ' ';
     texto[4] = '.';
     texto[5] = ' ';
     texto[6] = ' ';
     texto[7] = 223;
     texto[8] = 'C';
     texto[9] = ' ';
     texto[10] = 'T';
     texto[11] = 'i';
     texto[12] = ' ';
     texto[13] = ' ';
     texto[14] = '.';
     texto[15] = ' ';
     texto[16] = ' ';
     texto[17] = 223;
     texto[18] = 'C';
     texto[19] = 0;
     LCD_I2C_Write(texto);

     // Conversao e apresentacao dos valores de temperatura do painel fotovoltaico
     temp_whole = Tp >> RES_SHIFT ;                  // parte inteira
     texto[0] = (temp_whole/10)%10 + 48;               // extrai as dezenas para
mostrar no display
     log_string[28] =  (temp_whole/10)%10 + 48;                     // faz o mesmo
para colocar na strig que vai ser escrita no LOG
     texto[1] =  temp_whole%10     + 48;              // extrai as unidades para
mostrar no display
     log_string[29] =  temp_whole%10     + 48;                     // faz o mesmo
para colocar na strig que vai ser escrita no LOG
     texto[2] =  '.';
```

```
            // Extract temp_fraction and convert it to unsigned int
            temp_fraction  = Tp << (4-RES_SHIFT);
            temp_fraction &= 0x000F;
            temp_fraction *= 625;

            // Converte a parte fracionaria em caracteres
            texto[3] =  temp_fraction/1000     + 48;          // extrai os milhares para
mostrar no display
            log_string[31] =  temp_fraction/1000     + 48;  // e coloca na strig que vai
ser escrita no LOG
            texto[4] = (temp_fraction/100)%10 + 48;          // extrai as centenas para
mostrar no display
            log_string[32] = (temp_fraction/100)%10 + 48;  // e coloca na strig que vai
ser escrita no LOG
            texto[5] = 0;

            // Escreve a temperatura do painel fotovoltaico no display
            LCD_I2C_Send_Byte(0x00,_LCD_SECOND_LINE+2);    // Posiciona na segunda linha,
3o caracter
            LCD_I2C_Write(texto);

            temp_whole = Ti >> RES_SHIFT ;
            texto[0] = (temp_whole/10)%10 + 48;                // extrai as dezenas para
mostrar no display
            log_string[34] = (temp_whole/10)%10 + 48;       // e coloca na strig que vai
ser escrita no LOG
            texto[1] =  temp_whole%10      + 48;               // extrai as unidades para
mostrar no display
            log_string[35] =  temp_whole%10      + 48;       // e coloca na strig que vai
ser escrita no LOG
            texto[2] =  '.';

            // Extract temp_fraction and convert it to unsigned int
            temp_fraction  = Ti << (4-RES_SHIFT);
            temp_fraction &= 0x000F;
            temp_fraction *= 625;

            // Converte a parte fracionaria em caracteres
            texto[3] =  temp_fraction/1000     + 48;          // extrai os milhares para
mostrar no display
            log_string[37] = temp_fraction/1000     + 48;    // e coloca na strig que vai
ser escrita no LOG
            texto[4] = (temp_fraction/100)%10 + 48;          // extrai as centenas para
mostrar no display
            log_string[38] = (temp_fraction/100)%10 + 48;  // e coloca na strig que vai
ser escrita no LOG
            texto[5] = 0;
```

```
            // Escreve a temperatura do inversor no display
            LCD_I2C_Send_Byte(0x00,_LCD_SECOND_LINE+12);    // Posiciona na segunda linha,
13o caracter
            LCD_I2C_Write(texto);
            }

        void Display_Analog(void)
        {
            char txt[6];

            // Coloca no display o valor da tensao do painel fotovoltaico na terceira
linha, a partir do quarto caracter.
            txt[0] =  (Vp / 1000) + 48;
            txt[1] = ((Vp % 1000) / 100 ) + 48;
            txt[2] = '.';
            txt[3] = ((Vp %  100) / 10  ) + 48;
            txt[4] = ( Vp %   10) + 48;
            txt[5] = 0;                                  // Caracter "null" para finalizar
a string
            LCD_I2C_Send_Byte(0x00,_LCD_THIRD_LINE+3);       //  Terceira  linha,  quarto
caracter
            LCD_I2C_Write(txt);

            // Coloca no display o valor da corrente do painel fotovoltaico na terceira
linha, a partir do décimo quarto caracter.
            txt[0] =  (Ip / 1000) + 48;
            txt[1] = ((Ip % 1000) / 100 ) + 48;
            txt[2] = '.';
            txt[3] = ((Ip %  100) / 10  ) + 48;
            txt[4] = ( Ip %   10) + 48;
            txt[5] = 0;                                          // Caracter "null" para
finalizar a string
            LCD_I2C_Send_Byte(0x00,_LCD_THIRD_LINE+13);    // Terceira linha decimo quarto
caracter
            LCD_I2C_Write(txt);

            // Coloca no display o valor da potência do painel fotovoltaico na quarta
linha, a partir do terceiro caracter.
            txt[0] =  (Pp / 10000) + 48;
            txt[1] = ((Pp % 10000) / 1000 ) + 48;
            txt[2] = ((Pp %  1000) / 100  ) + 48;
            txt[3] = '.';
            txt[4] = ((Pp %   100) / 10   ) + 48;
            txt[5] = ( Pp %    10) + 48;
            txt[6] = 0;                                          // Caracter "null" para
finaluzar a string
```

```
            LCD_I2C_Send_Byte(0x00,_LCD_FOURTH_LINE+2);       //  Quarta  linha,  terceiro
caracter
            LCD_I2C_Write(txt);
        }

        void Display_VI(void)
        {
            char txt[11];

            // Coloca no display o valor da tensao do painel fotovoltaico na terceira
linha, a partir do quarto caracter.
            txt[0] = 'V';
            txt[1] = 'p';
            txt[2] = ':';
            txt[3] =  (Vp / 1000) + 48;
            txt[4] = ((Vp % 1000) / 100 ) + 48;
            txt[5] = '.';
            txt[6] = ((Vp %  100) / 10  ) + 48;
            txt[7] = ( Vp %   10) + 48;
            txt[8] = 'V';
            txt[9] = ' ';
            txt[10] = 0;                                          // Caracter "null" para
finalizar a string
            LCD_I2C_Send_Byte(0x00,_LCD_SECOND_LINE);   // Terceira linha, quarto caracter
            LCD_I2C_Write(txt);

            // Coloca no display o valor da corrente do painel fotovoltaico na terceira
linha, a partir do décimo quarto caracter.
            txt[0] = 'I';
            txt[1] = 'p';
            txt[2] = ':';
            txt[3] =  (Ip / 1000) + 48;
            txt[4] = ((Ip % 1000) / 100 ) + 48;
            txt[5] = '.';
            txt[6] = ((Ip %  100) / 10  ) + 48;
            txt[7] = ( Ip %   10) + 48;
            txt[8] = 'A';
            txt[9] = 0;                                           // Caracter "null" para
finalizar a string
            LCD_I2C_Send_Byte(0x00,_LCD_SECOND_LINE+10);   // Terceira linha decimo quarto
caracter
            LCD_I2C_Write(txt);

        }

        void Display_P(void)
        {
            char txt[12];
```

```
            // Coloca no display o valor da potência do painel fotovoltaico na quarta
linha, a partir do terceiro caracter.
            txt[0]  = 'P';
            txt[1]  = 'p';
            txt[2]  = ':';
            txt[3]  =  (Pp / 10000) + 48;
            txt[4]  = ((Pp % 10000) / 1000 ) + 48;
            txt[5]  = ((Pp %  1000) / 100  ) + 48;
            txt[6]  = '.';
            txt[7]  = ((Pp %   100) / 10   ) + 48;
            txt[8]  = ( Pp %    10) + 48;
            txt[9]  = 'W';
            txt[10] = ' ';
            txt[11] = ' ';
            txt[12] = ' ';
            txt[13] = ' ';
            txt[14] = ' ';
            txt[15] = ' ';
            txt[16] = ' ';
            txt[17] = ' ';
            txt[18] = ' ';
            txt[19] = ' ';
            txt[20] = 0;                                          // Caracter "null" para
finaluzar a string
            LCD_I2C_Send_Byte(0x00,_LCD_SECOND_LINE);   // Quarta linha, terceiro caracter
            LCD_I2C_Write(txt);
        }

        void Read_Analog (void)
        {

            Vp = ADC_Read(0);  // Read analog value from channel 0 (Photovoltaic module
voltage)
            Vp = Vp * 5000/1023;
            Ip = ADC_Read(1);  // Read analog value from channel 1 (Photovoltaic module
current)
            Ip = Ip * 5015/1023;
            /*EA3 = ADC_Read(2);  // Read analog value from channel 2
            EA3 = EA3 * 500 / 1023;
            EA4 = ADC_Read(2);  // Read analog value from channel 3
            EA4 = EA4 * 500 / 1023;
            EA5 = ADC_Read(2);  // Read analog value from channel 4
            EA5 = EA5 * 500 / 1023;
            EA6 = ADC_Read(2);  // Read analog value from channel 5
            EA6 = EA6 * 500 / 1023;
            EA7 = ADC_Read(2);  // Read analog value from channel 6
            EA7 = EA7 * 500 / 1023;
```

```
            EA8 = ADC_Read(2);  // Read analog value from channel 7
            EA8 = EA8 * 500 / 1023;   */
        }

        void init(void)
        {
            TRISA = 0xFF;                       // Define portA como entradas
            ADCON1 = 0b00000111;                 // define os canais 0~7 como analogico, o
restante como digital.
            ADCON2 |= 0b10000000;                  // Seta justificao a direita para os
valores analogicos.
            TRISB = 0x00;                       // RDefine portB como saidas
            TRISD.B0 = 1;                           // Configura RD0 como entrada - Sensor
temperatura painel fotovoltaico
            TRISD.B1 = 1;                           // Configura RD1 como entrada - Sensor
temperatura inversor
            CMCON = 0x07;
            CCP1CON = 0;
            RED_LED = 1;
            GREEN_LED = 1;
            Delay_ms(50);
            Soft_I2C_Init();
            LCD_I2C_Init();
            ADC_Init();
            // Inicializa a interface SPI
            SPI1_Init_Advanced(_SPI_MASTER_OSC_DIV64,            _SPI_DATA_SAMPLE_MIDDLE,
_SPI_CLK_IDLE_LOW, _SPI_LOW_2_HIGH);
            RED_LED = 0;

           if (Mmc_Fat_Init() == 0)       // reinicializa interface SPI (cartao) numa
velocidade maior
               SPI1_Init_Advanced(_SPI_MASTER_OSC_DIV4,          _SPI_DATA_SAMPLE_MIDDLE,
_SPI_CLK_IDLE_LOW, _SPI_LOW_2_HIGH);
           else
           {
                Message_Card_Error();   // se nao conseguiu inicializar a FAT16 exibe
mensagem de erro e trava o micro. É preciso ressetar depois de arrumar o cartão.
           }
        }

        void Escreve_mascaras(void)
        {
            char texto[20];

            // escreve no display a mascara   "Tp  .  °C Ti  .  °C"
            LCD_I2C_Send_Byte(0x00,_LCD_SECOND_LINE);   // Posiciona na segunda linha
            texto[0] = 'T';
            texto[1] = 'p';
```

```
texto[2] = ' ';
texto[3] = ' ';
texto[4] = '.';
texto[5] = ' ';
texto[6] = ' ';
texto[7] = 223;
texto[8] = 'C';
texto[9] = ' ';
texto[10] = 'T';
texto[11] = 'i';
texto[12] = ' ';
texto[13] = ' ';
texto[14] = '.';
texto[15] = ' ';
texto[16] = ' ';
texto[17] = 223;
texto[18] = 'C';
texto[19] = 0;
LCD_I2C_Write(texto);

 // escreve no display a mascara   "Vp   .  V Ip   .  A"
LCD_I2C_Send_Byte(0x00,_LCD_THIRD_LINE);   // Posiciona na terceira linha
texto[0] = 'V';
texto[1] = 'p';
texto[2] = ' ';
texto[3] = ' ';
texto[4] = '.';
texto[5] = ' ';
texto[6] = ' ';
texto[7] = ' ';
texto[8] = 'V';
texto[9] = ' ';
texto[10] = 'I';
texto[11] = 'p';
texto[12] = ' ';
texto[13] = ' ';
texto[14] = '.';
texto[15] = ' ';
texto[16] = ' ';
texto[17] = ' ';
texto[18] = 'A';
texto[19] = 0;
LCD_I2C_Write(texto);

// escreve no display a mascara   "Pp   .  W"
LCD_I2C_Send_Byte(0x00,_LCD_FOURTH_LINE);   // Posiciona na quarta linha
texto[0] = 'P';
texto[1] = 'p';
```

```
            texto[2] = ' ';
            texto[3] = ' ';
            texto[4] = ' ';
            texto[5] = '.';
            texto[6] = ' ';
            texto[7] = ' ';
            texto[8] = 'W';
            texto[9] = ' ';
            texto[10] =  0;
            LCD_I2C_Write(texto);
            LCD_I2C_Write(filename);      // coloca o nome do arquivo de LOG (só para
verificacao)

        }

        void set_chars(void)
        {
                                          // Define a base do nome do arquivo de LOG
            filename[0]  = 'L';
            filename[1]  = 'A';
            filename[2]  = 'A';
            filename[3]  = 'M';
            filename[4]  = 'M';
            filename[5]  = 'D';
            filename[6]  = 'D';
            filename[7]  = '.';
            filename[8]  = 'c';
            filename[9]  = 's';
            filename[10] = 'v';
            filename[11] =  0 ;           // Caracter "null" para finaluzar a string
                                          //o array do nome do arquivo nao pode ser
maior que 12 (formado do arquivo DOS 8.3)

                                          // Define a base dos dados que serao gravados
no LOG
            log_string[0]  = 'H';
            log_string[1]  = 'H';
            log_string[2]  = ':';
            log_string[3]  = 'M';
            log_string[4]  = 'M';
            log_string[5]  = ':';
            log_string[6]  = 'S';
            log_string[7]  = 'S';
            log_string[9]  = ';';
            log_string[9]  = 'V';
            log_string[10] = 'p';
            log_string[11] = '.';
            log_string[12] = 'V';
```

```
            log_string[13] = 'p';
            log_string[14] = ';';
            log_string[15] = 'I';
            log_string[16] = 'p';
            log_string[17] = '.';
            log_string[18] = 'I';
            log_string[19] = 'p';
            log_string[20] = ';';
            log_string[21] = 'P';
            log_string[22] = 'P';
            log_string[23] = 'p';
            log_string[24] = '.';
            log_string[25] = 'P';
            log_string[26] = 'p';
            log_string[27] = ';';
            log_string[28] = 'T';
            log_string[29] = 'p';
            log_string[30] = '.';
            log_string[31] = 'T';
            log_string[32] = 'p';
            log_string[33] = ';';
            log_string[34] = 'T';
            log_string[35] = 'i';
            log_string[36] = '.';
            log_string[37] = 'T';
            log_string[38] = 'i';
            log_string[39] = 0x0A;          // Caracter "CR" (Carrier Return)
            log_string[40] = 0x0D;          // Caracter "LF" (Line Feed)
            log_string[41] = 0;             // Caracter "null" para finalizar a string
        }

        void main()
        {
             char old_second;
             char texto[21];

             init();
             //write_time();                                  //so para escrever o horario
no pcf.
                                                            // tem que colocar o hoário
pra ser escrito no arquivo PCF8583.h
             set_chars();

             ReadTime();

             // coloca as informacoes de ano mes e dia no nome do arquivo
             filename[1] = ((TimeRead.year % 100) / 10) + 48;
             filename[2] = (TimeRead.year % 10)  + 48;
```

```
            filename[3] = (TimeRead.month / 10) + 48;
            filename[4] = (TimeRead.month % 10) + 48;
            filename[5] = (TimeRead.day / 10) + 48;
            filename[6] = (TimeRead.day % 10) + 48;

            while (!File_Test_Exist())                          // verifica se o arquivo
"LogAAMMDD.csv" existe.
                                                                // Se não existir tem que
criar.
            {
                LCD_I2C_Send_Byte(0x00,_LCD_FOURTH_LINE);     // Posiciona na quarta linha
                texto[0]  = 'C';
                texto[1]  = 'r';
                texto[2]  = 'i';
                texto[3]  = 'a';
                texto[4]  = 'n';
                texto[5]  = 'd';
                texto[6]  = 'o';
                texto[7]  = ' ';
                texto[8]  = 'A';
                texto[9]  = 'r';
                texto[10] = 'q';
                texto[11] = 'u';
                texto[12] = 'i';
                texto[13] = 'v';
                texto[14] = 'o';
                texto[15] = ' ';
                texto[16] = 'L';
                texto[17] = 'O';
                texto[18] = 'G';
                texto[19] =  0;
                LCD_I2C_Write(texto);
                File_Create_New();
                delay_ms(500);
                RED_LED = ~RED_LED;
            }

            LCD_I2C_Send_Byte(0x00, _LCD_CLEAR);      // Limpa o display

            delay_ms(100);

                                            //  Parte  a  conversao  de  temperatura  do
sensor 1 (painel fotovoltaico)
            Ow_Reset(&PORTD, 0);            // Sinal de reset para a comunicação OneWire
            Ow_Write(&PORTD, 0, 0xCC);      // Envia o comando SKIP_ROM
            Ow_Write(&PORTD, 0, 0x44);      // Envia o Comando CONVERT_T
```

```
                                        //  Parte  a  conversao  de  temperatura  do
sensor 2 (inversor)
            Ow_Reset(&PORTD, 1);        // Sinal de reset para a comunicação OneWire
            Ow_Write(&PORTD, 1, 0xCC);  // Envia o comando SKIP_ROM
            Ow_Write(&PORTD, 1, 0x44);  // Envia o comando CONVERT_T

            delay_ms(500);

            //Escreve_mascaras();

            Display_Time();

                                        // Faz a leitura de temperatura do sensor 1
(inversor)
            Ow_Reset(&PORTD, 0);        // Sinal de reset para a comunicação OneWire
            Ow_Write(&PORTD, 0, 0xCC);  // Envia o comando SKIP_ROM
            Ow_Write(&PORTD, 0, 0xBE);  // Envia o comando READ_SCRATCHPAD
            delay_ms(50);

            Tp =  Ow_Read(&PORTD, 0);
            Tp = (Ow_Read(&PORTD, 0) << 8) + Tp;

                                        // Faz a leitura de temperatura do sensor 2
(Painel Fotovoltaico)
            Ow_Reset(&PORTD, 1);        // Sinal de reset para a comunicação OneWire
            Ow_Write(&PORTD, 1, 0xCC);  // Envia o comando SKIP_ROM
            Ow_Write(&PORTD, 1, 0xBE);  // Envia o comando READ_SCRATCHPAD
            delay_ms(50);

             Ti =  Ow_Read(&PORTD, 1);
             Ti = (Ow_Read(&PORTD, 1) << 8) + Ti;
             Display_Temperature();

            while(1)          // Loop principal
            {
                // aqui tem que colocar uma rotina para verificar o botao de start/stop de
logging.
                // Se apertada para o logging, fecha o arquivo, e fica atualizando a tela
                // se apertar de novo, abre o arquivo e volta a escrever dados nele.


                ReadTime();

                if (TimeRead.seconds == 0 && TimeRead.minutes == 0 && TimeRead.hours == 0)
// se virou o dia fecha o arquivo atual e cria um novo arquivo
                {
                    while (!Mmc_Fat_Close());
                    filename[1] = ((TimeRead.year % 100) / 10) + 48;
```

```
                    filename[2] = (TimeRead.year % 10)  + 48;
                    filename[3] = (TimeRead.month / 10) + 48;
                    filename[4] = (TimeRead.month % 10) + 48;
                    filename[5] = (TimeRead.day / 10) + 48;
                    filename[6] = (TimeRead.day % 10) + 48;

                    LCD_I2C_Send_Byte(0x00,_LCD_FOURTH_LINE+9);    // Posiciona na quarta
linha decimo caracter
                    LCD_I2C_Write(filename);
                    File_Create_New();
                }

                if (TimeRead.seconds != old_second)           // verifica se teve mudanca
de segundos
                {
                   old_second = TimeRead.seconds;            // guarda o segundo atual
                   Display_Time();                             // atualiza o display com o
novo horario e data
                   Read_Analog();                              // Le os valores analogicos
de tensao e corrente
                   Ppt = Vp * Ip;                             // calcula a potencia
                   Pp = Ppt /100;                             // acerta os valores
                   //Display_Analog();                               // atualiza os valore
analogicos no display

                   switch(TimeRead.seconds%10)                 // verifica a unidade dos
segundos. Dependendo do resultado executa uma operacao especifica
                   {
                   case 0:                                        // Parte a conversao de
temperatura do sensor 1 (painel fotovoltaico)
                      Ow_Reset(&PORTD, 0);                     // Sinal de reset para a
comunicação OneWire
                      Ow_Write(&PORTD, 0, 0xCC);        // Envia o comando SKIP_ROM
                      Ow_Write(&PORTD, 0, 0x44);        // Envia o comando CONVERT_T

                                                        //   Parte   a   conversao   de
temperatura do sensor 2 (inversor)
                      Ow_Reset(&PORTD, 1);                     // Sinal de reset para a
comunicação OneWire
                      Ow_Write(&PORTD, 1, 0xCC);        // Envia o comando SKIP_ROM
                      Ow_Write(&PORTD, 1, 0x44);        // Envia o comando CONVERT_T
                      Display_Vi();                       // atualiza os valore analogicos
no display

                   break;
                   case 1:                                  // Faz a leitura de temperatura
do sensor 1 (painel fotovoltaico)
```

```
                        Ow_Reset(&PORTD, 0);                    // Sinal de reset para a
comunicação OneWire
                        Ow_Write(&PORTD, 0, 0xCC);        // Envia o comando SKIP_ROM
                        Ow_Write(&PORTD,  0,  0xBE);                //  Envia  o  comando
READ_SCRATCHPAD
                        delay_ms(50);

                        Tp =  Ow_Read(&PORTD, 0);
                        Tp = (Ow_Read(&PORTD, 0) << 8) + Tp;
                        //Display_Temperature();
                        Display_Vi();                      // atualiza os valore analogicos
no display
                    break;

                    case 2:                                 // Faz a leitura de temperatura
do sensor 2 (inversor)
                        Ow_Reset(&PORTD, 1);                    // Sinal de reset para a
comunicação OneWire
                        Ow_Write(&PORTD, 1, 0xCC);        // Envia o comando SKIP_ROM
                        Ow_Write(&PORTD,  1,  0xBE);                //  Envia  o  comando
READ_SCRATCHPAD
                        delay_ms(50);

                        Ti =  Ow_Read(&PORTD, 1);
                        Ti = (Ow_Read(&PORTD, 1) << 8) + Ti;
                        //Display_Temperature();
                        Display_Vi();                      // atualiza os valore analogicos
no display
                    break;

                    case 3:
                    case 4:
                    case 5:
                    Display_Vi();
                    break;

                    case 6:
                    Display_Temperature();
                    break;

                    case 8:
                    case 9:
                    Display_P();
                    break;
                    }
                    //  aqui  tem  que  colocar  no  log  os  dados  no  formato:
HH:MM:SS;Vp.Vp;Ip.Ip;Tp.Tp;Ti.Ti0x0A0x0D (34 caracteres)
                    log_string[0]  = (TimeRead.hours / 10) + 48;
```

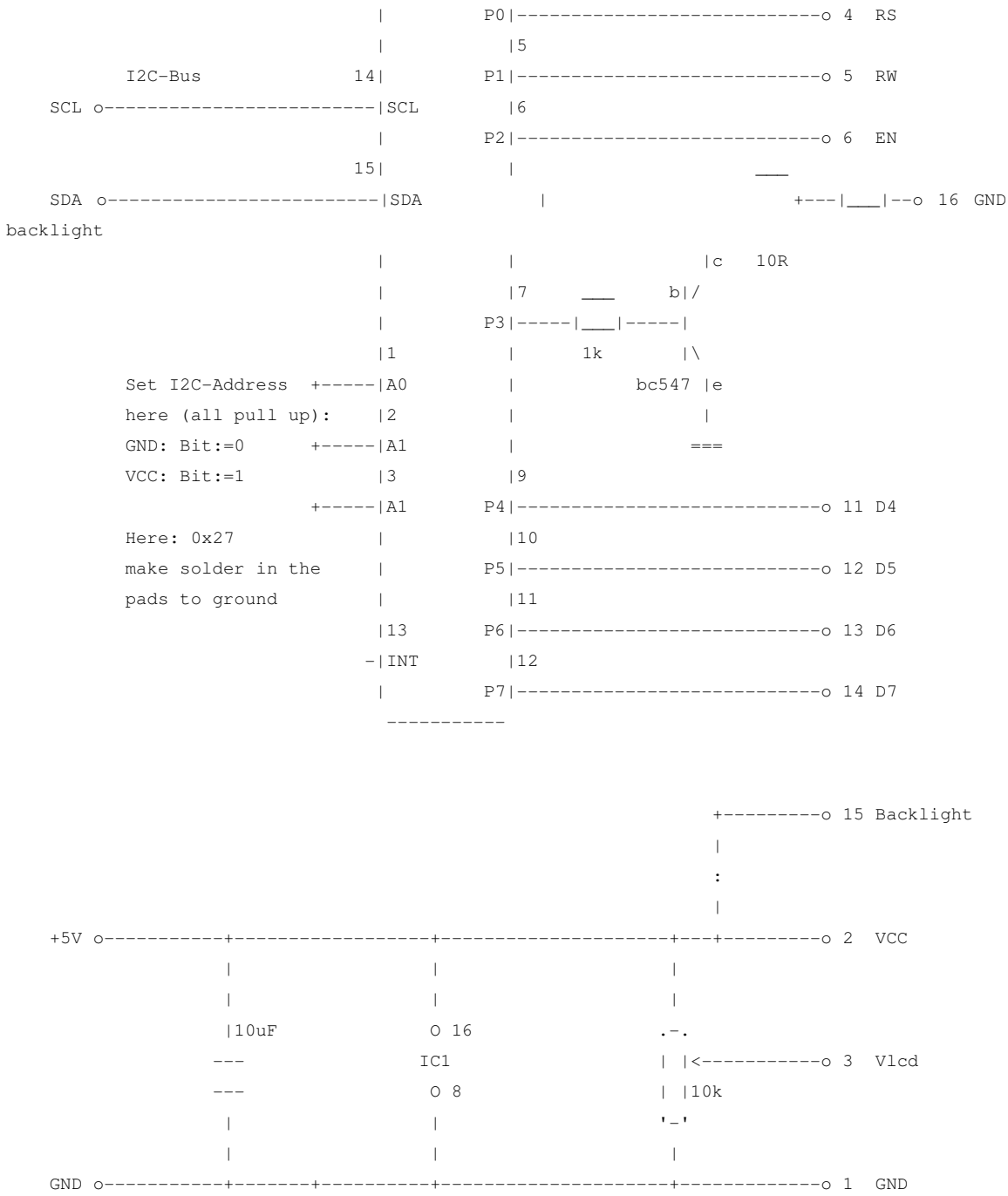
```
                log_string[1]  = (TimeRead.hours % 10) + 48;
                log_string[2]  = ':';
                log_string[3]  = (TimeRead.minutes / 10) + 48;
                log_string[4]  = (TimeRead.minutes % 10) + 48;
                log_string[5]  = ':';
                log_string[6]  = (TimeRead.seconds / 10) + 48;
                log_string[7]  = (TimeRead.seconds % 10) + 48;
                log_string[8]  = ';';
                log_string[9]  =  (Vp / 1000) + 48;
                log_string[10] = ((Vp % 1000) / 100 ) + 48;
                log_string[11] = '.';
                log_string[12] = ((Vp %  100) / 10  ) + 48;
                log_string[13] = ( Vp %   10) + 48;
                log_string[14] = ';';
                log_string[15] =  (Ip / 1000) + 48;
                log_string[16] = ((Ip % 1000) / 100 ) + 48;
                log_string[17] = '.';
                log_string[18] = ((Ip %  100) / 10  ) + 48;
                log_string[19] = ( Ip %   10) + 48;
                log_string[20] = ';';
                log_string[21] =  (Pp / 10000) + 48;
                log_string[22] = ((Pp % 10000) / 1000 ) + 48;
                log_string[23] = ((Pp %  1000) / 100  ) + 48;
                log_string[24] = '.';
                log_string[25] = ((Pp %   100) / 10   ) + 48;
                log_string[26] = ( Pp %    10) + 48;
                log_string[27] = ';';
//                log_string[28] = (Tp / 1000) + 48;
//                log_string[29] = (Tp / 100 ) + 48;
                log_string[30] = '.';
//                log_string[31] = (Tp / 10  ) + 48;
//                log_string[32] = (Tp % 10  ) + 48;
                log_string[33] = ';';
//                log_string[34] = (Ti / 1000) + 48;
//                log_string[35] = (Ti / 100 ) + 48;
                log_string[36] = '.';
//                log_string[37] = (Ti / 10  ) + 48;
//                log_string[38] = (Ti % 10  ) + 48;
                log_string[39] = 0x0A;

                File_Append();

            }
        }
    }
```

HW_def.H

```
//Hardware definition for software I2C interface
sbit Soft_I2C_Scl at RB1_bit;           // SCL port
sbit Soft_I2C_Sda at RB0_bit;           // SDA port

sbit Soft_I2C_Scl_Direction at TRISB1_bit;   // Tristate definition for SCL port
sbit Soft_I2C_Sda_Direction at TRISB0_bit;   // Tristate definition for SDA port
// eof Hardware definition for software I2C interface

// Hardware definition for MMC module connections
sbit Mmc_Chip_Select            at RC0_bit;  // for writing to output pin always use latch
(PIC18 family)
sbit Mmc_Chip_Select_Direction at TRISC0_bit;
// eof Hardware definition for MMC module connections
```

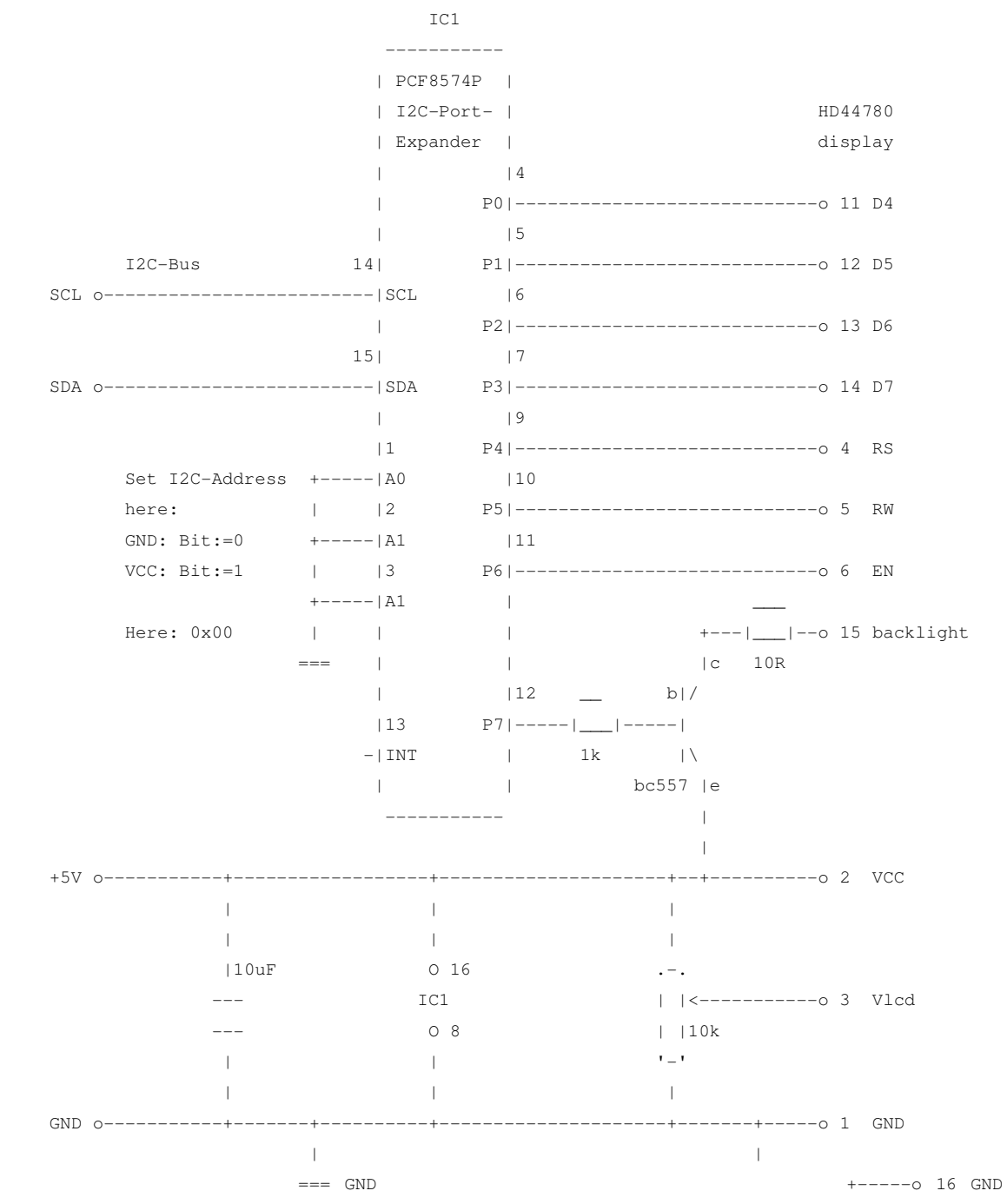
LCD_I2C_DRIVER.H

```
// LCD_I2C_DRIVER FOR MICROCHIP PIC
// Tested on PIC16F628A
// developed for microC PRO for PIC v.6.0.0  (may need to be adapted to other compilers)
// using SOFT I2C routines
//
// Written by: Daniel Piazzetta    (daniellp@ibest.com.br)
// REV-0
// 04 february 2014
//
// WRITES to LCD I2C Adapters that use I2C chip PCF8574P
// Does not READ
//
// Do not forget to include this library into your main code
// #include "LCD_I2C_DRIVER.h"
//
// Notes for "IIC module" adapter
// Model tested had I2C Address = 0x27.  Change if yours is different
//
//    P7 - D7        P3 - BACKLIGHT LED
//    P6 - D6        P2 - E
//    P5 - D5        P1 - R/W
//    P4 - D4        P0 - RS
//
/*
D44780: PCF8574P port expander on I2C bus

                                    IC1
                                 -----------
                                | PCF8574P  |
                                | I2C-Port- |                          HD44780
                                | Expander  |                          display
                                |           |4
```




```
// If your module has the following pinout:
//    P3 - D7        P7 - BACKLIGHT LED
//    P2 - D6        P6 - E
//    P1 - D5        P5 - R/W
```

```
//    P0 - D4        P4 - RS
//
// you must adapt the library to match your pinout.

D44780: PCF8574P port expander on I2C bus
```



```
*/



// IIC module
#define  LCD_I2C_add 0x27
//   RS  =    I2C_BYTE.0
//   RW  =    I2C_BYTE.1
//   E   =    I2C_BYTE.2
//   BL  =    I2C_BYTE.3
//   D4  =    I2C_BYTE.4
//   D5  =    I2C_BYTE.5
//   D6  =    I2C_BYTE.6
//   D7  =    I2C_BYTE.7


int I2C_BYTE = 0;

// Following line creates appropriate address byte for I2C interface (shift left 1-bit)
#define LCD_I2C_ADDR (LCD_I2C_add<<1)   //0x4E //

void LCD_I2C_Write(char text[]);                    // writes text string 'text[]'
void LCD_I2C_Init();                                       // Hitachi 44780U initialization
commands
void LCD_I2C_Putc(char c);                          // writes single character 'c'
void LCD_I2C_Send_Byte(char address, char n);        // sends arbitraty byte 'n' to address
'0' - useful for special commands/characters
void LCD_I2C_Write_Nibble(char n);

int lcd_j,lcd_max,lcd_tempbyte;
const int lcd_fixed_delay_us=500;                     // Fixed delay to allow LCD to process
I2C instruction
int LCD_PIC_line;
unsigned char lcd_buffer[20];


//==================================
void LCD_I2C_Write(char text[])
{
// Writes a string text[] to LCD via I2C
   I2C_BYTE |= 0b00000001;                   //RS=1 --> Data
   I2C_BYTE &= 0b11111101;                   //RW  = 0
   I2C_BYTE &= 0b11111011;                   //E = 0
   I2C_BYTE |= 0b00001000;                   //BL  = BL_ON;

   Soft_I2C_Start();
   delay_ms(1);
   Soft_I2C_Write(LCD_I2C_ADDR);
```

```
   delay_ms(1);
   lcd_max = strlen(text);
   for(lcd_j=0; lcd_j<lcd_max; ++lcd_j)
   {
      lcd_tempbyte=text[lcd_j];

// Send upper nibble
      LCD_I2C_Write_Nibble(lcd_tempbyte);
      delay_ms(1);
// Send lower nibble
      lcd_tempbyte <<= 4;
      LCD_I2C_Write_Nibble(lcd_tempbyte);
      delay_ms(1);
   }
   Soft_I2C_Stop();
}


//===================================
void LCD_I2C_Putc(char c)
{
// Writes a char 'c' to LCD via I2C to include some common formats
   switch(c)
   {
      case '\f':
         LCD_I2C_Send_Byte(0x00,0x01);
         LCD_PIC_line = 1;
         delay_ms(50);
         break;

      case '\n':
          LCD_I2C_Send_Byte(0x01, ++LCD_PIC_line);
          break;

      case '\b':
          LCD_I2C_Send_Byte(0x00,0x10);
          break;

      default:
          LCD_I2C_Send_Byte(0x01,c);
          break;
   }
}

//===================================
void LCD_I2C_Send_Byte(char address, char n)
{
// HD44780U LCD Address Type (RS): Command=0, Data=1
```

```
//

   char temp_data = 0;

   if(address)
   {
      I2C_BYTE |= 0b00000001;       //RS=1 --> Data
   }
   else
   {
      I2C_BYTE &= 0b11111110;       //RS=0 --> Command
   }

   I2C_BYTE &= 0b11111101;          //RW  = 0
   I2C_BYTE &= 0b11111011;          //E = 0
   I2C_BYTE |= 0b00001000;          //BL  = BL_ON;

   Soft_I2C_Start();
   delay_ms(1);
   Soft_I2C_Write(LCD_I2C_ADDR);
   delay_ms(1);
// Send upper nibble
   LCD_I2C_Write_Nibble(n);
   delay_ms(1);

// Send lower nibble
   n <<= 4;
   LCD_I2C_Write_Nibble(n);
   delay_ms(1);
   Soft_I2C_Stop();
}

//==================================
void LCD_I2C_Init()
{
// Initialization commands for Hitachi HD44780U LCD Display
//
   delay_ms(300);                   // Let LCD power up

// Following bytes are all Command bytes, i.e. address = 0x00
   LCD_I2C_Send_Byte(0x00,  0x03);      //  Write  Nibble  0x03  three  times  (per  HD44780U
initialization spec)
   delay_us(lcd_fixed_delay_us);    // (per HD44780U initialization spec)
   LCD_I2C_Send_Byte(0x00, 0x03);   //
   delay_us(lcd_fixed_delay_us);    // (per HD44780U initialization spec)
   LCD_I2C_Send_Byte(0x00, 0x03);   //
   delay_us(lcd_fixed_delay_us);    // (per HD44780U initialization spec)
```

```
   LCD_I2C_Send_Byte(0x00, 0x02);    // Write Nibble 0x02 once (per HD44780U initialization
spec)
   delay_us(lcd_fixed_delay_us);     // (per HD44780U initialization spec)
   LCD_I2C_Send_Byte(0x00, 0x28);    // Set mode: 4-bit, 2+lines, 5x8 dots
   delay_us(lcd_fixed_delay_us);     // (per HD44780U initialization spec)
   LCD_I2C_Send_Byte(0x00, 0x0C);    // Display ON
   delay_us(lcd_fixed_delay_us);     // (per HD44780U initialization spec)
   LCD_I2C_Send_Byte(0x00, 0x01);    // Clear display
   delay_us(lcd_fixed_delay_us);     // (per HD44780U initialization spec)
   LCD_I2C_Send_Byte(0x00, 0x06);    // Set cursor to increment
   delay_us(lcd_fixed_delay_us);     // (per HD44780U initialization spec)
   Soft_I2C_Stop();                  // Terminate I2C transfer
   delay_ms(100);                    //
}

//===================================
void LCD_I2C_Write_Nibble(char n)
{
// Send nibble

   n &= 0xF0;                        // clean lower nibble from data

   I2C_BYTE &= 0x0F;                 // clean upper nibble  from control pins

   I2C_BYTE |= n;                       // put upper nibble (data) together with control to
transmit

   I2C_BYTE &= 0b11111011;           //E = 0
   Soft_I2C_Write(I2C_BYTE);
   delay_ms(1);
   I2C_BYTE |= 0b00000100;           //E = 1
   Soft_I2C_Write(I2C_BYTE);
   delay_ms(1);
   I2C_BYTE &= 0b11111011;           //E = 0
   Soft_I2C_Write(I2C_BYTE);
   delay_ms(1);
}
```

PCF8583.H

```
/*
 * Project name:
     RTC_Read (Demonstration on Working with the RTC Module)
 * Copyright:
     (c) MikroElektronika, 2006.
 * Description:
      This project is simple demonstration how to Read date and time from
```

```
      PCF8583 RTC (real-time clock). The code can be used with any MCU that has
      MSSP module at PORTC.
      The offset from Y2000 is stored in the PCF8583 RAM at address 0x10.
      The time format is 24h and weekdays are not used.
 * Test configuration:
     MCU:             PIC16F887
     Dev.Board:       EasyPIC5
     Oscillator:      HS, 8.000 MHz
     Ext. Modules:    RTC on PORTC
     SW:              mikroC v8.0.0.0
 * NOTES:
     - Be sure to switch the portC LEDs OFF
     - For proper I2C communication, PORTC must be in the pull-up mode
     - In order to use the example, address pin A0 of PCF8583 must be set to 0V.
       (on mikroElektronika's RTC module this is done by default)
 */

#define _RTC_ADDRESS 0xA0

struct TTime {
       char year, month, day, hours, minutes, seconds;
} TimeRead;

struct Wr_TTime {
       char year, month, day, hours, minutes, seconds;
} TimeToWrite;

char yearmod4, byteRead;

void ReadTime()
{
char updateYear;

   updateYear = 0;

   Soft_I2C_Start();               // issue start signal
   Soft_I2C_Write(_RTC_ADDRESS);              // address PCF8583
   Soft_I2C_Write(2);                 // first word address
   Soft_I2C_Start();                // issue start signal
   Soft_I2C_Write(_RTC_ADDRESS|0x01);              // address PCF8583 for reading R/W=1

   byteRead = Soft_I2C_Read(1);      // read seconds byte
   TimeRead.seconds = (byteRead >> 4)*10 + (byteRead & 0x0F); // transform seconds
   delay_us(250);                        // (time to finish the I2C reading)

   byteRead = Soft_I2C_Read(1);     // read minutes byte
   TimeRead.minutes = (byteRead >> 4)*10 + (byteRead & 0x0F); // transform minutes
   delay_us(250);                        // (time to finish the I2C reading)
```

```
   byteRead = Soft_I2C_Read(1);     // read hours byte
   TimeRead.hours = (byteRead >> 4)*10 + (byteRead & 0x0F); // transform hours
   delay_us(250);                        // (time to finish the I2C reading)

   byteRead = Soft_I2C_Read(1);     // read year/day byte
   TimeRead.day = ((byteRead & 0b00110000) >> 4)*10 + (byteRead & 0x0F); // transform day
   yearmod4 = (byteRead & 0b11000000) >> 6;   // get year mod 4 from RTC
   delay_us(250);                        // (time to finish the I2C reading)

   byteRead = Soft_I2C_Read(0);     // read weekday/month byte
   TimeRead.month = ((byteRead & 0b00010000) >> 4)*10 + (byteRead & 0x0F); // transform
month
   delay_us(250);                        // (time to finish the I2C reading)
   Soft_I2C_Stop();

   Soft_I2C_Start();                // issue start signal
   Soft_I2C_Write(0xA0);              // address PCF8583
   Soft_I2C_Write(0x10);              // first word address
   Soft_I2C_Start();                // issue start signal
   Soft_I2C_Write(0xA1);              // address PCF8583 for reading R/W=1

   byteRead = Soft_I2C_Read(0);     // read year
   if (yearmod4 != byteRead % 4 ) { // check if year is incremented in RTC
       byteRead++;  // in this case the new value should be written to RTC RAM at address
16(0x10)
       updateYear = 1;
       }
   TimeRead.year = byteRead;

   delay_us(250);                        // (time to finish the I2C reading)
   Soft_I2C_Stop();

   if (updateYear > 0) {
      Soft_I2C_Start();            // issue start signal
      Soft_I2C_Write(_RTC_ADDRESS);          // address PCF8530
      Soft_I2C_Write(0x10);          // start from word at address 16
      Soft_I2C_Write(TimeRead.year);         // write year to RAM
      Soft_I2C_Stop();             // issue stop signal
      }
}

void write_time(void)
{
   TimeToWrite.year    = 15;    // 2011 (offset from 2000)
   TimeToWrite.month   = 4;
   TimeToWrite.day     = 30;
   TimeToWrite.hours   = 8;
```

```
   TimeToWrite.minutes = 45;
   TimeToWrite.seconds = 0;


   Soft_I2C_Start();            // issue start signal
   Soft_I2C_Write(_RTC_ADDRESS);          // address PCF8583
   Soft_I2C_Write(0);            // start from word at address 0 (configuration word)
   Soft_I2C_Write(0x80);          // write 0x80 to config (pause counter...)
   Soft_I2C_Write(0);            // write 0 to cents word
   Soft_I2C_Write(((TimeToWrite.seconds/10)<<4)   +   (TimeToWrite.seconds%10));   //   write
seconds word
   Soft_I2C_Write(((TimeToWrite.minutes/10)<<4)   +   (TimeToWrite.minutes%10));   //   write
minutes word
   Soft_I2C_Write(((TimeToWrite.hours/10)<<4) + (TimeToWrite.hours%10)); // write hours word
   Soft_I2C_Write(((TimeToWrite.year%4)<<6)            +            ((TimeToWrite.day/10)<<4)
+(TimeToWrite.day%10));
   Soft_I2C_Write(((TimeToWrite.month/10)<<4)   +   (TimeToWrite.month%10));      //   write
weekday/month
   Soft_I2C_Stop();             // issue stop signal

   Soft_I2C_Start();            // issue start signal
   Soft_I2C_Write(_RTC_ADDRESS);          // address PCF8530
   Soft_I2C_Write(0x10);          // start from word at address 16
   Soft_I2C_Write(TimeToWrite.year);         // write year to RAM
   Soft_I2C_Stop();             // issue stop signal

   Soft_I2C_Start();            // issue start signal
   Soft_I2C_Write(0xA0);           // address PCF8530
   Soft_I2C_Write(0);              // start from word at address 0
   Soft_I2C_Write(0);              // write 0 to config word (enable counting)
   Soft_I2C_Stop();             // issue stop signal

   }
```

SD_Card.H

```
char filename[12];
char log_string[40];


void File_Create_New(void)
{
  char cabecalho [119];
  cabecalho[0]  = 'H';
  cabecalho[1]  = 'o';
  cabecalho[2]  = 'r';
  cabecalho[3]  = 'a';
  cabecalho[4]  = 'r';
```

```
cabecalho[5]  = 'i';
cabecalho[6]  = 'o';
cabecalho[7]  = ';';
cabecalho[8]  = 'T';
cabecalho[9]  = 'e';
cabecalho[10] = 'n';
cabecalho[11] = 's';
cabecalho[12] = 'a';
cabecalho[13] = 'o';
cabecalho[14] = ' ';
cabecalho[15] = 'P';
cabecalho[16] = 'a';
cabecalho[17] = 'i';
cabecalho[18] = 'n';
cabecalho[19] = 'e';
cabecalho[20] = 'l';
cabecalho[21] = ' ';
cabecalho[22] = 'S';
cabecalho[23] = 'o';
cabecalho[24] = 'l';
cabecalho[25] = 'a';
cabecalho[26] = 'r';
cabecalho[27] = ';';
cabecalho[28] = 'C';
cabecalho[29] = 'o';
cabecalho[30] = 'r';
cabecalho[31] = 'r';
cabecalho[32] = 'e';
cabecalho[33] = 'n';
cabecalho[34] = 't';
cabecalho[35] = 'e';
cabecalho[36] = ' ';
cabecalho[37] = 'P';
cabecalho[38] = 'a';
cabecalho[39] = 'i';
cabecalho[40] = 'n';
cabecalho[41] = 'e';
cabecalho[42] = 'l';
cabecalho[43] = ' ';
cabecalho[44] = 'S';
cabecalho[45] = 'o';
cabecalho[46] = 'l';
cabecalho[47] = 'a';
cabecalho[48] = 'r';
cabecalho[49] = ';';
cabecalho[50] = 'P';
cabecalho[51] = 'o';
cabecalho[52] = 't';
```

```
cabecalho[53] = 'e';
cabecalho[54] = 'n';
cabecalho[55] = 'c';
cabecalho[56] = 'i';
cabecalho[57] = 'a';
cabecalho[58] = ' ';
cabecalho[59] = 'P';
cabecalho[60] = 'a';
cabecalho[61] = 'i';
cabecalho[62] = 'n';
cabecalho[63] = 'e';
cabecalho[64] = 'l';
cabecalho[65] = ' ';
cabecalho[66] = 'S';
cabecalho[67] = 'o';
cabecalho[68] = 'l';
cabecalho[69] = 'a';
cabecalho[70] = 'r';
cabecalho[71] = ';';
cabecalho[72] = 'T';
cabecalho[73] = 'e';
cabecalho[74] = 'm';
cabecalho[75] = 'p';
cabecalho[76] = 'e';
cabecalho[77] = 'r';
cabecalho[78] = 'a';
cabecalho[79] = 't';
cabecalho[80] = 'u';
cabecalho[81] = 'r';
cabecalho[82] = 'a';
cabecalho[83] = ' ';
cabecalho[84] = 'P';
cabecalho[85] = 'a';
cabecalho[86] = 'i';
cabecalho[87] = 'n';
cabecalho[88] = 'e';
cabecalho[89] = 'l';
cabecalho[90] = ' ';
cabecalho[91] = 'S';
cabecalho[92] = 'o';
cabecalho[93] = 'l';
cabecalho[94] = 'a';
cabecalho[95] = 'r';
cabecalho[96] = ';';
cabecalho[97] = 'T';
cabecalho[98] = 'e';
cabecalho[99] = 'm';
cabecalho[100] = 'p';
```

```
  cabecalho[101] = 'e';
  cabecalho[102] = 'r';
  cabecalho[103] = 'a';
  cabecalho[104] = 't';
  cabecalho[105] = 'u';
  cabecalho[106] = 'r';
  cabecalho[107] = 'a';
  cabecalho[108] = ' ';
  cabecalho[109] = 'I';
  cabecalho[110] = 'n';
  cabecalho[111] = 'v';
  cabecalho[112] = 'e';
  cabecalho[113] = 'r';
  cabecalho[114] = 's';
  cabecalho[115] = 'o';
  cabecalho[116] = 'r';
  cabecalho[117] = 0x0A;
  cabecalho[118] = 0;
  Mmc_Fat_Assign(&filename, 0xA0);
  Mmc_Fat_Set_File_Date(TimeRead.year+2000,  TimeRead.month,  TimeRead.day,  TimeRead.hours,
TimeRead.minutes, TimeRead.seconds);
  Mmc_Fat_Append();
  Mmc_Fat_Write(cabecalho, 118);
}

void File_Delete()
{
  Mmc_Fat_Assign(&filename, 0x00);
  Mmc_Fat_Delete();
}

short File_Test_Exist()     // Check if file already exist.
{
  return Mmc_Fat_Assign(&filename,0x00);
}

// Opens an existing file and appends data to it
//               (and alters the date/time stamp)
void File_Append()
{
   Mmc_Fat_Assign(&filename, 0x00);
   Mmc_Fat_Set_File_Date(TimeRead.year+2000,  TimeRead.month,  TimeRead.day,  TimeRead.hours,
TimeRead.minutes, TimeRead.seconds);
   Mmc_Fat_Append();                                    // Prepare file for append
   Mmc_Fat_Write(log_string, 40);                       // Write data to assigned file
}

void Message_Card_Error(void)
```

```
{
     char mensagem_erro[20];

     LCD_I2C_Send_Byte(0x00,_LCD_FIRST_LINE+1);   // Posiciona na primeira linha
     mensagem_erro[0]  = 'F';
     mensagem_erro[1]  = 'a';
     mensagem_erro[2]  = 'l';
     mensagem_erro[3]  = 'h';
     mensagem_erro[4]  = 'a';
     mensagem_erro[5]  = ' ';
     mensagem_erro[6]  = 'n';
     mensagem_erro[7]  = 'o';
     mensagem_erro[8]  = ' ';
     mensagem_erro[9]  = 'c';
     mensagem_erro[10] = 'a';
     mensagem_erro[11] = 'r';
     mensagem_erro[12] = 't';
     mensagem_erro[13] = 'a';
     mensagem_erro[14] = 'o';
     mensagem_erro[15] = ' ';
     mensagem_erro[16] = 'S';
     mensagem_erro[17] = 'D';
     mensagem_erro[18] =  0;

     LCD_I2C_Write(mensagem_erro);


     LCD_I2C_Send_Byte(0x00,_LCD_SECOND_LINE+1);   // Posiciona na terceira linha
     mensagem_erro[0]  = 'V';
     mensagem_erro[1]  = 'e';
     mensagem_erro[2]  = 'r';
     mensagem_erro[3]  = 'i';
     mensagem_erro[4]  = 'f';
     mensagem_erro[5]  = 'i';
     mensagem_erro[6]  = 'q';
     mensagem_erro[7]  = 'u';
     mensagem_erro[8]  = 'e';
     mensagem_erro[9] = ' ';
     mensagem_erro[10] = 'o';
     mensagem_erro[11] = ' ';
     mensagem_erro[12] = 'c';
     mensagem_erro[13] = 'a';
     mensagem_erro[14] = 'r';
     mensagem_erro[15] = 't';
     mensagem_erro[16] = 'a';
     mensagem_erro[17] = 'o';
     mensagem_erro[18] =  0;;
     LCD_I2C_Write(mensagem_erro);
```

```
    LCD_I2C_Send_Byte(0x00,_LCD_FOURTH_LINE+1);   // Posiciona na quarta linha
    mensagem_erro[0]  = 'r';
    mensagem_erro[1]  = 'e';
    mensagem_erro[2]  = 's';
    mensagem_erro[3]  = 'e';
    mensagem_erro[4]  = 't';
    mensagem_erro[5]  = ' ';
    mensagem_erro[6]  = 'o';
    mensagem_erro[7]  = ' ';
    mensagem_erro[8]  = 'd';
    mensagem_erro[9]  = 'a';
    mensagem_erro[10] = 't';
    mensagem_erro[11] = 'a';
    mensagem_erro[12] = 'l';
    mensagem_erro[13] = 'o';
    mensagem_erro[14] = 'g';
    mensagem_erro[15] = 'g';
    mensagem_erro[16] = 'e';
    mensagem_erro[17] = 'r';
    mensagem_erro[18] = 0;
    LCD_I2C_Write(mensagem_erro);

    RED_LED = 1;
    GREEN_LED=0;
    asm SLEEP;
}
```