UNIVERSIDADE DO ESTADO DE SANTA CATARINA
CENTRO DE CIÊNCIAS TECNOLÓGICAS - CCT
BACHARELADO EM ENGENHARIA ELÉTRICA

GABRIELA PEREIRA DA SILVA

ESTUDO E APLICAÇÃO DE CIRCUITO INTEGRADO PARA ESTIMAÇÃO DO ESTADO DE CARGA (SOC) E ESTADO DE SAÚDE (SOH) DE BATERIAS DE CHUMBO-ÁCIDO

JOINVILLE

2017

UNIVERSIDADE DO ESTADO DE SANTA CATARINA
CENTRO DE CIÊNCIAS TECNOLÓGICAS - CCT
BACHARELADO EM ENGENHARIA ELÉTRICA

GABRIELA PEREIRA DA SILVA

ESTUDO E APLICAÇÃO DE CIRCUITO INTEGRADO PARA ESTIMAÇÃO DO ESTADO DE CARGA (SOC) E ESTADO DE SAÚDE (SOH) DE BATERIAS DE CHUMBO-ÁCIDO

Trabalho de Conclusão de Curso submetido ao Bacharelado em Engenharia Elétrica do Centro de Ciências Tecnológicas da Universidade do Estado de Santa Catarina, para a obtenção do Grau de Engenheiro Eletricista.

Orientador: Prof. Dr. Alessandro Batschauer

JOINVILLE

2017

GABRIELA PEREIRA DA SILVA

ESTUDO E APLICAÇÃO DE CIRCUITO INTEGRADO PARA ESTIMAÇÃO DO ESTADO DE CARGA (SOC) E ESTADO DE SAÚDE (SOH) DE BATERIAS DE CHUMBO-ÁCIDO

Trabalho de Conclusão de Curso apresentado ao curso de Engenharia Elétrica como requisito parcial para obtenção do título de Bacharel em Engenharia Elétrica.

Banca Examinadora

Orientador:

Prof. Dr. Alessandro Batschauer
UDESC

Membros:

Prof. Dr. Yales Rômulo de Novaes
UDESC

Prof. Dr. Sérgio Vidal Garcia Oliveira
UDESC

Joinville, 6 de Dezembro 2017.

AGRADECIMENTOS

Primeiramente a Deus, pela força e coragem de superar todas as adversidades durante esta longa caminhada. E que em todos os momentos foi o maior mestre que conheci.

Aos meus pais, Liliana Pereira da Silva e Luiz Eduardo Pereira da Silva pelo esforço e dedicação durante toda a minha vida. Pelo amor, incentivo e apoio incondicional para que eu superasse todas as dificuldades na busca dos meus sonhos. E por viverem comigo cada conquista e dividirem o peso dos momentos de tristeza.

Ao meu namorado, Gabriel Medeiros Restle, por toda a paciência nos momentos difíceis e estressantes. Por sempre acreditar no meu potencial e me proporcionar a certeza de que tudo daria certo.

Ao meu orientador, Alessandro Batschauer, por todo o conhecimento repassado, orientação e conselhos não só no período deste trabalho porém durante todo o meu tempo nesta universidade.

Aos membros do nPEE, em especial Marcos Vinicius Bressan, Felipe Joel Zimann e Rubens Hock, por estarem sempre disponíveis para ajudar. Compartilhando conhecimento e contribuindo de maneira indispensável para o meu crescimento e a elaboração deste projeto.

A todos os professores da Udesc, que de alguma forma se colocaram à disposição para sanar dúvidas e colaboraram imensamente no meu caminho até aqui.

A todos os meus amigos, pelos momentos de descontração proporcionados, os quais tornaram mais leve esta caminhada. Em especial, Beatriz Barros pelo apoio durante toda a faculdade e por sempre me mostrar um lado positivo nos acontecimentos. E Rodrigo Arturo Ramirez, por todas as risadas compartilhadas, as quais foram responsáveis por tornar os momentos complicados suportáveis.

RESUMO

O presente trabalho teve como objetivo destacar a importância dos sistemas de armazenamento de energia e seu controle. A crescente demanda de eletricidade, juntamente com a utilização de fontes renováveis, leva à necessidade de desenvolvimento de técnicas mais precisas de monitoramento e controle de baterias. O enfoque na química chumbo-ácido se deu em função de sua ampla difusão no Brasil e sua característica econômica de baixo custo. Aplicar processos controlados de carga e descarga nas células, geram vantagens não somente para o usuário como para o fabricante. Evitando sobretensões, picos de corrente e operação em temperatura elevada, a vida útil da bateria é prolongada. Isto gera mais confiabilidade ao sistema, e acarreta uma economia para ambas as partes. Pois, com um controle aprimorado, o sobredimensionamento dos projetos não se faz mais necessário. Neste trabalho, é apresentado o estudo de um circuito, posteriormente implementado em laboratório, para exemplificar um sistema de monitoramento. É utilizado um circuito integrado da *Texas Instruments*, o BQ34110, o qual através do método de estimação *Coulomb Counter* monitora os parâmetros da bateria e faz a estimação do estado de carga e estado de saúde da mesma. Suas características e funcionalidades são descritas, e alguns resultados apresentados.

Palavras-chave: Bateria, chumbo-ácido, armazenamento de energia, circuito integrado, monitoramento, controle.

ABSTRACT

This work aimed to highlight the importance of energy storage systems and their control. The increasing demand for electricity, coupled with the use of renewable sources, leads to the need of develop more precise techniques for monitoring and controlling batteries. The focus on lead-acid chemistry was due to its wide diffusion in Brazil and its low cost. Applying controlled charging and discharging processes to the cells, generate advantages not only for the user as for the manufacturer. By avoiding overvoltages, current peaks and operation with a high temperature, the lifetime of the battery is improved. This creates more reliability for the system. With improved control, over-sizing projects is no longer necessary, so both parts have an economic advantage. In this work it is presented also the study of a circuit, later implemented in the laboratory, to exemplify a battery monitoring system (BMS). The integrated circuit used is BQ34110 from Texas Instruments, which through the Coulomb Counter method estimates the state-of-charge and state-of-health of the batterie pack. Its features and functionalities are described, and some results exhibited.

Keywords: Battery, lead-acid, energy storage, integrated circuit, monitoring, control, battery monitoring system.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADC	*Analogic Digital Converter*
BMS	*Battery Management System*
CEDV	*Compensated End-of-Discharge Voltage*
CI	Circuito Integrado
EOS	*End-of-Service*
HDQ	*High-Speed Data Queue*
NTC	*Negative Temperature Coefficient*
OCV	*Open Circuit Voltage*
PSR	*Power Supply Rejection*
I^2C	*Inter-Integrated Circuit*
RTC	*Real Time Clock*
LED	*Light Emitting Diode*
TFT	*Thin-Film Transistor*
bps	bits por segundo
EDV	*End-of-Discharge Threshold*

SUMÁRIO

1 INTRODUÇÃO

Não é de hoje que o uso de energia elétrica deixou de ser sofisticação e se tornou uma necessidade na maioria dos setores de produção. Porém quanto mais o mundo fica dependente da eletricidade, mais se caminha para o fim das fontes de energia fósseis. Complementarmente, as fontes renováveis vêm ganhando destaque por sua característica inesgotável. Sendo assim, crescem simultaneamente os estudos para armazenamento da energia gerada.

A falta de energia não é mais apenas uma preocupação de setores onde há risco de vida, como por exemplo clínicas e hospitais. A interrupção repentina de eletricidade pode acarretar danos irreparáveis em máquinas caras no setor industrial, juntamente com o prejuízo oriundo da paralização da produção por tempo determinado. Indo além, muitos sistemas de segurança não conseguem se manter operantes por muito tempo sem alimentação. Logo, sendo o armazenamento de energia de suma importância e junto à ele sua supervisão.

Portanto, este trabalho tem por objetivo, explicitar como o monitoramento e controle desses sistemas de armazenamento são importantes. Prolongando a vida útil das baterias e aumentando a confiabilidade do sistema.

No capítulo 2 é apresentado o funcionamento da bateria, bem como os aspectos estruturais de cada tipo, com suas vantagens e desvantagens. Então é abordada mais a fundo a bateria de chumbo-ácido, pois foi a química escolhida para este trabalho. São exibidos diversos métodos de carga, enfatizando o objetivo de controlar sobretensões, picos de corrente e temperaturas elevadas. Logo após, é apresentada a modelagem da bateria, a qual é fundamental para que possam ser feitas boas estimações de seus indicadores e um dimensionamento mais exato dos componentes. A seguir é possível encontrar a importância de monitorar e controlar o estado de carga das baterias e os diversos métodos para realizar este monitoramento.

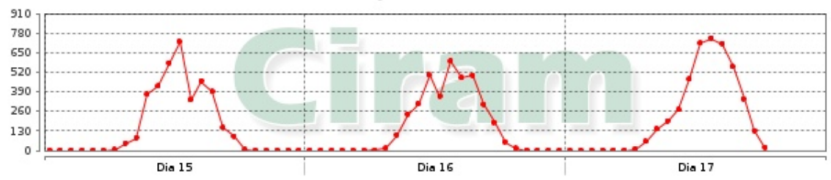
No capítulo 3 são retratadas as características do circuito integrado selecionado para a implementação deste trabalho. Mostra-se o circuito da placa confeccionada, a conexão realizada em laboratório e a utilização do CI. A comunicação é feita através de I^2C, então neste capítulo este protocolo é abordado. É explicitado também como acessar as posições de memória do dispositivo através do microcontrolador arduíno e então detalhadas as funções de seus algoritmos internos para realizar as medições e estimações. Por fim, são apresentadas as configurações feitas em laboratório e alguns resultados obtidos.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 MOTIVAÇÃO

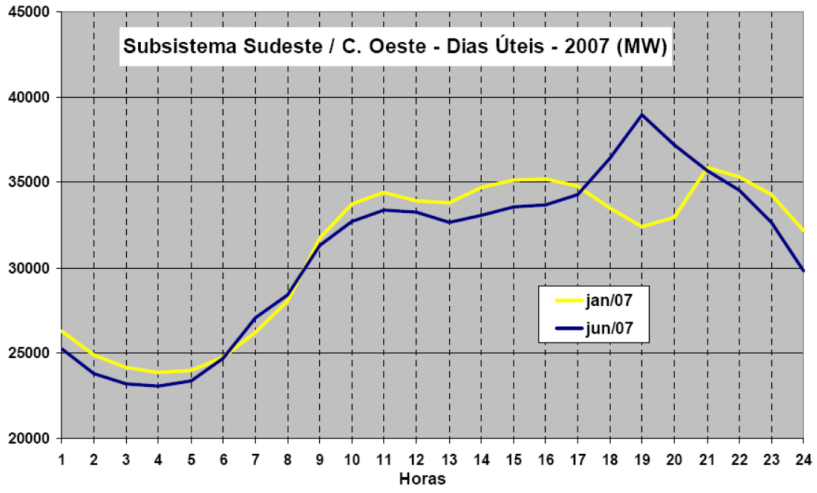
A partir da Primeira Revolução Industrial, com a descoberta do carvão e, mais tarde a utilização do petróleo, um rápido desenvolvimento tecnológico foi alcançado, gerando uma dependência dos combustíveis fósseis. Porém, após a crise do petróleo, o mundo percebeu como os combustíveis fósseis eram finitos e tinham um limite que poderia ser imposto por razões políticas, além das naturais. Então, iniciou-se a busca por fontes alternativas de energia a fim de diminuir essa dependência (BASTOS, 2013). Assim, com o aprimoramento das tecnologias e a inserção das fontes de energia renováveis na sociedade, os problemas de produção de energia foram ganhando soluções. Mesmo que ainda devagar, a sociedade caminha para um uso cada vez maior de fontes renováveis, chamadas fontes de energia limpa. Entretanto juntamente com este avanço um novo problema surgiu. As fontes renováveis não estão disponíveis para gerar energia o dia todo e o pico de produção de energia raramente coincide com o pico da demanda de energia. Por exemplo a energia solar não é uma fonte de energia disponível durante todo o dia. Pela Figura 2.1 nota-se que a maior incidência de luz é por volta do meio dia, e pela Figura 2.2 nota-se que a maior demanda ocorre em torno das 18 às 21 horas no inverno e das 14 às 17 horas no verão. Para resolver este problema foi preciso desenvolver uma maneira de armazenar esta energia produzida para ser utilizada nos momentos de pico da demanda.

Figura 2.1 – Radiação absorvida por um conjunto de painéis fotovoltaicos durante um dia típico.



Fonte: https://goo.gl/rVcXAt

Figura 2.2 – Curva de Carga obtida no verão e inverno Região Sudeste do Brasil



Fonte: https://goo.gl/pFnEsm

Além da motivação citada acima, segundo (COELHO, 2001) os equipamentos elétricos fazem parte atualmente das necessidades básicas para manutenção da vida e ordem da sociedade. Sendo assim, há alguns setores em que o corte repentino no fornecimento de energia sofre grandes consequências. Alguns exemplos seriam os hospitais, onde não pode existir a opção da falta de eletricidade em um centro cirúrgico durante uma operação. Por outro lado, do ponto de vista econômico, em certas empresas a falta de eletricidade pode danificar equipamentos muito caros, trazendo grande prejuízo para a produção. Portanto, muitas vezes é extremamente necessário um investimento em fontes de alimentação ininterrupta, como por exemplo um sistema de no-break.

Na literatura (FAKHAM; LU; FRANCOIS, 2011) são apresentadas duas das formas de armazenamento, através de banco de baterias ou ultra capacitores. As vantagens do primeiro consistem no fato de que sua densidade é maior e seu custo reduzido. Porém, o segundo consegue entregar uma corrente muito mais elevada, a qual reduz significativamente a vida útil das baterias e aumenta suas perdas. O banco de baterias possui uma maior capacidade de fornecimento de energia, enquanto que os ultra capacitores um fornecimento de potência.

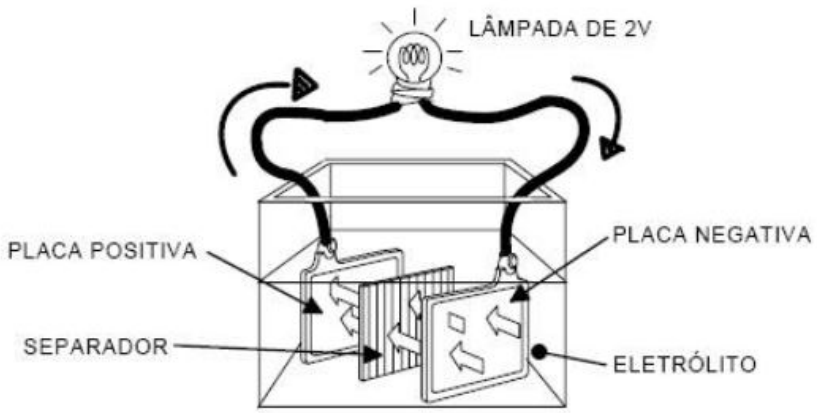
2.2 ASPECTOS GERAIS

Em 1799 começaram os estudos que mais tarde deram origem às baterias. Aproximadamente 60 anos depois, em 1859, Gaston Planté fez uma demonstração prática de um sistema composto por duas tiras de chumbo, separadas por fitas de borracha e mergulhadas numa solução de 10% de ácido sulfúrico. Passado determinado tempo, tinha-se uma célula com 2 volts.

(CHAGAS, 2007) Porém até o ano de 1886 não tinha sido encontrada uma aplicação e essa experiência tinha permanecido apenas como curiosidade de laboratório. Foi a partir do ano citado acima, que então inúmeras aplicações foram descobertas e várias pesquisas focadas no desenvolvimento de novos processos de fabricação e teorias de funcionamento.

A bateria é um elemento armazenador de energia elétrica na forma eletroquímica. Devido às reações químicas que ocorrem dentro da mesma, armazena ou fornece a energia nela armazenada podendo ser na forma de tensão ou corrente contínua. Em baterias que são recarregáveis este processo pode ser repetido várias vezes. (COELHO, 2001) Seu funcionamento consiste no eletrodo negativo, ou ânodo, ceder elétrons durante a descarga, através do processo eletroquímico chamado oxi-redução (como pode ser visto na Figura 2.3). Estes elétrons fluem através da carga conectada à bateria, cedendo energia. Transportados então para o eletrodo positivo, ou cátodo, onde os elétrons sofrem a redução eletroquímica. No caso do carregamento da bateria esse processo é inverso. (DEKKA et al., 2015)

Figura 2.3 – Demanda de Energia



Fonte: http://portal.macamp.com.br/portal-conteudo.php?varId=79

Baterias são associações em série ou paralelo de células unitárias, permitindo assim obter-se valores de tensão ou corrente desejados. Por permitir acumular energia, conservá-la e restituí-la mais tarde, a bateria representa uma fonte autônoma de energia. Segundo (CHAGAS, 2007) as baterias podem ser separadas em duas categorias: baterias primárias e baterias secundárias. As baterias primárias produzem energia a partir de uma reação química irreversível, inutilizando-as depois. Ou seja, não podem ser recarregadas e fazer esse processo mais de uma vez. Já as baterias secundárias podem ser carregadas e descarregadas diversas vezes.

Existem distintos tipos de bateria, feitos com diferentes compostos. Existem tipos de baterias que possuem uma alta densidade de armazenamento de energia, porém isso acaba implicando em uma desvantagem no custo, que fica mais elevado. Além disso, em alguns casos, estes tipos de baterias necessitam grande segurança ou fornecem riscos maiores ao meio am-

biente. A escolha do tipo de bateria a ser utilizada deve ser uma combinação de desempenho, vida útil, custo, segurança e mínimo impacto ambiental (COELHO, 2001). Sendo assim, cada tipo de bateria possui suas próprias características, vantagens, desvantagens e aplicações.

Nenhuma bateria, independente de seu tipo, tem uma eficiência de 100%. Isso se deve ao fato de que no processo de armazenamento ou fornecimento de energia, uma parte é sempre perdida na forma de calor. Dessa forma, cargas e descargas mais lentas possuem uma maior eficiência, pois o fluxo de corrente por hora não precisa ser tão alto e assim, aquecendo menos (o que implica em perdas menores).

Na literatura (COELHO, 2001) existem duas classes para divisão dos tipos de baterias. Estas podem ser divididas conforme a finalidade de sua utilização ou como são construídas. Dentro da primeira opção, tem-se aplicações em automóveis, transporte marítimo e baterias para ciclo profundo. Já na segunda, as principais formas de construção são com fluido, gel ou fibra de vidro. Quanto maior a descarga requerida no tipo de bateria, mais grossas são as placas que a compõem. Por outro lado, essa alteração para obter maior potência tem desvantagens como a diminuição da densidade de energia e da durabilidade da bateria.

Na aplicação automotiva, as baterias são utilizadas principalmente para ignição ou partida dos motores. Neste caso é requerido por um curto período de tempo, uma elevada corrente. Por este motivo, estas baterias podem ser gravemente danificadas quando descarregadas completamente (COELHO, 2001).

Já na aplicação marítima, as baterias são uma mistura das características das automotivas com as de ciclo profundo. E estas, por sua vez, podem ser descarregadas até no máximo 50% de sua carga total (COELHO, 2001).

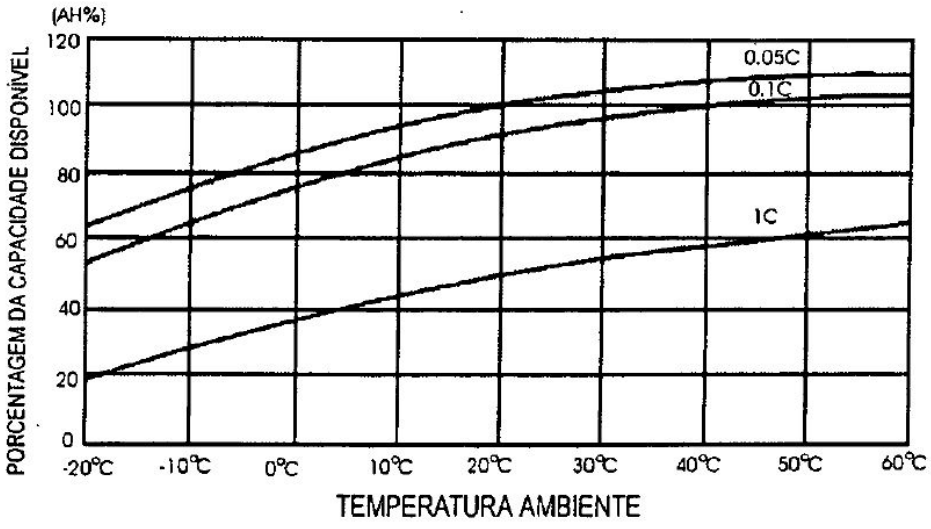
As baterias de ciclo profundo podem ter até 80% da sua carga descarregada, sendo assim, suas placas são muito mais grossas que as placas dos dois tipos citados anteriormente. Sua aplicação consiste em empilhadeiras elétricas, veículos de tração ou baterias estacionárias (alimentação de emergência como por exemplo fontes ininterruptas de energia). Normalmente este tipo de bateria pode ser descarregado até no máximo 1000 vezes durante sua vida útil, porém este número depende da profundidade de descarga (COELHO, 2001). Uma bateria que é descarregada em 50% dura aproximadamente duas vezes mais que uma bateria que é descarregada 80%. Todavia, para se ter uma profundidade de descarga menor, é necessário que a bateria tenha uma capacidade elevada em relação a capacidade requerida pela aplicação (COELHO, 2001). Ao passar muito tempo sem utilização, a bateria acaba perdendo parte de sua capacidade. Então é necessário manter uma tensão de flutuação nos terminais das baterias para evitar este dano.

Com respeito ao aspecto construtivo, as baterias de gel possuem esse nome pois seu ácido foi transformado em gel ao acrescentar sílica em gel. Este tipo de bateria possui uma maior segurança na manutenção, visto que não há possibilidade de espirrar ácido, como acontece na

forma líquida. Entretanto sua desvantagem consiste em ter uma descarga mais lenta para não haver a criação de gás, em função disso não fornecem uma elevada corrente. As baterias de meio sólido tem suas características muito semelhantes às das baterias em gel. E por fim, as baterias de meio líquido possuem válvulas para regulação da pressão interna e quando recarregadas muitas vezes podem perder água, o que as deixa inúteis ou requerem manutenção. Para resolver esse problema algumas possuem capas especiais que convertem hidrogênio e oxigênio novamente em água, reduzindo a perda de água e aumentando sua vida útil (COELHO, 2001).

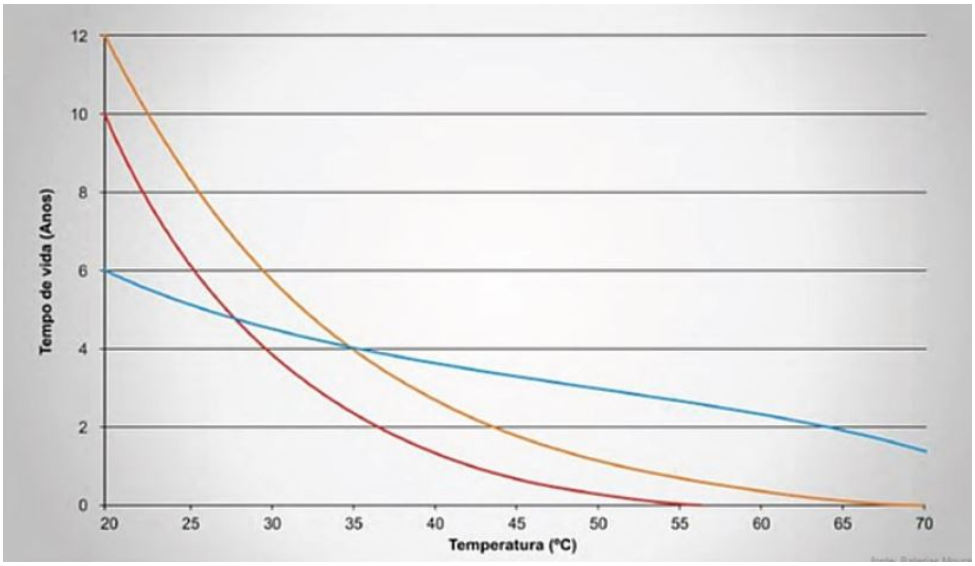
Para a escolha do tipo ideal de bateria para determinada aplicação também é muito importante analisar suas características elétricas. Dentre as quais, se enquadram a capacidade do fornecimento de corrente (ou seja, a capacidade da bateria) e a tensão nos seus terminais. Fatores externos como a temperatura influenciam de forma significativa na capacidade da bateria e na sua vida útil. Com temperatura reduzida, a capacidade da bateria sofre uma queda. Porém em altas temperaturas a vida útil é diminuída. Para as baterias de Chumbo-Ácido por exemplo, sua capacidade é reduzida em 50% a -30°C e sua vida útil cai pela metade para cada 8.5°C acima de 24°C (COELHO, 2001). Nas Figuras 2.4 e 2.5 pode-se ver a curva de capacidade e durabilidade das baterias.

Figura 2.4 – Capacidade das Baterias



Fonte: Adaptado de (COELHO, 2001)

Figura 2.5 – Variação da vida útil de diferentes tecnologias de bateria com a variação da temperatura



Fonte: (MOURA, 2011)

Segundo (JOSEPH; M.SHAHIDEHPOUR, 2006), alguns tipos de baterias disponíveis no mercado seriam: Chumbo-Ácido, Sódio-Enxofre, Zinco-Bromo e Íon de Lítio.

2.2.1 Bateria de Lítio

Este tipo de bateria tem seu cátodo feito de óxido de lítio e seu ânodo de grafite de carbono. O eletrólito é composto por sais de lítio dissolvidos em carbonato orgânico (DEKKA et al., 2015). Suas vantagens são uma alta densidade de energia, com uma eficiência perto de 100% e um longo ciclo de vida. Sua maior desvantagem, e a única que priva este tipo de bateria de dominar o mercado é seu custo, um tanto elevado em função da necessidade de uma embalagem especial e proteções internas para os circuitos (JOSEPH; M.SHAHIDEHPOUR, 2006).

2.2.2 Bateria de Sódio-Enxofre

Este tipo de bateria possui um eletrólito sólido, formado por alumina, o qual separa o ânodo do cátodo. O primeiro é constituído por enxofre líquido enquanto o segundo por sódio, também na forma líquida (JOSEPH; M.SHAHIDEHPOUR, 2006) (DEKKA et al., 2015). Sua eficiência é alta, porém um pouco menor que a das baterias de lítio, ficando em torno de 89%. Seu uso é predominante no Japão, onde é muito difundida em aplicações para armazenar energia a ser utilizada nas horas de pico (JOSEPH; M.SHAHIDEHPOUR, 2006).

2.2.3 Bateria Zinco-Bromo

Neste tipo de bateria dois eletrólitos diferentes fluem por um eletrodo de carbono plástico em dois compartimentos separados por uma membrana micro porosa. A eficiência desta bateria fica em torno de 75% (JOSEPH; M.SHAHIDEHPOUR, 2006).

2.2.4 Bateria de Chumbo-Ácido

O último de tipo de bateria a ser descrito neste trabalho e o tipo que será dado enfoque, é a bateria de Chumbo-Ácido. No começo do século XX, em função do começo do desenvolvimento da indústria automotiva, as baterias de chumbo-ácido se tornaram o alvo principal de inúmeras pesquisas, pois tinham finalmente uma aplicação promissora e em expansão (CHAGAS, 2007). Os primeiros veículos criados pela General Motors, bem como Ford e outras companhias utilizavam baterias de Chumbo-Ácido e atualmente mais de 90% dos automóveis ainda utilizam este tipo de bateria. Sua função consiste na partida dos motores automotivos (ignição), assim como iluminação. (COELHO, 2001). Hoje, suas aplicações estão se diversificando devido à necessidade de acumulação de energia no sistema de geração de eletricidade por fontes renováveis, entre outros motivos (CHAGAS, 2007). Dessa forma, o objetivo das pesquisas antigas (diminuir o custo em detrimento da qualidade das células) mudou. Com a ampliação das aplicações, técnicas de controle estatístico de qualidade foram aplicadas para aumentar a qualidade das baterias.

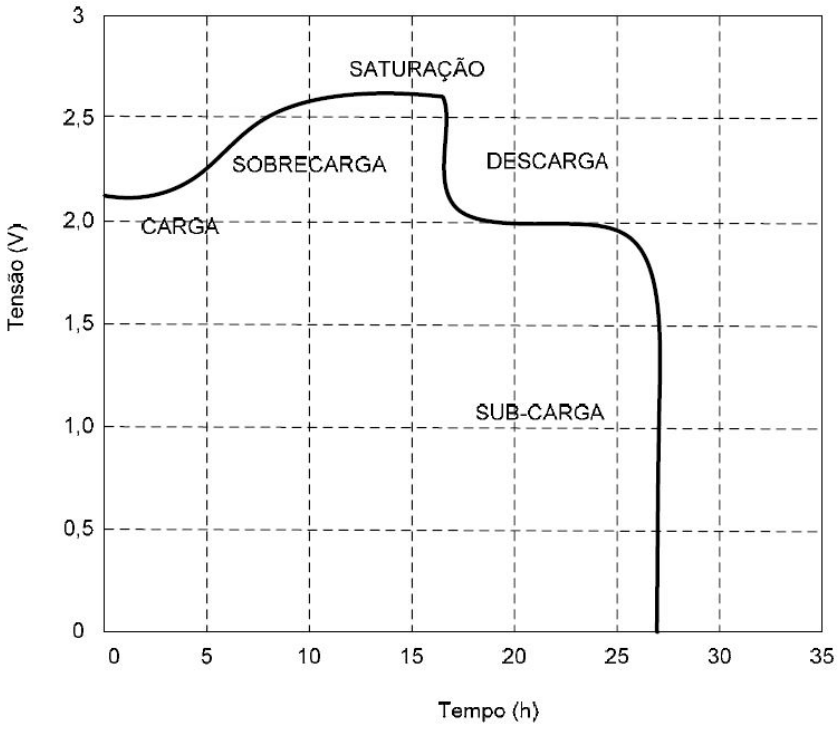
Esta bateria é formada por duas placas imersas em uma solução diluída de ácido sulfúrico. Seu cátodo é feito de dióxido de chumbo e seu ânodo feito de chumbo. Podendo operar em dois modos (carga ou descarga), a bateria tem sua reação total descrita abaixo (FRAGA, 2009).

$$PbO_2 + Pb + 2H_2SO_4 tle \longleftrightarrow 2PbSO_4 + 2H_2O$$

Da esquerda para a direita é o modo de descarga e da direita para a esquerda é o modo de carga da bateria. No primeiro modo, a corrente flui para fora do terminal positivo e a tensão da bateria diminui. O oposto ocorre no segundo modo, onde a corrente flui para a bateria pelo terminal positivo, aumentando lentamente a tensão da bateria (FRAGA, 2009). Outros dois modos devem ser considerados, modos tais que danificam a célula. O modo de sub-carga e sobrecarga. O primeiro é alcançado quando a carga da bateria está próxima do mínimo valor recomendado mas o circuito exige uma situação de além descarga. Esse estado resulta em uma diminuição da densidade interna originando uma sedimentação no fundo dos elementos da bateria. O segundo ocorre quando a carga da bateria excede o valor máximo recomendado.

Então dois efeitos podem ser originados: a produção de gás venoso e a corrosão do material no eletrodo positivo. Ambos estados danificam irreversivelmente a célula (FRAGA, 2009). Na Figura 2.6 pode-se ver os modos de operação das baterias.

Figura 2.6 – Modos de operação baterias



Fonte: Adaptado de (FRAGA, 2009)

Este tipo de bateria é o mais utilizado nos sistemas de armazenamento por ser o economicamente mais viável (visto que seus elementos para fabricação - chumbo e ácido sulfúrico - são baratos) e altamente difundido no Brasil. Tem sua desvantagem baseada na necessidade de manutenção e no peso elevado. Adicionalmente, quando comparada à bateria de Lítio por exemplo, possui um ciclo de vida mais curto, o qual ainda pode ser reduzido caso seja descarregada abaixo de 30% de sua capacidade total (a vida útil reduz aos poucos desde quando as baterias tem sua capacidade reduzida em 80%, porém os números ficam mais críticos quando ultrapassa 30%). Suas condições externas de trabalho também influenciam muito seu desempenho e durabilidade, como já foi citado.

As baterias de Chumbo-Ácido ainda podem ser dividas em duas categorias, conforme a literatura (CHAGAS, 2007). A primeira categoria é a de baterias ventiladas (FVLA). É a forma mais comum das baterias de Chumbo-Ácido, porém possui uma grande desvantagem que consiste no fato de devido à emissão de gases perigosos, devem ser instaladas em salas exclusivas com sistemas especiais de segurança (a prova de explosão). Sua vida útil fica em torno de 15 anos quando armazenada numa temperatura de 25°C. O fim de sua vida útil ocorre

quando sua capacidade atinge 80% da nominal (CHAGAS, 2007). A Figura 2.7 mostra um modelo de bateria ventilada.

Figura 2.7 – Bateria Ventilada

Fonte: Adaptado de (CHAGAS, 2007)

A segunda categoria é a de baterias reguladas por válvulas (VRLA), podendo ser com eletrólito gel ou eletrólito absorvido. Estudos possibilitaram essas baterias operarem em qualquer posição, pois o líquido é retido nos separadores ou no gel. Com grande redução na perda de água, essas baterias são utilizadas principalmente em aplicações estacionárias como sistemas de telecomunicações e no-breaks. Continuando com o baixo custo tradicional das baterias de Chumbo-Ácido e agora solucionando o problema de requerer muita manutenção, as baterias seladas, como podem ser chamadas, são a melhor escolha em inúmeras aplicações (CHAGAS, 2007).

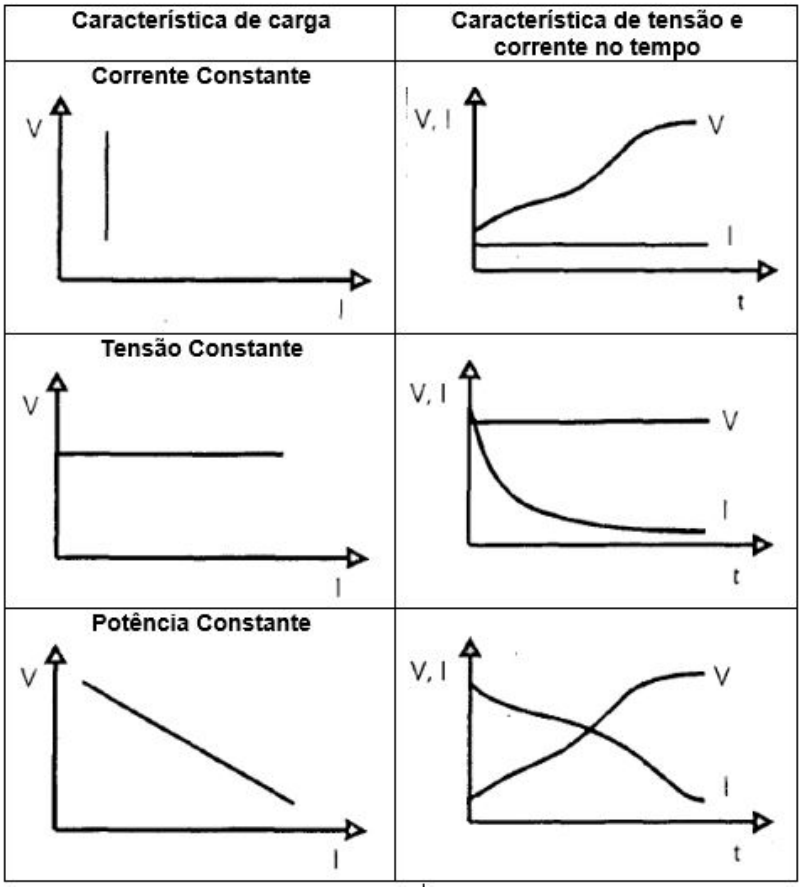
2.2.5 Métodos de Carga

Como o enfoque deste trabalho é a bateria de Chumbo-Ácido, optou-se por trabalhar apenas com métodos de carga para essas baterias. Segundo a literatura (COELHO, 2001), os métodos de carga variam conforme o tipo de carga que se deseja efetuar. Para realizar um carregamento adequado é necessário cuidar com alguns fatores, como por exemplo monitorar a corrente, para que não ultrapasse o máximo suportável, assim como a temperatura da bateria. Caso a carga esteja ligada diretamente à célula, então é necessário monitorar a tensão de flutuação a fim de evitar sub-tensão.

Conforme (COELHO, 2001), de modo geral pode-se classificar três métodos de carga. Seriam estes por corrente constante, tensão constante ou potência constante. O método da corrente constante é considerado o que apresenta melhores resultados, visto que a corrente é controlada, implicando num controle da temperatura. Entretanto, é necessário monitorar sua tensão para que não ultrapasse a tensão máxima permitida pelo fabricante. O método de tensão

constante pode ser aplicado apenas por um curto intervalo de tempo e sempre tendo sua temperatura monitorada, já que atinge correntes elevadas. Por um breve período, também o método de potência constante pode ser utilizado. Controlando a temperatura, a diferença deste método para o anterior é o fato de que neste não ocorre redução da potência à medida que o carregamento aumenta. Sendo assim suas perdas são maiores, pois a potência total injetada é maior. Na Figura 2.8 mostra-se o comportamento de tensão e corrente no tempo desses 3 métodos.

Figura 2.8 – Métodos de Carga



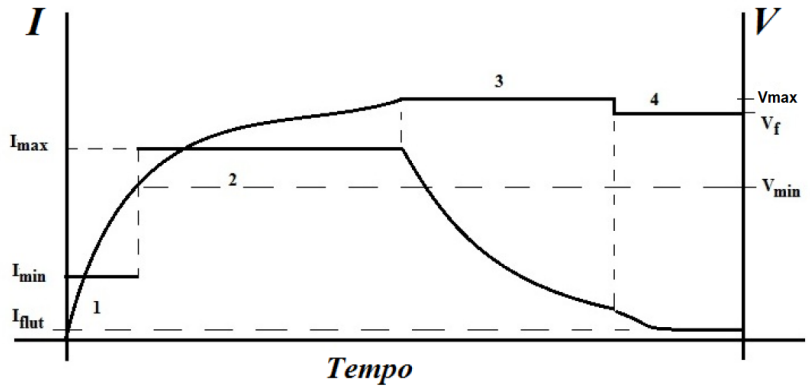
Fonte: Adaptado de (COELHO, 2001)

A aplicação do método de corrente constante consiste em carga parcial, completa e sobrecarga. O método de tensão constante somente é permitida por pequenos intervalos de tempo e com supervisão da temperatura. E o método de potência constante também só pode ser aplicado por pequenos intervalos de tempo e a corrente deve ser limitada de forma que seja absorvida pela bateria sem aquecimento excessivo.

Ainda pode-se fazer uma combinação de dois desses métodos, originando vários outros. Dentre os quais, se enquadram por exemplo o método de carga com duplo nível de tensão (COELHO, 2001). Este método é a junção da corrente constante com a tensão constante, alternando ambos. Seu objetivo é utilizar as melhores características de cada um através de três estágios.

No primeiro (quando a bateria está em um nível mínimo de energia) é aplicada uma corrente constante mínima I_{min} até atingir um nível mínimo de tensão V_{min}. Isso ocorre para evitar picos de corrente no início da carga. Ao ser alcançado então V_{min}, é injetada a corrente de carga desejada I_{max}. Em função da injeção de uma corrente muito alta, a bateria alcança a máxima tensão permitida na bateria V_{max}, a qual é fixada como a tensão de sobretensão na célula. Após esse processo, a corrente de carga inicia um processo de queda à medida que a bateria se aproxima da carga completa. Quando a carga é completa é necessário apenas compensar a auto descarga aplicando uma tensão de flutuação V_f (até a bateria entrar em operação). Este processo pode ser visto na Figura 2.9:

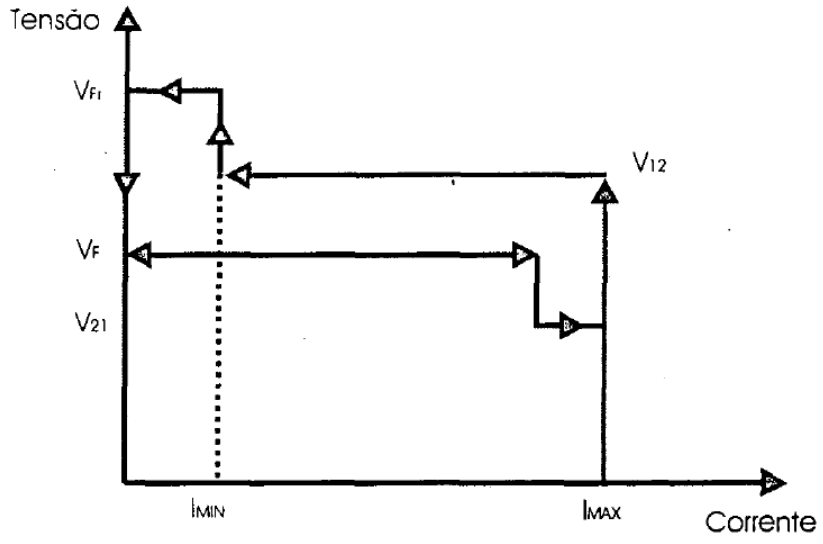
Figura 2.9 – Método Dois níveis de Tensão



Fonte: Adaptado de (COELHO, 2001)

Outro método resultante da mistura dos métodos originais é o método a dois níveis de corrente e um nível de tensão (COELHO, 2001). Método aplicado essencialmente quando se tem um banco de baterias, possui dois estágios. No primeiro, uma corrente equivalente a 10% da corrente nominal é aplicada até atingir uma tensão um pouco maior que a tensão nominal, porém sem atingir a sobretensão. No segundo estágio então, aplica-se uma tensão constante até a corrente atingir um valor próximo a 5% do valor nominal. Por fim, essa corrente é mantida constante até a tensão se estabilizar. Por conseguinte, a corrente se anula e a tensão passa para um nível de flutuação até entrar em operação (COELHO, 2001).Este método pode ser visto na Figura 2.10.

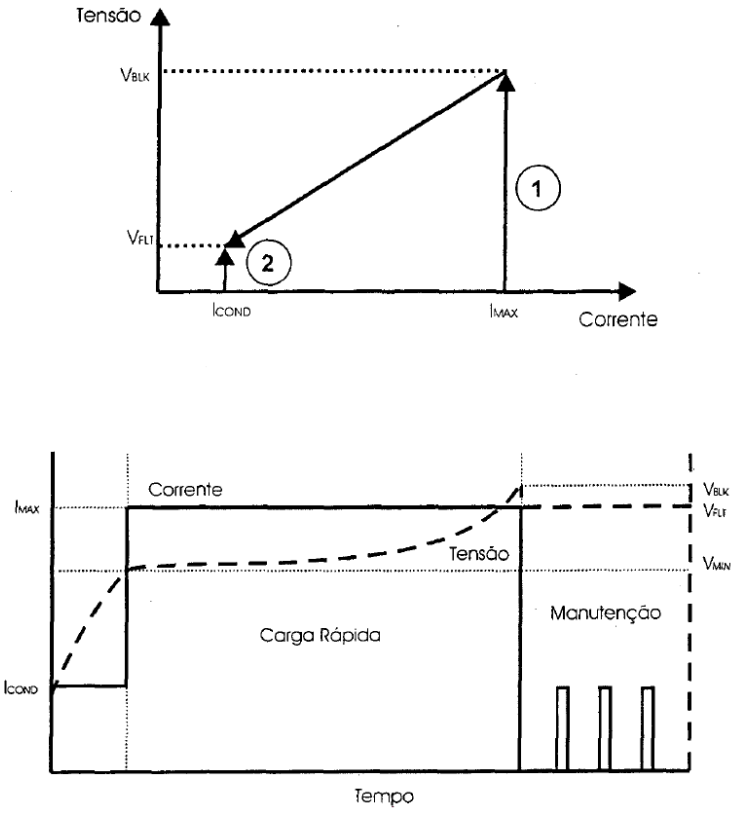
Figura 2.10 – Método dois Níveis de Corrente e um de Tensão



Fonte: Adaptado de (COELHO, 2001)

Um terceiro método oriundo da combinação dos outros é o método a dois níveis de corrente (COELHO, 2001). Aplicando dois níveis de corrente, sendo o primeiro para que a carga das células de bateria atinjam 90% de sua capacidade e o segundo para realizar a compensação da auto descarga da placa positiva. O problema deste método consiste na possibilidade de impor uma corrente de retenção muito baixa ou muito alta, acarretando uma sobrecarga ou perda de capacidade da bateria. A solução seria monitorar a tensão da bateria o tempo todo e ajustar a corrente para seu valor mínimo. A Figura 2.11 exemplifica esse método.

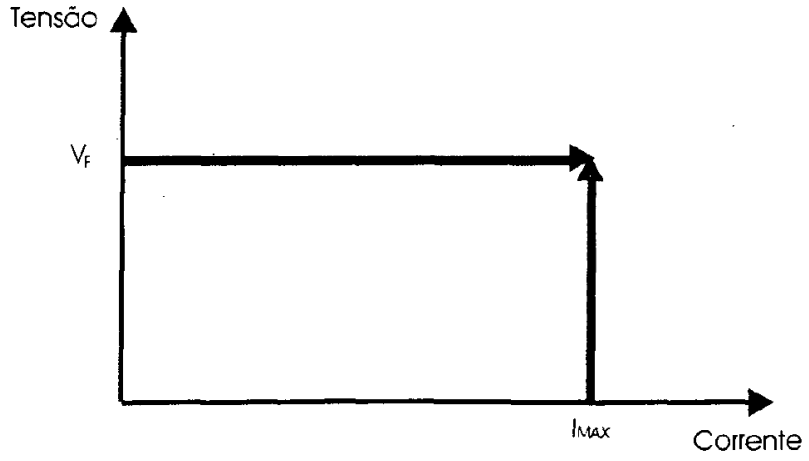
Figura 2.11 – Método dois níveis de Corrente



Fonte: Adaptado de (BASTOS, 2013)

O penúltimo método a ser mencionado neste trabalho é o método a um nível de corrente e um nível de tensão (COELHO, 2001). Primeiro a corrente é mantida constante até atingir uma certa tensão de equalização, quando então o valor de tensão é regulado para uma tensão de flutuação (a qual tem por objetivo proteger da auto descarga, enquanto a bateria não está em operação). Pode-se ver este método na Figura 2.12.

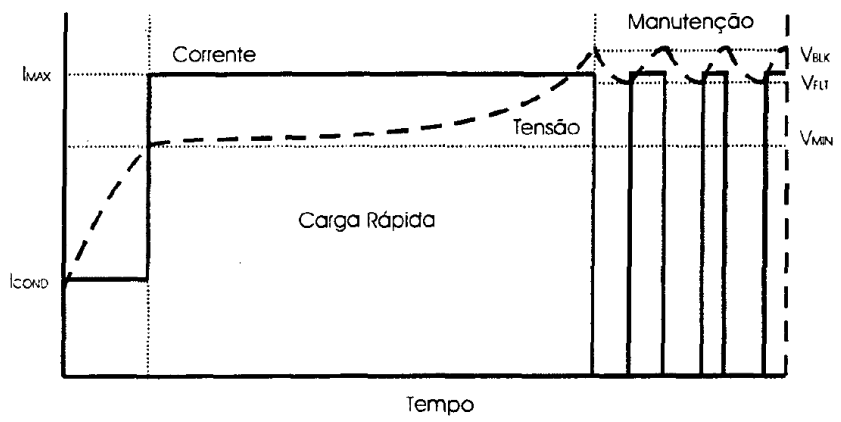
Figura 2.12 – Método um Nível de Corrente e um Nível de Tensão



Fonte: Adaptado de (COELHO, 2001)

Por fim, tem-se o método de equalização com corrente pulsante (COELHO, 2001). Através de um relé liga/desliga, a tensão nos terminais da bateria é controlada. A lógica consiste em carregar a bateria até sua tensão final, através da auto descarga a tensão é reduzida lentamente, e quando esta chegar na tensão de flutuação a recarga é automaticamente reativada, crescendo até a tensão final novamente e recomeçando o ciclo. Sua desvantagem encontra-se no fato de que sua tensão final pode ser alterada por fatores externos como a temperatura, fazendo a bateria sofrer com a sobrecarga, por exemplo (COELHO, 2001). Este método pode ser visto na Figura 2.13.

Figura 2.13 – Método Corrente Pulsante



Fonte: Adaptado de (COELHO, 2001)

2.2.6 Modelagem da Bateria de Chumbo Ácido

A modelagem da bateria de Chumbo-Ácido é fundamental para que se possam ser feitas boas estimações dos seus indicadores, além de um melhor conhecimento do comportamento da bateria e um melhor dimensionamento de componentes eletrônicos. Dessa maneira, a queda de tensão devido a corrente de disparo pode ser compensada (ARAUJO et al., 2010). Para fazer a modelagem de uma bateria existem diversos circuitos, indo dos mais simples aos mais complexos, cada um possuindo suas características, vantagens e desvantagens. Para modelar uma bateria, há outras maneiras além dos circuitos equivalentes, como por exemplo o modelo eletroquímico. Este modelo é um dos que consegue fornecer os resultados mais precisos, entretanto sua complexidade é tão alta que inviabiliza sua implementação na maioria dos casos. É composto por várias equações diferenciais parciais, e é dependente de diversas constantes químicas, como a taxa de difusidade, concentração molar do eletrólito, raio das partículas, entre outras. Outras duas formas de fazer a modelagem é a modelagem estocástica (através de análises estatísticas) e a do tipo "Caixa Preta", ou Black-Box como é mais conhecida. Porém também essas maneiras necessitam de parâmetros obtidos experimentalmente das baterias, dificultando sua implementação, além de individualizar cada modelo (pois o modelo só vale para as células das quais foram realizados os experimentos), juntamente com o fato de que os parâmetros obtidos só valem para as condições aplicadas (REMES; OLIVEIRA, 2014). Sendo assim, a modelagem por circuitos elétricos (Electric Circuit Model - ECM) torna-se a maneira de maior custo benefício para representar as parcelas dinâmicas e estáticas da bateria. Atingindo um equilíbrio bom entre complexidade e acurácia, é a modelagem mais utilizada.

Modelar uma bateria tem como seu objetivo principal possibilitar o monitoramento e a simulação dos parâmetros da bateria durante a sua descarga. Sendo assim, é necessário que o modelo satisfaça 3 requisitos. O primeiro seria permitir a medição da tensão nos terminais, a temperatura da célula, o estado de saúde e a resistência interna da bateria, para que o gerenciamento da bateria seja feito de forma adequada. O segundo é que a complexidade do modelo seja suficiente para respeitar o requisito anterior, mas que também possa ser executável em tempo real com sistema de gerenciamento. E por último, que a saída dos parâmetros do modelo da bateria possa ser entendida por pessoas que não tenham um conhecimento técnico aprofundado (GUO, 2010).

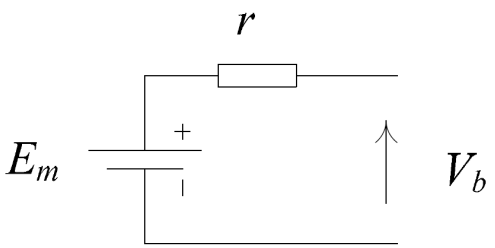
O circuito equivalente corresponde a forma mais simples de representar um circuito que contém as características elétricas do original. Normalmente, é composto por elementos lineares e passivos. Porém, circuitos mais complexos podem ser usados para aproximar o comportamento não linear dos circuitos originais (GUO, 2010). Vários estudos acabaram por desenvolver diferentes modelos de circuitos equivalentes, com diferentes níveis de complexidade,

satisfazendo assim inúmeras aplicações. Seis destes modelos serão apresentados a seguir neste trabalho.

2.2.6.1 Modelo Simples

Segundo a literatura (GUO, 2010) este é o tipo de modelagem mais simples e pode ser visto na Figura 2.14. Este modelo consiste em uma fonte ideal E_m que representa a bateria e um resistor r conectado em série, o qual equivale à resistência interna da bateria. A tensão nos terminais da bateria é indicada por V_b.

Figura 2.14 – Modelo Simples



Fonte: Adaptado de (GUO, 2010)

Este modelo é pouco utilizado pois não leva em consideração a variação característica da resistência interna para os diferentes níveis de carga e não tem a informação energética. (GUO, 2010)

2.2.6.2 Modelo Simples Avançado

Como o próprio nome já diz, este modelo é uma melhora do modelo citado anteriormente. Sua única diferença encontra-se na utilização de um resistor variável, dependente do estado de carga (SOC) (GUO, 2010). A equação responsável pela relação entre a resistência e o SOC é mostrada na equação (2.1).

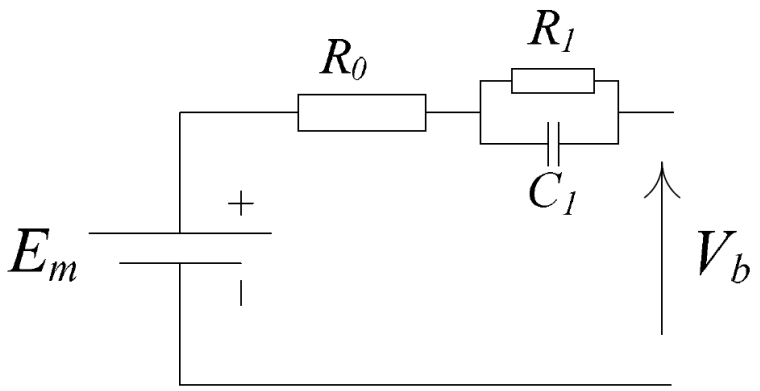
$$r = \frac{R_0}{SOC^a} \tag{2.1}$$

Onde R_0 é a resistência da bateria quando completamente carregada, "a" é o coeficiente de capacidade e o SOC o estado de carga, o qual varia de 0 (totalmente sem carga) a 1 (carga completa). Sua desvantagem porém, está no fato de não considerar as perdas que a bateria pode sofrer nem os comportamentos transientes da bateria (GUO, 2010).

2.2.6.3 Modelo de Thevénin

Conforme (GUO, 2010), este modelo é composto por uma fonte E_m, uma resistência interna R_0, uma resistência de sobretensão R_1 (a qual representa contato entre a placa e o eletrólito da bateria) e uma capacitância C_1. Podendo ser visto na Figura 2.15.

Figura 2.15 – Modelo Thevenin



Fonte: Adaptado de (GUO, 2010)

Sua grande desvantagem consiste em assumir seus parâmetros como sendo constantes, quando na realidade são extremamente dependentes do SOC e outras características de descarga (GUO, 2010).

Para resolver este problema foi desenvolvido o modelo dinâmico. o qual leva em consideração as características não lineares da tensão de circuito aberto e resistência interna. Através da equação (2.2).

$$V_b = V_{oc} - (R_b + \frac{K}{SOC})\, I \tag{2.2}$$

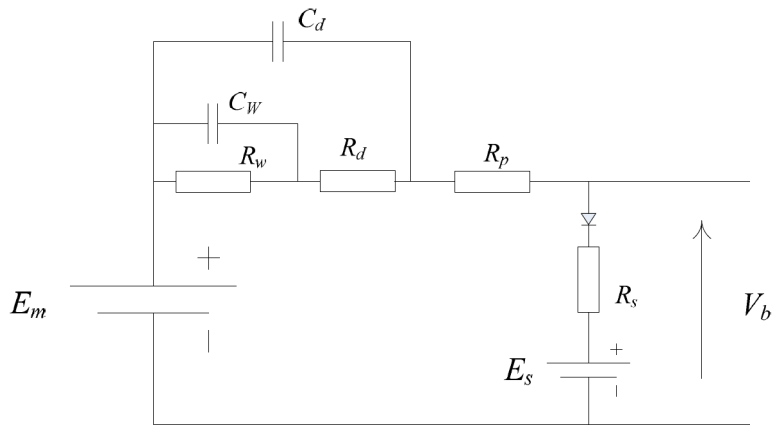
Onde V_b é a tensão no terminal da bateria, V_{oc} é a tensão de circuito aberto, R_b é o resistor do terminal da bateria, usualmente uma grandeza em torno de 0.1Ω , K é a constante de polarização e I a corrente de descarga (GUO, 2010).

2.2.6.4 Modelos de Ordem Mais Elevada

Os modelos de maior ordem são mais sofisticados, e desta forma mais precisos. Levam em consideração fatores como os parâmetros serem dependentes do SOC e as características não-lineares do sistema. O modelo originário dos modelos de ordem maior, é o modelo dinâmico de quarta ordem (GUO, 2010), o qual é mostrado na Figura 2.16. Composto de uma associação em paralelo de resistores e capacitores, sua principal desvantagem sendo a obten-

ção dos parâmetros, juntamente com o fato de ter equações de quarta ordem, complicando sua resolução.

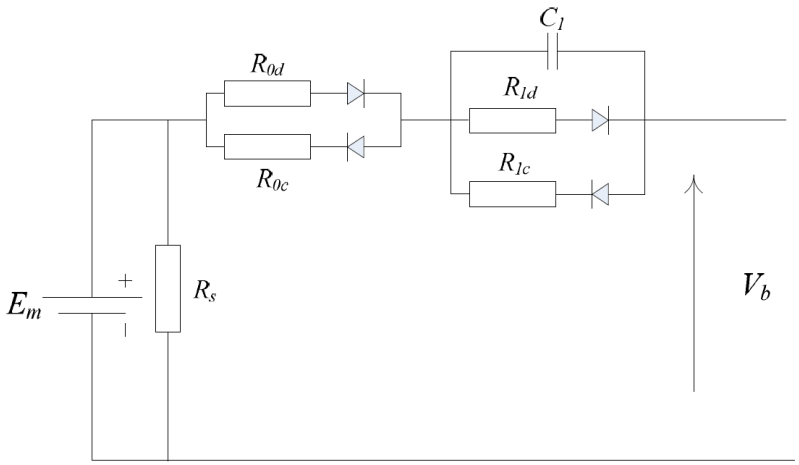
Figura 2.16 – Modelo Quarta Ordem



Fonte: Adaptado de (GUO, 2010)

Baseado neste modelo foi criado o Modelo Avançado. Sua diferença está nos diodos inseridos em série com os resistores, para representar a carga e descarga da bateria e pode ser visto na Figura 2.17 (GUO, 2010).

Figura 2.17 – Modelo Avançado



Fonte: Adaptado de (GUO, 2010)

2.2.6.5 Modelo Matemático

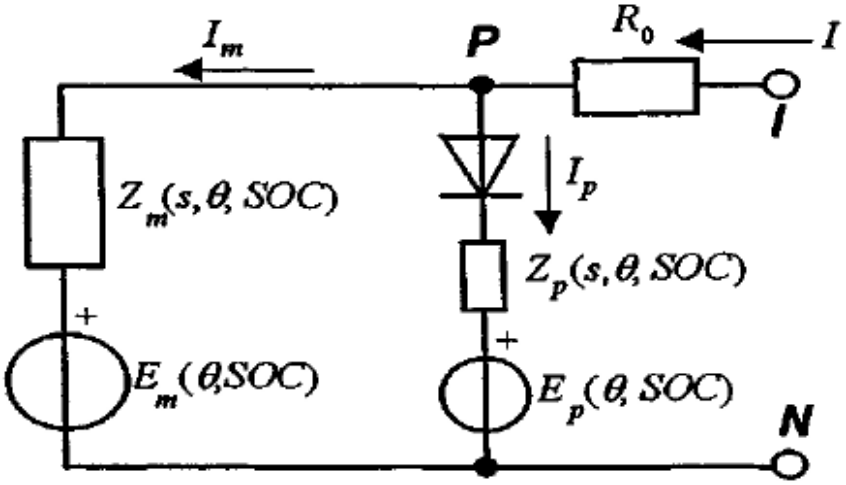
Por fim, o último modelo a ser apresentado é o mais usual. Criado com base nos modelos de ordem mais elevada, possui um comportamento mais dinâmico e não requere tanto tempo

computacional pois suas equações não são de uma ordem tão elevada. Este modelo envolve uma fonte de tensão variável, resistores variáveis representando a resistência interna e a resistência de sobretensão e um capacitor que representa a capacitância da bateria.

A simplificação que é feita do modelo original (modelo de ordem maior) é a desconsideração da parte do circuito de auto-descarga. Possibilitado pelo fato de que a auto-descarga normalmente é menor que 10% por mês nas tecnologias modernas, não havendo perdas na simulação ao ser ignorada.

Na literatura (CERAOLO, 2000), é apresentado um modelo matemático que pode ser tirado como base para qualquer outro modelo mais avançado que seja desejado. A Figura 2.18 apresenta esse modelo:

Figura 2.18 – Modelo Matemático



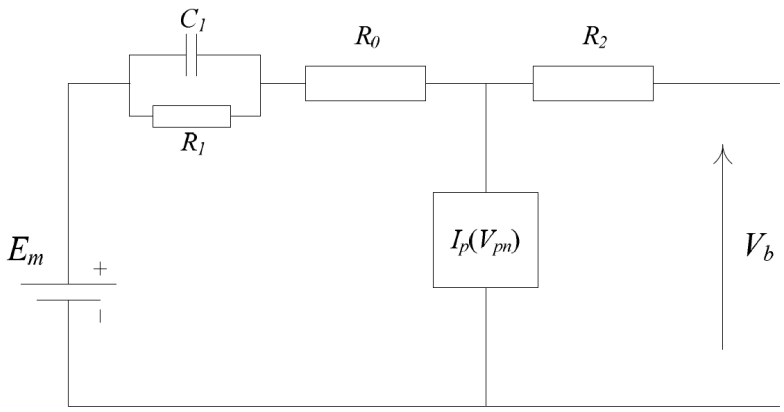
Fonte: Adaptado de (CERAOLO, 2000)

Este modelo está longe de ser linear, E_m e Z_m dependem do estado de carga da bateria e da temperatura do eletrólito (representado por θ). Neste circuito, a carga armazenada na bateria é apenas uma parte (I_m) da corrente total (I) entrando na bateria. Isso ocorre porque apesar da bateria ser um dispositivo armazenador de energia, esta não consegue armazenar toda a carga fornecida à mesma. Parte dessa energia é perdida por razões, como por exemplo a eletrólise da água (GUO, 2010). Sendo assim, neste modelo estas perdas são representadas por um ramo chamado parasita (elementos com subscrito p). A energia dissipada nas partes reais das impedâncias Z_m e Z_p é convertida em calor, contribuindo para aumentar a temperatura da bateria (CERAOLO, 2000).

Esse ramo parasita, entretanto, entra em funcionamento apenas quando a bateria está sendo carregada. Assim, a corrente parasita (I_p) pode ser descrita como uma função da tensão no ramo parasita (V_{PN}) para descrever a relação não linear neste ramo (o qual se representado apenas por um resistor (R_p) não ocorreria) (GUO, 2010). Dessa maneira, um circuito equiva-

lente ao anterior apresentado encontra-se na Figura 2.19. Onde C_1, R_1 e R_0 são os elementos da impedância interna Z_m.

Figura 2.19 – Modelo Matemático Equivalente



Fonte: Adaptado de (GUO, 2010)

Neste trabalho o estudo destes modelos serviu para o melhor entendimento dos parâmetros da bateria e para a determinação de problemas encontrados na parte de implementação. Visto que, posteriormente foi decidido utilizar um circuito integrado para fazer a estimação dos dados, pois a criação de um código implementando um modelo juntamente com o Filtro de Kalman mostrou-se muito complicado para o tempo disponível.

2.2.7 Estado de Carga (SOC) e Estado de Saúde (SOH)

2.2.7.1 História do Desenvolvimento da Indicação do SOC

Conforme a literatura (POP et al., 2008), sistemas para gerenciamento de baterias têm se tornado cada vez mais imprescindíveis desde o aparecimento de baterias recarregáveis. Para um adequado gerenciamento, é necessário uma boa medição do estado de carga das baterias (State of Charge - SOC). Uma precisa medição do SOC permite um aumento do desempenho e confiabilidade do sistema, evitando sobretensões e descargas excessivas, assim, prolongando a vida útil da bateria. O SOC é definido como a percentagem da capacidade total da bateria que ainda está disponível para ser utilizada.

O conhecimento do SOC possibilita um melhor planejamento por parte do usuário do sistema. O exemplo mais simples seria o celular. Saber o quanto de bateria ainda resta, permite que o usuário se prepare para carregar e dessa forma não fique sem carga em um momento importante. Outro aspecto importante é o estado de saúde da bateria (State of Health - SOH), este indica quanto tempo de vida útil a bateria ainda possui. Uma exemplificação neste caso seria,

se por exemplo a bateria do celular está com problema e requer manutenção. O conhecimento da vida útil restante pode ajudar a definir se vale mais a pena consertar o equipamento ou é mais vantajoso comprar uma bateria nova (POP et al., 2008).

Porém as vantagens da precisa determinação do SOC e do SOH se estendem também aos fabricantes de sistemas com baterias ou das próprias baterias (CHAGAS, 2007). Em função do desenvolvimento tecnológico, uma sofisticação do gerenciamento de baterias foi possível. Utilizando monitores de capacidade, controladores de carga, informações de tempo de descarga disponível, bem como contadores de ciclos, permitiu que as baterias fabricadas funcionem melhor dentro dos seus limites impostos no seu dimensionamento. Dessa maneira, reduz a necessidade de sobre-dimensionar os sistemas, o que além da vantagem financeira, pode resultar em uma bateria de menor tamanho e peso.

Uma estimação errônea do SOC ou a falta dela pode gerar danos graves às baterias. Induzindo à recarga excessiva, o que acaba por diminuir a vida útil da bateria. Por outro lado, pode deixar a bateria ser submetida a sobrecarga ou ficar em estado parcial de carga (falta de carga) o que é ainda mais prejudicial à vida útil da bateria (CHAGAS, 2007).

Existem inúmeras maneiras de determinar ou estimar o estado de carga de uma bateria. A precisão varia conforme a aplicação desejada. O primeiro medidor de SOC foi desenvolvido em 1963 pela empresa Curtis Instruments, e foi utilizado para medir o SOC de baterias de veículos de tração. A lógica era medir o tempo transcorrido até atingir uma determinada tensão, após a descarga com uma corrente predeterminada (POP et al., 2008).

Mais a frente, o método Current-Sharing foi desenvolvido. Este utiliza a comparação entre duas correntes, sendo a primeira fornecida pela bateria com um SOC conhecido e a segunda sendo a corrente fornecida pela bateria quando não é conhecido seu atual SOC. E assim, outros métodos tinham sua base na determinação do SOC por comparação entre correntes ou entre tensões (POP et al., 2008).

Em 1975 então, foi desenvolvido o método mais conhecido e pode-se dizer mais utilizado. A determinação do estado de carga baseado em uma tensão de circuito aberto (open-circuit voltage - OCV). Esta tensão é diretamente proporcional ao SOC da bateria e pode ser calculada segundo a equação (2.3):

$$OCV = V_{bat} + I\,R \tag{2.3}$$

onde V_{bat} é a tensão no terminal da bateria, I é a corrente atual (considerando valor positivo durante a descarga e negativo durante a carga) e R é a resistência interna. A desvantagem deste método está no fato de que após uma medição e interrupção da corrente é necessário um determinado tempo de relaxamento.

Então, em 1990 foi desenvolvido o método chamado "Coulomb Counting", o qual é baseado na medição e integração da corrente que é absorvida ou fornecida pela bateria. Porém este método só é válido para operações de curto espaço de tempo, onde a acumulação do erro não é extremamente prejudicial (POP et al., 2008). Sendo assim, uma junção dos dois últimos métodos citados foi proposta. A ideia é que o erro acumulado pelo Coulomb Counting pode ser corrigido medindo a tensão de circuito aberto do tempo restante de carga na bateria. Essa melhoria pode chegar a 99% de precisão, entretanto tem um custo alto (POP et al., 2008).

2.2.7.2 Métodos para Medição do SOC

Essencialmente os inúmeros métodos para medir ou estimar o estado de carga (SOC) podem ser separados em dois grupos: medição direta e medição indireta. Na medição direta, o sistema mede as variáveis da bateria como tensão, impedância e tempo de relaxação depois de aplicar um nível de corrente. Como a maioria das variáveis dependem da temperatura, esta também deve ser medida e controlada. Sua vantagem é que os métodos diretos não precisam estar conectados na bateria tempo integral (POP et al., 2008). Como este trabalho é direcionado para baterias de Chumbo-Ácido, os métodos apresentados serão explicitamente para este tipo de bateria.

Teste de Descarga (Discharge test) é o método mais confiável para estimação do SOC. Seu processo consiste em fazer um teste de descarga onde as condições são controladas. Porém sua desvantagem reside no tempo de descanso que é exigido para fazer uma nova descarga e novas medições (PILLER; PERRIN; JOSSEN, 2001).

Ampere Hour Counting é o método mais usual. Como a carga ou descarga está diretamente ligada a corrente fornecida ou drenada, a ideia é ver o comportamento da corrente. Dado um estado de carga inicial (SOC_o), o valor da integral (2.4) da corrente é um indicador do SOC (PILLER; PERRIN; JOSSEN, 2001).

$$SOC = SOC_o + \frac{1}{C_N} * \int_{t_0}^{t} (I_{bat} - I_{loss}) d\tau \tag{2.4}$$

Onde C_N é a capacidade nominal, I_{bat} é a corrente da bateria e I_{loss} é a corrente consumida nas perdas. Sua desvantagem é que uma medição errada da corrente implica em um erro grande na estimação do SOC e um aparelho de medição de corrente possui um custo muito elevado. Uma complicação adicional é a obrigação de levar em consideração a corrente de perdas. Todavia esses problemas podem ser significativamente reduzidos com uma re-calibração (PILLER; PERRIN; JOSSEN, 2001).

A *medição das propriedades físicas dos eletrólitos* é baseada na mudança da densidade do ácido presente na bateria. Este método aplica-se somente às baterias de chumbo-ácido venti-

ladas. A densidade é medida direta ou indiretamente pela concentração de íons, condutividade, viscosidade, entre outros aspectos. Porém, neste modelo a estratificação do ácido, bem como a perda de água são problemas que dificultam a utilização deste método. (PILLER; PERRIN; JOSSEN, 2001).

A *Tensão de Circuito Aberto* já foi citada acima e possui uma relação linear com o SOC. Normalmente esta técnica é utilizada em combinação com outras técnicas, com o objetivo de ajustá-las. Na maioria das vezes a diferença entre a tensão de circuito aberto para a bateria totalmente carregada e totalmente descarregada é cerca de 100mV.

Espectro de Impedância é um método para investigar o processo eletroquímico e é estudado para utilização não só na estimação do SOC, mas também do SOH (estado de saúde). Componentes de impedância de frequências diferentes são utilizados como entrada para cálculo do SOC e uma precisão de 95% foi encontrada. Fortemente influenciado pela temperatura, este método deve ser realizado em ambientes com temperatura controlada. Ainda com muitas controvérsias sobre este método, no que diz respeito ao funcionamento em certas frequências, estudos indicam que é aplicável somente em um determinado range de frequências (PILLER; PERRIN; JOSSEN, 2001).

Estimadores Recursivos utilizam técnicas recursivas para estimar os parâmetros de um determinado modelo à medida que os dados são disponibilizados (AGUIRRE, 2007). Esse modo é muito vantajoso para sistemas onde há não-linearidades, perdas, dentre outros influenciadores como é o caso da carga e descarga de baterias. Existem vários estimadores recursivos, pode-se citar como exemplo o Estimador Recursivo de Mínimos Quadrados (EMQ), MQ, KF (Filtro de Kalman), entre outros. O Filtro de Kalman é um estimador recursivo ótimo, estudado em detalhes na sessão a seguir.

2.2.8 Filtro de Kalman

O Filtro de Kalman nada mais é que um conjunto de equações recursivas, as quais são utilizadas para estimar estados de um sistema dinâmico. Assume-se que o sistema sob estudo pode ser descrito pelo modelo linear discreto apresentado na equação (2.5)

$$\begin{cases} x_{k+1} = \Phi_k \mathbf{x}_k + \Gamma_k \mathbf{u}_k + \Upsilon_k \mathbf{w}_k; \\ y_{k+1} = H_{k+1} \mathbf{x}_{k+1} + \mathbf{v}_{k+1}; \end{cases} \tag{2.5}$$

sendo que $\mathbf{w}$ e $\mathbf{v}$ são variáveis aleatórias independentes, de média nula, e que satisfazem $E[\mathbf{w}_k \mathbf{w}_k^T] = Q_k$, $E[\mathbf{v}_k \mathbf{v}_k^T] = R_k$ e $E[\mathbf{v}_i \mathbf{v}_j^T] = 0$, $\forall\ i,\ j$ (AGUIRRE, 2007). Usualmente $\mathbf{w}$ e $\mathbf{v}$ representam ruídos, sendo o primeiro oriundo do processo dinâmico ou um ruído multiplicativo e o segundo um ruído de medição. O algoritmo de Kalman pode ser utilizado para estimar os

estados de um sistema ou estimar estados e também parâmetros do modelo simultaneamente. Sendo esse o principal motivo de ser muito utilizado na prática.

Conforme a literatura (AGUIRRE, 2007), o algoritmo originalmente proposto por Kalman só se aplica a sistemas lineares. Para ser utilizado em sistemas não lineares é necessário linearizar analiticamente as equações do sistema. Dessa "técnica" veio o chamado Filtro de Kalman Estendido, o qual será descrito mais a frente.

2.2.8.1 Conceitos Básicos

Para explicar sucintamente o Filtro de Kalman, separa-se em dois casos: o caso estático e o caso dinâmico. Um exemplo simples do primeiro caso seria um objeto parado em determinada posição, a qual é descrita em função de duas variáveis, x e y. Deseja-se então conhecer a posição x, a partir de medições da posição y. Assim obtém-se uma $f(x \mid y)$, que quantifica a distribuição da densidade de probabilidade de x, baseada na medição de y. Supondo então que sejam feitas N medições para y, tem-se então uma função de densidade de probabilidade de condicional de x baseadas em todas estas medições, obtendo então a função $f(x \mid y_1,...,y_N)$. Essa função é chamada condicional porque depende do conjunto de medidas de $y_1, y_2, ..., y_N$. Ou seja, é um estimador da posição x do objeto. Neste caso, foi considerado que o objeto não tem sua posição alterada entre t_1 e t_2. Sendo assim, não havia nenhuma razão para mudar a estimativa de sua posição em t_2 antes de chegar a nova medição. Entretanto, se o objeto tivesse movimento de acordo com alguma lei como por exemplo $\dot{x} = v$, onde v é uma velocidade constante, então seria interessante utilizar essa lei para propagar a informação da posição de t_1 para t_2 antes de receber a nova medição $y(t_2)$. Portanto, utilizando a aproximação de Euler e incluindo uma variável aleatória que quantifica a incerteza na equação dinâmica que descreve o sistema, é possível estimar a nova posição do objeto no instante t_2 (AGUIRRE, 2007). Escrevendo como a equação (2.6).

$$\hat{x}(t_2|t_1) = \hat{x}(t_1|t_1) + (t_2 - t_1)v \tag{2.6}$$

Dessa forma, quando chega a nova medição, a equação (2.6) reduz a incerteza associada à nova estimativa. Isso mostra a estrutura *predição-correção* do algoritmo, visto que o conhecimento da lei de movimento torna possível fazer a *predição* e o conhecimento da nova medida possibilita a *correção*. As equações de correção são as mesmas para ambos os casos e estão descritas na equação (2.7):

$$\begin{aligned} \hat{x}(t_2|t_2) &= \hat{x}(t_2|t_1) + K(t_2)[y(t_2) - \hat{x}(t_2|t_1)] \\ K(t_2) &= \frac{\sigma^2(t_2|t_1)}{\sigma^2(t_2|t_1) + \sigma^2(t_2)} \end{aligned} \tag{2.7}$$

Onde K é um ganho que faz a correção da estimativa e σ é o desvio padrão da medida. Um fato muito importante é que se o modelo utilizado for do tipo contínuo, será necessário utilizar técnicas de integração numérica a fim de propagar os estados ou a saída do sistema (AGUIRRE, 2007). Por causa disso, neste trabalho apenas o caso em tempo discreto será considerado.

Portanto, o Filtro de Kalman estima valores reais de grandezas e valores associados predizendo um valor futuro, estimando a incerteza ou o erro do valor predito, e recalculando de forma a chegar em um valor preciso.

2.2.8.2 O Filtro de Kalman Discreto

Nesta caso então a nomenclatura do equacionamento muda, a fim de generalizar as leis. O instante t_1 será substituído pela iteração atual, indicada por k, e o instante t_2 pela próxima iteração, indicada por k+1. Além disso, a notação $(t_1|t_1)$ será substituída por um sinal '+', referindo-se ao instante t_i após a inclusão da informação chegada em t_i. Da mesma forma, será utilizado um sinal '-' para indicar que a grandeza refere-se ao instante t_i antes de ser inserida a informação daquele instante (AGUIRRE, 2007).

Depois de ter a nomenclatura adaptada, então apresenta-se a sequência de cálculo do filtro de Kalman discreto no sistema (2.8)

$$\begin{cases} \hat{\mathbf{x}}_{k+1}^- = \Phi_k \hat{\mathbf{x}}_k^+ + \Gamma_k \mathbf{u}_k; \\ P_{k+1}^- = \Phi_k P_k^+ \Phi_k^T + \Upsilon_k Q_k \Upsilon_k^T; \\ K_{k+1} = P_{k+1}^- H_{k+1}^T [H_{k+1} P_{k+1}^- H_{k+1}^T + R_{k+1}]^{-1}; \\ \hat{\mathbf{x}}_{k+1}^+ = \hat{\mathbf{x}}_{k+1}^- + K_{k+1}[\mathbf{y}_{k+1} - H_{k+1}\hat{\mathbf{x}}_{k+1}^-]; \\ P_{k+1}^+ = P_{k+1}^- - K_{k+1} H_{k+1} P_{k+1}^-; \end{cases} \tag{2.8}$$

A primeira equação do sistema é utilizada para estimar o valor antes de um determinado tempo k. A segunda equação é a matriz de covariância de $\hat{\mathbf{x}}_{k+1}^-$. A afirmação de que a matriz de covariância do vetor de estado estimado é igual à matriz de covariância do erro de estimação leva à obtenção da segunda equação, a qual é responsável por estimar o erro sobre o valor que foi estimado na primeira equação. A terceira equação faz a compensação entre o valor estimado e o erro. Onde H_{k+1} é o valor esperado para a saída do modelo nesse instante, utilizando a informação até o instante k. Então, a quarta equação faz uma nova estimação, agora após o tempo k, mais precisa que a anterior uma vez que o erro foi compensado. E por fim, um novo

erro é estimado e caso este ainda seja elevado, repetem-se as equações até os valores estimados obterem a precisão desejada (AGUIRRE, 2007).

2.2.8.3 O Filtro de Kalman Estendido

Segundo a literatura (AGUIRRE, 2007), o desenvolvimento do filtro de Kalman mostrado anteriormente é válido apenas para sistemas lineares. No caso não linear, a propagação não pode ser feita por um único modelo linear. A solução clássica então passou a ser conhecida como Filtro de Kalman Estendido, e consiste em linearizar analiticamente o sistema em torno do atual estado e posteriormente aplicar as equações do filtro de Kalman.

Um novo sistema de equações então é considerado, em (2.9):

$$\begin{cases} \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_k) \\ \mathbf{y}_{k+1} = h(\mathbf{x}_{k+1} + \mathbf{v}_{k+1}) \end{cases} \tag{2.9}$$

onde f e h são as funções não-lineares do processo e de observação, respectivamente. O filtro de Kalman estendido, é o próprio filtro de Kalman implementado, porém utilizando matrizes jacobianas de f e h, ou seja, linearizando (a partir dos primeiros termos da expansão da série de Taylor) dessas funções, como pode ser visto em (2.10).

$$Df(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \vdots & \ldots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \tag{2.10}$$

A matriz (2.10) representa a matriz jacobiana da função f, e de maneira análoga são compostas a matriz jacobiana de f com respeito ao vetor $\mathbf{w}$ e a matriz jacobiana da função h. Ao contrário do que acontece nos sistemas lineares, neste caso a matriz jacobiana não é constante. Porém, se D$f(\mathbf{x})$ e D$h(\mathbf{x})$ forem analisadas em torno de valores específicos do vetor de estado, como por exemplo $x = x_0$, então estas matrizes se tornam constantes. Dessa forma, $Df(\mathbf{x}_0)$ e $Dh(\mathbf{x}_0)$ são as linearizações de f e h em torno de $x = x_0$. Procedendo assim, as equações do filtro de Kalman estendido, no caso discreto, podem ser reescritas como:

$$\begin{cases} \hat{\mathbf{x}}_{k+1}^- = f(\hat{\mathbf{x}}_k^+, \mathbf{u}_k); \\ P_{k+1}^- = Df(\mathbf{x}_k) P_k^+ Df(\hat{\mathbf{x}}_k^+)^T + Df(\mathbf{w}_k) Q_k Df(\mathbf{w}_k)^T; \\ K_{k+1} = P_{k+1}^- Dh(\mathbf{x}_{k+1})^T [Dh(\mathbf{x}_{k+1}) Dh(\mathbf{x}_{k+1})^T + Dh(\mathbf{x}_{k+1}) R_{k+1} Dh(\mathbf{x}_{k+1})^T]^{-1}; \\ \hat{\mathbf{x}}_{k+1}^+ = \hat{\mathbf{x}}_{k+1}^- + K_{k+1}[\mathbf{y}_{k+1} - h(\hat{\mathbf{x}}_{k+1}^-)]; \\ P_{k+1}^+ = P_{k+1}^- - K_{k+1} Dh(\mathbf{x}_{k+1}) P_{k+1}^-; \end{cases} \tag{2.11}$$

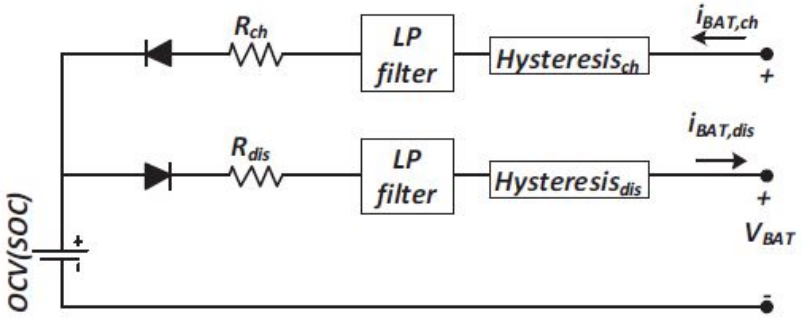
É importante notar que a primeira equação do sistema (2.11) representa a propagação do vetor de estado, utilizando a função não-linear *f*. Simultaneamente, a quarta equação do sistema (2.11) utiliza a função *h* para determinar a saída desejada, dado o vetor de estado propagado. Nas demais equações, as matrizes jacobianas foram utilizadas para determinar as matrizes de covariância e o ganho de Kalman (AGUIRRE, 2007).

Conforme (DRAGICEVIC; SUCIC; GUERRERO, 2013), para obter uma estimação correta do estado de carga, é preciso uma estimação precisa dos parâmetros do modelo de bateria utilizado. Por este motivo, foi criado o Filtro de Kalman Dual, o qual estima simultaneamente o vetor de estado e os parâmetros. O algoritmo é separado em duas sessões a fim de manter uma limitada dimensão das matrizes de transição de estados. O princípio é o cálculo do novo SOC em cada instante, e depois adaptar de acordo com a estimação dos parâmetros. Por fim, os parâmetros que variam mais devagar são atualizados com base no primeiro SOC estimado, como se estes permanecessem constantes caso um novo valor estimado de SOC for utilizado.

2.2.8.4 Aplicação do Filtro de Kalman Dual no Modelo de Ordem Maior

Uma imagem mais simplificada do modelo de quarta ordem, já incluindo um filtro passa-baixas e a histerese é apresentada na Figura 2.20.

Figura 2.20 – Modelo da bateria para aplicar o Filtro de Kalman



Fonte: Adaptado de (DRAGICEVIC; SUCIC; GUERRERO, 2013)

Baseado no método *Ampére-Counting*, o estado de carga pode ser calculado conforme a equação (2.12)

$$SOC_{k+1} = SOC_k - \eta \frac{i_{BAT,k}}{C_{BAT}} \Delta t \tag{2.12}$$

Onde, η é a eficiência de carga e descarga, $i_{BAT,k}$ é a corrente de bateria, C_{BAT} é a capacidade nominal e Δt é a variação discreta do tempo. (DRAGICEVIC; SUCIC; GUERRERO, 2013)

A queda de tensão instantânea leva em consideração as resistências internas de carga (R_{ch}) e descarga (R_{dis}). E a queda de tensão transiente é modelada como uma combinação de um filtro passa-baixas (considerado dinâmica rápida) com o efeito de histerese (considerado dinâmica lenta). A equação do filtro discreto é representada em (2.13)

$$\begin{cases} filt(i_{BAT,k}) = C_f f_k \\ f_{k+1} = A_f f_k + B_f\, i_{BAT,k} \end{cases} \tag{2.13}$$

Onde A_f é a matriz de transição de estados do filtro, B_f é a matriz de estados de entrada e C_f é a matriz de estados de saída. Já a dinâmica lenta pode ser representada pela equação (2.14), na qual γ é a velocidade de resposta e A_{hist} é a amplitude de histerese (DRAGICEVIC; SUCIC; GUERRERO, 2013).

$$g_{k+1} = exp\left(-\frac{|\eta \cdot i_{BAT,k} \cdot \gamma \cdot \Delta t|}{|C_{BAT}|}\right) + \left(1 - exp(-\frac{|\eta \cdot i_{BAT,k} \cdot \gamma \cdot \Delta t|}{|C_{BAT}|}\right) A_{hist} \tag{2.14}$$

A soma das equações apresentadas anteriormente permitem a composição da equação (2.15), que fornece a tensão no terminal da bateria a qual é comparada com a tensão medida na parte de correção do algoritmo (DRAGICEVIC; SUCIC; GUERRERO, 2013).

$$y_k = OCV(SOC_k) + g_k + filt(i_{BAT,k}) - R\, i_{BAT,k} \tag{2.15}$$

Como o processo de estimação é dual, ou seja estima os parâmetros e o estado simultaneamente, é necessário definir o vetor de parâmetros θ, o qual contém as resistências internas, a amplitude de histerese e a constante de tempo (γ), e está apresentado na equação (2.16)

$$\theta = \begin{bmatrix} R_{ch} & R_{dis} & A_{hist} & \gamma \end{bmatrix} \tag{2.16}$$

A função que incorpora o método *coulomb counting* é $f(x_k, u_k, \theta_k)$, enquanto a função $h(x_k, u_k, \theta_k)$ calcula a tensão de terminal da bateria de acordo com (2.15). A variável u_k compreende a corrente da bateria $i_{BAT,k}$.

Considerando o sistema apresentado na equação (2.9) é agora apresentado um sistema análogo, apenas adicionando a parte dual, resultando então no sistema (2.17)

$$\begin{cases} x_{k+1} = f(x_k, u_k, \theta_k) + w_k \\ \theta_{k+1} = \theta_k + r_k \end{cases} \tag{2.17}$$

Como o sistema não é linear, deve-se linearizá-lo por meio da matriz jacobiana apresentada previamente na equação (2.10). Portanto o sistema é linearizado em torno do ponto $x = x_0$ e analogamente neste caso $\theta = \theta_0$.

Chegando então, conforme apresentado em (DRAGICEVIC; SUCIC; GUERRERO, 2013), às equações (2.18).

$$\begin{cases} \theta_0^+ = E(\theta_0),\ P_{\theta_0^+} = E[(\theta_0 - \theta_0^+)(\theta_0 - \theta_0^+)^T] \\ x_0^+ = E(x_0),\ P_{x_0^+} = E[(x_0 - x_0^+)(x_0 - x_0^+)^T] \end{cases} \tag{2.18}$$

Após obter a linearização analítica, é possível aplicar então as equações do sistema (2.11), adaptando apenas a equação $\hat{x}_{k+1}^- = f(\hat{x}_k^-, u_k)$ para $\hat{x}_{k+1}^- = f(\hat{x}_k^-, u_k, \theta_k)$.

Os dois últimos passos são as atualizações das medidas dos estados e parâmetros. Primeiramente o ganho de Kalman é computado e depois usado para adaptação do SOC e dos outros parâmetros, com o propósito de reduzir o erro entre a medição da tensão nos terminais da bateria e a saída do modelo de bateria. A fim de possibilitar, caso desejado, o desacoplamento dessa sincronização dual (atualização e correção das médias do SOC e dos parâmetros), a adaptação dos parâmetros é configurada para usar em seus cálculos o valor prévio estimado para o SOC (DRAGICEVIC; SUCIC; GUERRERO, 2013).

2.3 CONCLUSÃO DO CAPÍTULO

Para garantir a confiabilidade de sistemas de armazenamento, é necessário monitorar seus processos de carga e descarga, evitando sobretensões, temperatura elevada e picos de corrente. Para haver um bom planejamento é importante uma precisa estimação do estado de carga das baterias, além do seu estado de saúde. O primeiro passo seria fazer uma correta modelagem da bateria, a fim de conhecer seus parâmetros para controlá-los de forma satisfatória. Os modelos vão dos mais simples porém menos precisos até os mais complexos. É necessário analisar qual a precisão desejada. a estimação dos estado de carga e de saúde pode ser feita por diversos métodos, sendo o mais usual os estimadores recursivos, como o Filtro de Kalman. Com os parâmetros das células bem definidos através da correta modelagem da bateria, este estimador recursivo ótimo retorna a quantidade da carga remanescente. O filtro de Kalman é um algoritmo de predição e erro, pois ele estima um valor, depois calcula seu erro e posteriormente aplica a correção. Este processo se repete até que o valor estimado esteja o mais perto possível do real. Este controle implica não somente em vantagens para o usuário do produto, bem como para o fabricante. Os métodos de modelagem apresentados neste capítulo tem a função de determinar os parâmetros da bateria que influenciam no seu estado de carga e estado de saúde. Ou seja, quais parâmetros devem ser monitorados para realizar o controle da bateria, afim de obter a melhor vida útil possível. No caso deste trabalho, estes parâmetros serviram de auxílio para entender melhor o funcionamento da bateria e o que poderia estar influenciando nos resultados que não bateram com o desejado. Pois estes parâmetros são medidos internamente pelo circuito

integrado utilizado. Os métodos de estimação tornam possível a construção de um algoritmo para a estimar o estado de carga e saúde das baterias. A intenção inicial deste trabalho seria a obtenção deste código através do estimador recursivo Filtro de Kalman, porém seria muito complicado criar este código no tempo disponível e por isso optou-se por utilizar um circuito integrado que faz essa estimação e validar seu funcionamento. O CI é apresentado no capítulo a seguir e ele utiliza o método *Coulomb Counter* para fazer as medições.

3 IMPLEMENTAÇÃO

3.1 CIRCUITOS INTEGRADOS COMERCIAIS

Para a implementação de um BMS para baterias de chumbo-ácido foram pesquisados circuitos integrados comerciais que façam uma estimação do estado de carga (SOC) e estado de saúde (SOH) de baterias. Dois CIs fabricados pela Texas Instruments foram selecionados como mais interessantes para este trabalho e nesta secção será apresentada uma pequena comparação entre suas funcionalidades e por fim, o CI escolhido mais adequado à esta aplicação.

3.1.1 BQ34110

O primeiro CI a ser apresentado é o BQ34110. (Texas Instruments, 2016) Este componente pode ter suas funções para um célula única ou para uma associação de células que resultem em até 65V, através de um associação interna ou externa de divisores resistivos para reduzir a tensão até um range aceitável pelo ADC do CI. Este divisor resistivo porém deve ser controlado para evitar perdas de dissipação desnecessárias. A partir de medições de tensão, corrente e temperatura, juntamente com a tecnologia CEDV (tensão de fim de descarga), determina informações da célula como capacidade remanescente, capacidade total e corrente média. Possui também a função EOS (término de serviço), a qual monitora a "saúde" da bateria e emite um alerta quando a mesma está chegando ao fim de sua vida útil.

Para a medição de temperatura, o componente possui um sensor integrado além de suportar um termistor NTC externo. A corrente da bateria é medida monitorando a corrente através de um resistor (R_{SENSE}) posicionado em série com a bateria, o qual tem seu valor típico entre $5m\Omega$ e $20m\Omega$. Este CI possui dois ADCs: o primeiro é dedicado à medição de corrente e o segundo para medir diversos outros parâmetros como temperatura e tensão.

A comunicação é feita através da interface I^2C, suportanto um barramento de até 400kHz. A fim de minimizar o consumo de potência, o componente possui alguns modos de operação como normal, *snooze* e *sleep*.

3.1.2 BQ34Z100-g1

O segundo CI selecionado é o BQ34Z100-g1, também da Texas Instruments. Como o anterior, pode ser utilizado para um única célula ou para associações destas, atingindo uma

tensão de até 65V, fazendo as configurações externas necessárias. (Texas Instruments, 2015) Pode fornecer informações como capacidade remanescente, capacidade total de carga e corrente média. Os dados são armazenados em uma memória *flash* não volátil.

O diferencial deste circuito integrado é sua alta precisão da estimativa do SOC. A qual é feita por um algoritmo através do modelo de espectro de impedância. Este algoritmo utiliza medidas de tensão, características e propriedades para criar predições do estado de carga que podem chegar à uma precisão de 1% de erro em diversas condições de operação. Este dispositivo mede a atividade de carga e descarga através do monitoramento da tensão através de um resistor (R_{SENSE}) conectado em série com o lado de baixa tensão da célula. Quando uma carga é conectada, a impedância da célula é medida comparando sua OCV com a tensão medida sobre condições de carga. A temperatura pode ser medida da mesma maneira que o CI BQ34100.

Sua comunicação pode ser feita não só pela interface I^2C, como também pela HDQ (High-Speed Data Queue - transferência de dados de alta qualidade) . Com o objetivo de minimizar o consumo de potência, possui três modos de operação como normal, *sleep* e *full sleep*.

A Tabela 3.1 mostra uma comparação resumida entre os dois CIs apresentados acima.

Tabela 3.1 – Comparação CIs

Características	BQ34110	BQ34Z100-g1
Número de Células em Série	> 4	> 4
Capacidade da Bateria (Ah)	< 29	< 29
Interface de Comunicação	I^2C	I^2C e HDQ
Precisão	Normal	Alta
Range de Temperatura de Operação (°C)	-40 a 85	-40 a 85
Quantidade de Pinos	14	14
Indicação Externa de Capacidade	nenhuma	LED
Corrente máxima (A)	32	32
Funções Especiais	Fim de serviço (EOS) Autenticação SHA-1	Autenticação SHA-1

Fonte: Adaptado do site da Texas Instruments

O dispositivo escolhido para este projeto foi o BQ34110, por ser mais simples e mesmo assim satisfazer todas as necessidades deste projeto. Enquanto o BQ34100-g1 utiliza o método espectro de impedância para estimar o estado de carga e de saúde, no CI escolhido é utilizado o método *Coulomb Counter*. Este segundo método apesar de menos preciso é muito mais simples e por isso a justificativa da escolha pelo BQ34110. Neste trabalho então a intenção é analisar a precisão do método implementado.

3.2 O CIRCUITO INTEGRADO ESCOLHIDO

A seguir será detalhado o funcionando e utilização do circuito integrado BQ34110 que foi o escolhido para implementação deste trabalho. É importante ressaltar que este circuito integrado não faz medição de célula por célula quando várias associadas em série, porém apenas a estimação do *pack* (conjunto das células) todo. Desta forma, não é possível saber em qual célula encontra-se uma sobretensão ou sobretemperatura, apenas é possível reconhecer que há essa anormalidade no conjunto.

3.2.1 Layout da Placa

O *layout* da placa a ser implementada foi baseado em (Texas Instruments, 2016) com adaptações para os componentes encontrados no Brasil. Antes de detalhar os circuitos existentes na placa é apresentada na Tabela 3.2 a pinagem do BQ34110.

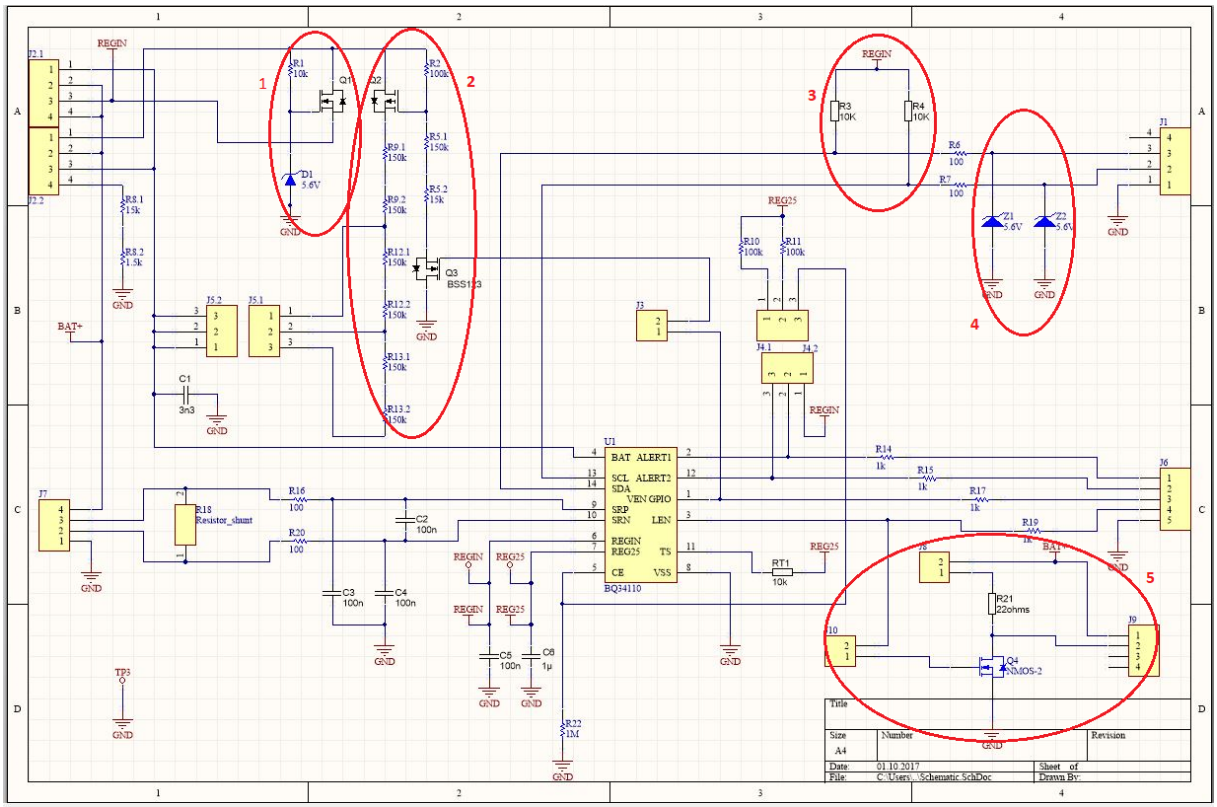
Tabela 3.2 – Pinagem BQ34110

Nome	**Número**	**Tipo**	**Descrição**
VEN/GPIO	1	O	Este pino é utilizado para selecionar ou não o divisor de tensão contido na placa afim de reduzir o consumo de potência.
ALERT1	2	O	Pino aberto para utilizar como alerta do sistema. Com a tensão limitada por resistor de *pull-up*.
LEN	3	O	Saída para divisor de tensão externo e controle da saída para carga.
BAT	4	P	Entrada para medida de tensão.
CE	5	I	Habilita o sistema. O regulador de tensão interno é desligado quando este pino está no estado *low*.
REGIN	6	P	Entrada do regulador interno de tensão.
REG25	7	P	Saída do regulador interno de tensão.
VSS	8	P	Aterramento do CI.
SRP	9	I	Entrada analógica conectada ao periférico *coulomb-counter* interno, para integrar a tensão entre SRN e SRP, quando SRN é conectado ao *BAT-*.
SRN	10	I	Entrada analógica conectada ao periférico *coulomb-counter* interno, para integrar a tensão entre SRN e SRP, quando SRN é conectado ao *PACK-*.
TS	11	I	Sensor do thermistor interno.
ALERT2	12	O	Pino aberto para utilizar como alerta do sistema.
SCL	13	I	Pino para o *clock* da comunicação serial I^2C.
SDA	14	I/O	Pino para transferência de dados na comunicação I^2C.

Fonte: Adaptado do site da Texas Instruments

O esquemático foi desenvolvido no programa Altium Designs e pode ser visto na Figura 3.1.

Figura 3.1 – Esquemático para desenvolvimento da placa

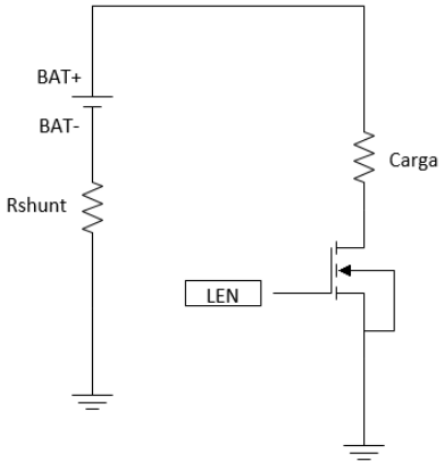


Fonte: Próprio Autor.

O banco de baterias é ligado ao conector J7, a partir dessa entrada é transferido para os conectores J2.1 e J2.2. Nestes, duas configurações podem ser escolhidas para o circuito: quando se utiliza apenas uma bateria (tendo então tensão abaixo de 5V) ou quando se utiliza várias células em série (podendo chegar até 64V). Quando escolhida a opção de célula única, *jumpers* são colocados em J2.1 e assim a tensão que chega do terminal positivo da bateria (BAT+) é transferia diretamente aos pinos BAT (através da conexão dos pinos 1 e 2) e REGIN (através da conexão dos pinos 3 e 4). O pino BAT é a entrada para a medição de tensão do circuito integrado, enquanto que o pino REGIN é a entrada do regulador de tensão interno ao CI. Quando a configuração de várias células é escolhida, então são posicionados *jumpers* em J2.2. Neste modelo de configuração, a tensão oriunda do terminal positivo da bateria (BAT+) é transferida para os drenos dos mosfets Q1 e Q2 e então possui sua tensão reduzida por divisores resistivos até atingir um range suportável pelo BQ34110. O circuito composto por Q1 (utilizado apenas na configuração para mais de uma célula), um zener e um resistor tem como função estender o range de tensão a ser aplicado ao regulador interno. Esse circuito cria um conversor dc/dc e a tensão em excesso, a qual não pode ser passada para a entrada do regulador, fica sobre

o transistor Q1. O diodo zener e o resistor R1 providenciam a polarização do *gate* do mosfet e a corrente "puxada" pelo regulador determina a tensão no *source* do mosfet. A máxima tensão presente no *gate* deve ser simplesmente a máxima tensão recomendada para a entrada do regulador de tensão. Dessa forma Q1 trabalha na região linear e não deixa a tensão em REGIN ultrapassar 5.5 que é o limite físico suportado por este pino. Essa tensão REGIN também aparece alimentando o barramento de comunicação, com resistores de pull-up entre esta tensão e o barramento, os quais seriam os resistores R3 e R4. Conectado a este barramento também se encontram dois zeners com a função de proteção. O circuito composto por Q2 e uma série de resistores é ativado só na configuração de mais de uma célula. Assim, a tensão proveniente da bateria passa por vários divisores resistivos e através dos conectores J5.1 e J5.2 é possível selecionar qual o nível desejado de tensão que será colocada no pino BAT do CI. Através da conexão dos terceiro e quarto pinos do conector J2.2 é adicionado uma resistência resultante de 16.5k (somando R8.1 e R8.2) o qual quando selecionada irá fornecer uma tensão muito baixa para BAT, menor que 1V. Pois apesar de seu limite físico ser até 5.5V, quando selecionada a utilização de divisores resistivos externos, é instruído pela Texas Instruments que esta tensão não ultrapasse 0.9V. O circuito composto por Q2, R21 e os conectores J8, J9 e J10 é preparado para a associação de uma carga ao circuito. O circuito pode ser configurado para ativar ou desativar a alimentação desta carga conforme o estado de carga da bateria permitir ou não. Esta função é realizada pelo pino LEN e o *jumper* J10. Com o objetivo de proteger a carga, o pino LEN pode jogar sua saída para zero, o que faz com que o mosfet Q4 não conduza e então sobre a carga não terá tensão, pois ficará em aberto. Através de J8 é possível colocar uma carga externa em série com a carga já existente a fim de diminuir a corrente passando pelo circuito. J9 permite medir a tensão em cima da carga, bem como adicionar uma carga em paralelo caso seja desejado. Esse circuito é associado com R18 que é o resistor *shunt* para medições de corrente do circuito. O circuito resultante pode ser visto na figura 3.2. A corrente que passa pela carga será a mesma que passa pelo R18, o qual tem suas extremidades conectadas aos pinos SRN e SRP do CI. Estes pinos são entradas analógicas conectadas ao conversor ADC exclusivo do periférico *Coulomb Counting*. Este periférico integra a tensão entre os dois pinos. Os resistores R14, R15, R17 e R19 têm a função de limitar a corrente dos pinos de saída. Os quais estão ligados ao conector J6 para serem medidos ou, quando conectados a outro circuito, acenderem um LED conforme a atividade programada ou alarme acionado.

Figura 3.2 – Circuito formado pelas células, a carga e o resistor shunt



Fonte: Próprio autor.

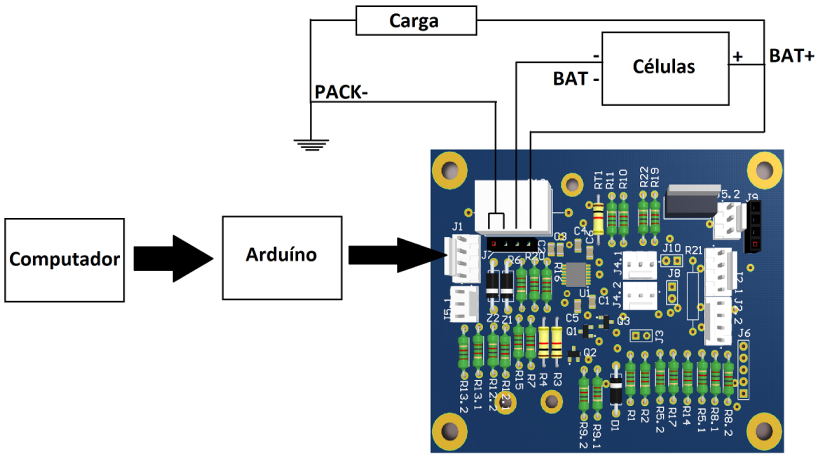
Os resistores R3 e R4 (resistores de *pull-up*) são utilizados para garantir que as entradas para sistemas lógicos se ajustem em níveis lógicos esperados se os dispositivos externos estão desconectados. Ou seja, para não deixar a porta flutuando. Na comunicação I^2C é necessário tanto em seu pino de clock (SCL) como em seu pino de dados (SDA), visto que são pinos do modelo coletor aberto. Isso significa que o chip pode apenas puxar as linhas para o estado *LOW*. Como a comunicação I^2C pode possuir vários escravos no mesmo barramento, esta configuração permite que um nó determine se outro está transmitindo. Por exemplo, se é medido o estado *HIGH* na linha, e então é variado um nó e a linha permanece no estado lógico *HIGH* (significa que nenhum outro nó está puxando o barramento para *LOW*), sendo assim nenhum outro nó está transmitindo simultaneamente e o barramento está "livre". Caso outro nó "puxe"o barramento para *LOW*, então o primeiro nó perde a arbitragem e cessa a transmissão. O capacitor C1 tem a função de jogar o *ripple* AC para o terra. Com o objetivo de reduzir sua influência nas medições de tensão da bateria. Diminuindo assim, o ruído na medição. O capacitor C5 serve para aumentar a PSR (Power Supply Rejection) e melhorar a regulação efetiva. Também servindo de capacitor de desacoplamento. O capacitor C6 é utilizado para garantir um reservatório de corrente para os picos de carga quando houver muitos periféricos. Além de agir para estabilizar a saída do regulador de tensão e reduzir a tensão de *ripple* no núcleo. Os conectores J4.1 e J4.2 são utilizados para ativar os resistores de *pull-up* das saídas de alertas do CI. É aplicada uma tensão proveniente da saída do regulador de tensão interno ao chip. A tensão REG25 que fica em torno de 2.5V a 2.8V passa pelos resistores R10 e R11 e é aplicada às saídas. É através destes conectores também que o chip é alimentado e o seu pino ENABLE ativado. Aplicando a tensão de entrada do regulador de tensão, fazendo a conexão do pino 1 de J4.2 ao pino 3 de J4.1.

Por fim, o conector J3 é utilizado para ativar a redução do consumo de potência polarizando o mosfet Q3.

3.2.2 Conectando bq34110 ao banco de baterias

A Figura 3.3 mostra como o banco de baterias deve ser conectado à placa. O lado positivo da célula deve ser conectado ao pino 4 do conector J7, o lado positivo ao pino 3 e os pinos 1 e 2 devem ser conectados entre si (para que o PACK- seja aterrado). A carga conectada entre o pino positivo da célula e o terra entrará em paralelo com a carga já presente na placa (R21). A segunda opção de conexão de uma carga em série com a carga já implementada é conectando através do conector J8.

Figura 3.3 – Conexão do Banco de células ao Circuito



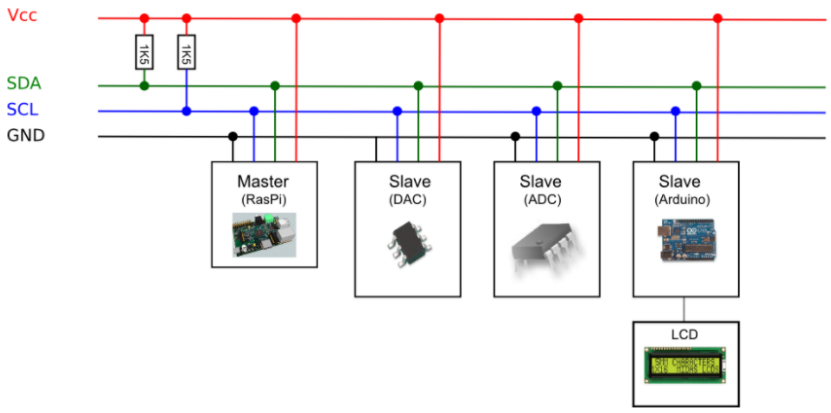
Fonte: Próprio autor.

3.3 A COMUNICAÇÃO I2C

I^2C é um protocolo de comunicação que trabalha no modo *master-slave* (mestre-escravo). É um protocolo de barramento, ou seja, com os mesmos fios conectamos todos os dispositivos. Neste barramento há pelo menos um mestre e pode ter até 127 escravos conectados. Pode haver mais de um mestre no barramento, entretanto somente um pode estar ativo, ou ocorrerá uma colisão de dados. A função do mestre é coordenar a comunicação, sendo o mesmo que envia ou solicita informações de determinado escravo. Essa característica de comunicação em barramento traz a vantagem que o mestre (um microcontrolador por exemplo), precisa de apenas 2 pinos dedicados à este protocolo, pois usará sempre os mesmos fios seja para um ou 127

escravos. A ordem em que são conectados os dispositivos não importa. A Figura 3.4 exemplifica este protocolo.

Figura 3.4 – Barramento para comunicação I^2C



Fonte: https://goo.gl/eu27c9.

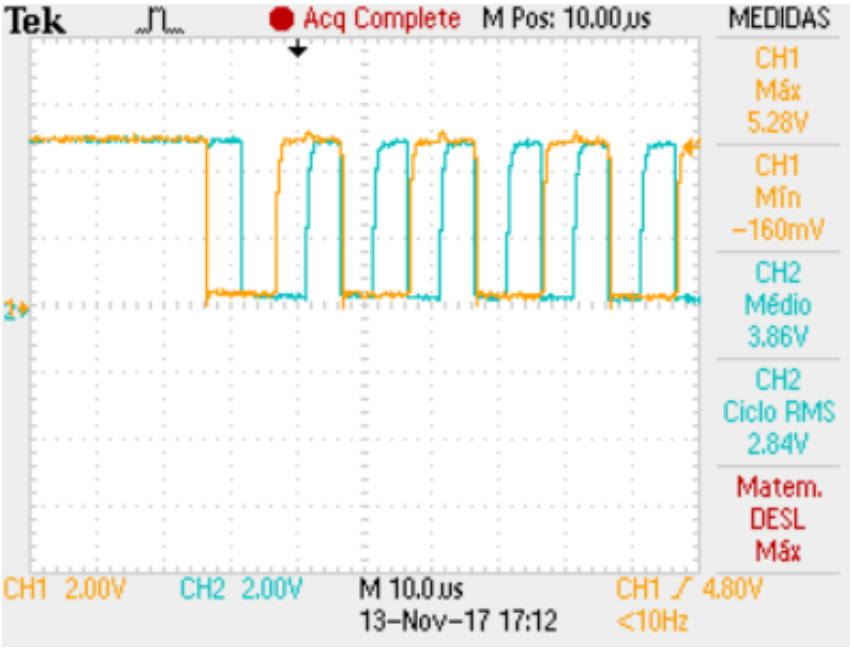
Para a comunicação é necessário um pino de *clock* e um pino de dados, além de ser necessário também interligar o terra dos dispositivos. Pela Figura 3.4 nota-se que é preciso alimentar este barramento através de resistores de *pull-up* como já foi explicado na secção anterior. O pino de dados é responsável pelo envio e recebimento dos dados, sendo assim uma linha bi-direcional. Ou seja, ora é enviado dados por este pino, ora é recebido dados do mesmo. A distância normal de trabalho no barramento é de aproximadamente 1 metro. Passando disso, é possível ter problemas de impedância.

Existem uma variedade enorme de dispositivos que utilizam o protocolo I^2C, como Raspberry (um microcomputador), memórias externas (EEPROM), *I/O expanders*, RTC, visores (LED, TFT), inúmeros sensores (temperatura, acelerômetro, etc) e o microcontrolador arduíno (o qual foi escolhido para este projeto).

Como a linha de dados é bi-direcional, a comunicação I^2C possui duas condições para indicar aos dispositivos quando começa e quando termina o envio de informações. Essas condições são chamadas de início e término. A condição de início consiste na ocorrência de uma transição do nível alto para o nível baixo na linha de dados, enquanto a linha de *clock* permanece em nível alto. Ou seja, quando isso ocorre os dispositivos conectados ao barramento ficam "preparados" para receber o comando. A Figura 3.5 é a imagem dos pulsos aproximados para a condição de início no osciloscópio[1] obtida em laboratório quando feita a transmissão dos dados do arduíno para o bq34110.

[1] A curva azul é a curva de clock e a curva amarela a de dados.

Figura 3.5 – Condição de Início obtida em Laboratório

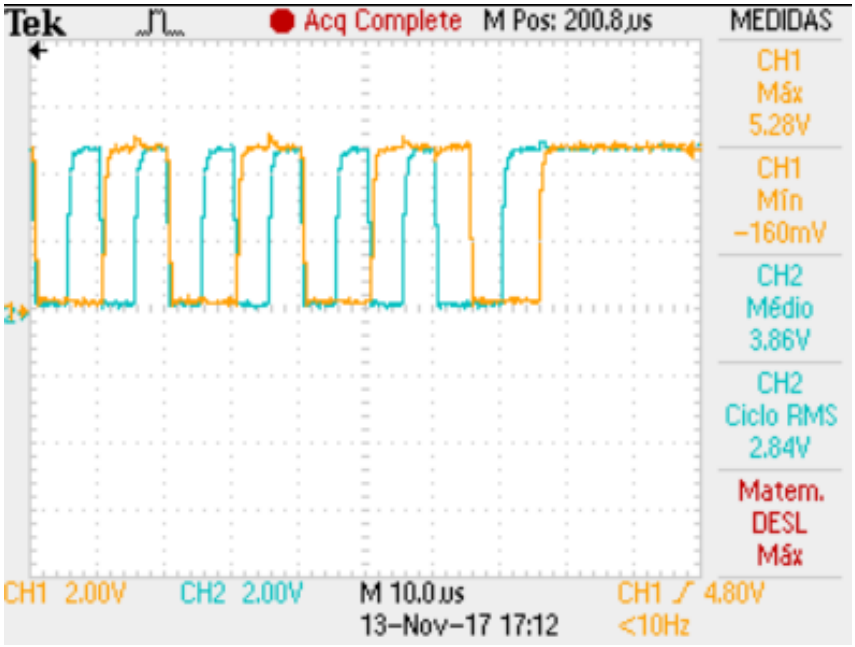


Fonte: Próprio autor.

A condição de término ocorre quando há uma transição de nível baixo para nível alto na linha de dados enquanto a linha de *clock* permanece em nível alto. A Figura[2] 3.6 é a imagem dos pulsos aproximados para a condição de término no osciloscópio obtida em laboratório quando feita a transmissão dos dados do arduíno para o bq34110.

[2] A curva azul é a curva de clock e a curva amarela a de dados.

Figura 3.6 – Condição de Término obtida em Laboratório

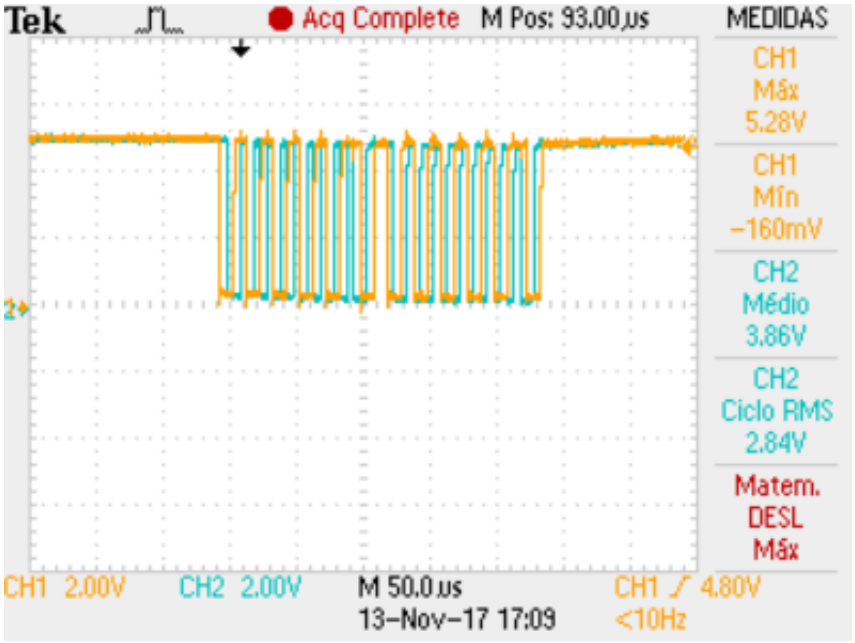


Fonte: Próprio autor.

É sempre o mestre o responsável pela geração dessas condições. Após uma condição de início o barramento é considerado ocupado e apenas volta a ficar livre algum tempo depois da condição térmico ocorrer. O número de *bytes* que pode ser transferido é limitado. Cada informação pode ter no máximo o comprimento de um byte. Caso o dispositivo que está recebendo o sinal não possa trabalhar os dados, este pode alterar a linha de *clock*, colocando-a em nível baixo e assim forçar o mestre a entrar em um estado de espera. Na Figura[3] 3.7 é possível ver a imagem do osciloscópio nesta transferência de dados. (REIS, 2017).

[3] A curva azul é a curva de clock e a curva amarela a de dados.

Figura 3.7 – Pulsos Originados pelo Arduíno na Comunicação



Fonte: Próprio autor.

3.3.0.1 Endereçamento dos Dispositivos

Cada dispositivo conectado ao barramento possui um endereço que os identifica. É através deste endereço que o mestre irá "escolher" para qual escravo irá enviar ou requerer informações. Neste protocolo, este endereço é composto por 7 bits (desta forma podendo ter 127 endereços diferentes, justificando o fato de o barramento suportar teoricamente 127 dispositivos conectados ao mesmo).

Como a linha é bi-direcional é necessário o mestre indicar se quer escrever no dispositivo ou ler alguma informação. Isso é feito através do oitavo bit, chamado de bit *read/write*. Este bit será zero quando o mestre quer escrever algo no dispositivo, e será um quando quiser ler algo do mesmo. Assim, cada dispositivo deve indicar em seu *datasheet* um endereço composto por 7 bits.

No caso deste trabalho, o endereço indicado pelo BQ34110 é 85. Estes sete bits devem ser deslocados para a esquerda, ou seja, estes são os bits mais significativos do endereço. Então, é adicionado o bit *read/write* no final, como o bit menos significativo. O arduíno faz essa adição internamente, sendo necessário apenas informar o endereço de 7 bits. Caso o endereço fornecido pelo *datasheet* do dispositivo seja de oito bits, é preciso fazer o cálculo do endereço correto, sem o bit que define a escrita ou leitura. O range válido de endereços I^2C vai de 8 a 119 (em decimal), os demais endereços são reservados e não podem ser utilizados por

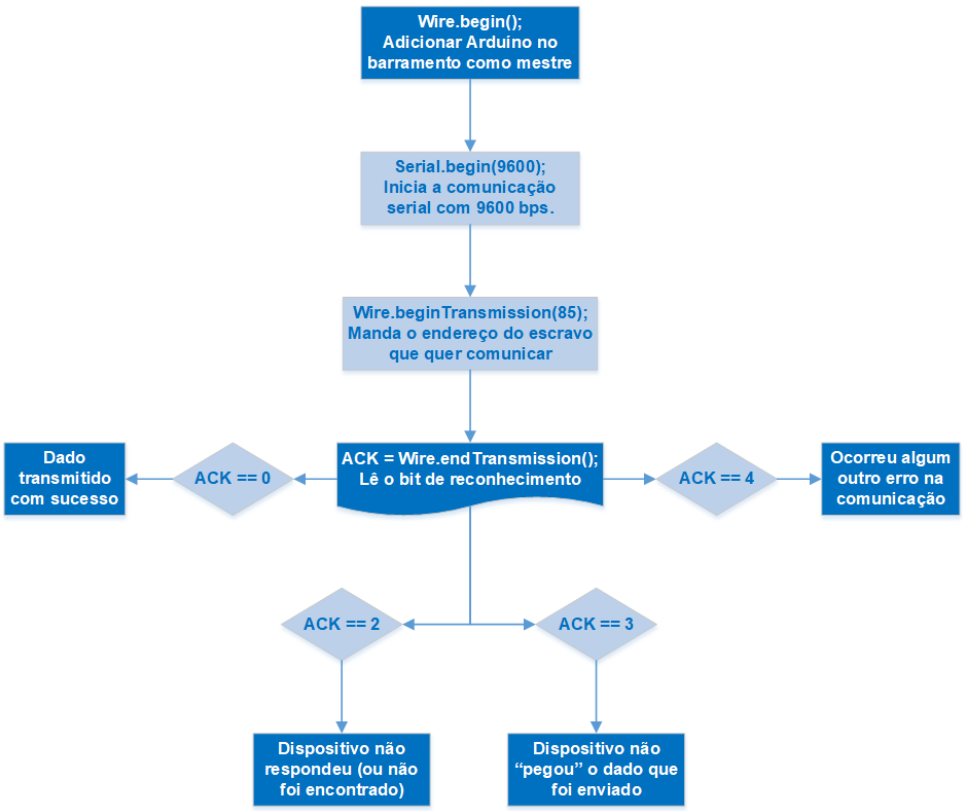
nenhum dispositivo. Então agora é possível notar que em teoria, este barramento suporta 127 dispositivos, porém na prática há somente 112 endereços distintos válidos. (UNKNOWN, 2017)

3.3.0.2 Escrita e Leitura

Todo o trabalho com dispositivos I^2C se resume a uma operação de leitura ou de escrita. Esta secção irá descrever como realizar a comunicação e o processo de escrita e leitura através do microcontrolador arduíno, pois foi o escolhido para a realização deste trabalho.

A biblioteca utilizada para este protocolo é a *Wire.h*. A partir da inicialização da biblioteca no código, o primeiro passo é inserir o microcontrolador no barramento como mestre. Feito isto, inicia-se a comunicação serial, para possibilitar "imprimir" na tela as informações desejadas. Depois então o arduíno "chama" o dispositivo com o qual quer se conectar. É possível ler uma resposta do dispositivo, indicando se o mesmo entendeu o comando e está pronto para receber ou enviar os dados. Esta resposta é chamada de bit de reconhecimento e pode indicar quatro respostas diferentes. Na Figura 3.8 tem estes passos melhor explicados.

Figura 3.8 – Estabelecendo a comunicação I^2C através do arduíno



Fonte: Próprio autor.

A inicialização deve ser feita dentro da função *setup*() do arduíno, a leitura do bit de reconhecimento pode ser feita na função *loop*(). O código para implementação é mostrado na Figura 3.9.

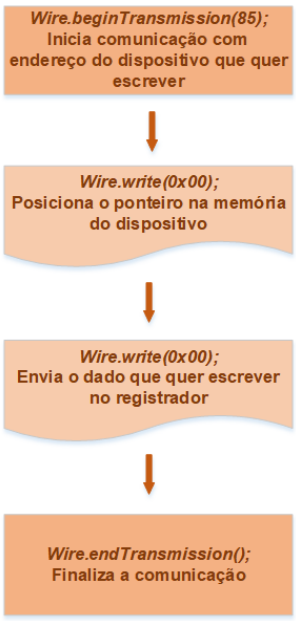
Figura 3.9 – Código para estabelecer a comunicação

```
#include<Wire.h>

int ACK;

void setup() {
  // put your setup code here, to run once:
  Wire.begin(); // adiciona o master no barramento I2C
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  Wire.beginTransmission(85);
  ACK= Wire.endTransmission();
  Wire.endTransmission();
}
```

Fonte: Próprio autor.

Para escrever através de I^2C é preciso iniciar a comunicação, enviar o endereço onde quer escrever para depois enviar o dado que quer escrever e por fim, fechar a comunicação. O fluxograma da Figura 3.10 detalha melhor este processo.

Figura 3.10 – Escrever através de I^2C

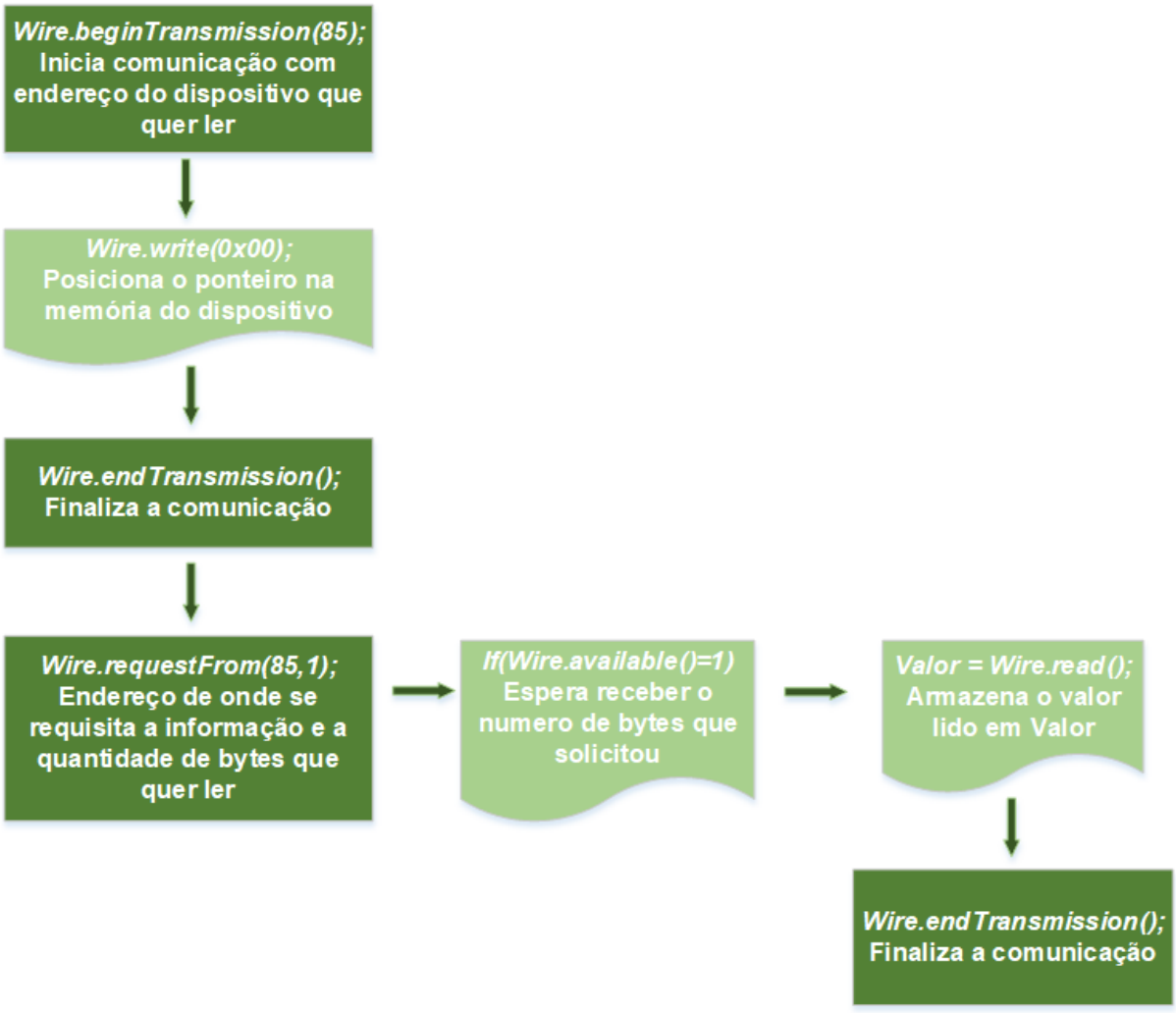


Fonte: Próprio autor.

É importante notar que o mesmo comando é utilizado para informar onde se deseja gravar o dado, como o que se quer gravar.

O processo de leitura é semelhante, porém envolve mais comandos. Neste caso é preciso "abrir" a comunicação para enviar o endereço do dado que se quer ler e "fechar"a comunicação. Para então "abrir" novamente requisitando os dados que deseja. Esse processo está detalhado na Figura 3.11.

Figura 3.11 – Ler através de I^2C



Fonte: Próprio autor.

3.4 CONFIGURAÇÃO DO BQ34110

Para o correto funcionamento do circuito integrado BQ34110 é necessário configurar seus registradores, bem como acessar a memória flash e alterar algumas configurações na mesma. Nesta secção são utilizados muitos termos em inglês, alguns possuem tradução no português

porém optou-se pela língua original com o intuito de facilitar a correlação deste trabalho com o Manual Técnico do Fabricante quando necessário.

3.4.1 Modos de Acesso

Este CI possui quatro modos de acesso: *BOOTROM*, *FULLACCESS*, *SEALED* e *UNSEALED*. Esses modos (exceto o *BOOTROM*) são modos de segurança que controlam a permissão do acesso à memória *flash*. O modo *SEALED* deve ser o modo ativo no dispositivo caso este seja utilizado na composição de um produto final. Ou seja, quando fornecido ao consumidor, este modo deve ser ativo para que o mesmo não possa alterar as configurações feitas, prejudicando o funcionando do produto. Os modos *FULLACCESS* e *UNSEALED* parecem idênticos, porém o primeiro permite o dispositivo trocar diretamente para o modo *BOOTROM* e também criar chaves de segurança para alteração dos modos. O dispositivo vem com a configuração *UNSEALED* de fábrica.

3.4.2 Modos de Funcionamento

Além dos modos de acesso, o bq34110 possui os modos de funcionamento. Modos de funcionamento são quatro: *NORMAL*, *SNOOZE*, *SLEEP* e *SHUTDOWN*. No primeiro modo, o dispositivo está consumindo sua potência normal e pode executar qualquer tarefa permitida por seu modo de acesso. No segundo modo, o *SNOOZE*, o dispositivo "acorda" periodicamente para fazer aquisição de dados e ambos os osciladores (o de baixa frequência e o de alta frequência) estão ativos, porém o consumo do dispositivo é reduzido. No terceiro modo, o *SLEEP*, o dispositivo mantém o oscilador de baixa frequência funcionando porém desliga o de alta frequência. Funcionando com um reduzido consumo de potência quando comparado aos modos anteriores, ainda realiza aquisição de dados e cálculos periodicamente. Por fim, no último modo, *SHUTDOWN*, o dispositivo está completamente desligado e só pode ser ligado através de uma tensão imposta em seu pino CE (*Chip Enable*).

O endereço $0x4159$ da memória *flash* é o endereço da corrente *sleep*. O valor *default* para este endereço é 10 mA. O que significa que quando a corrente medida pelo dispositivo ficar abaixo deste valor por um tempo configurado no endereço $0x4161$ (*Sleep Current Time*), este entra automaticamente em um dos modos de consumo reduzido de potência. A escolha entre esses dois modos depende da configuração de dois registradores, os quais são: *Control_Status* e *Operation Config A*. Para entrar no modo *SNOOZE* é preciso que o bit [SNOOZE] do primeiro registrador esteja setado e o bit [SLEEP] do segundo registrador também. Para permitir entrar

no modo *SLEEP* é preciso que o bit [SNOOZE] esteja 'limpo' e o bit [SLEEP] setado. Mais detalhes sobre estes registradores podem ser encontrados em (Texas Instruments, 2016).

Quando o valor de corrente medido for maior que o valor configurado na memória, o dispositivo volta automaticamente para o modo NORMAL.

3.4.3 Calibração

Para as medições feitas pelo dispositivo serem adequadas é necessário realizar um processo de calibração na placa. A sequência de calibração deve seguir uma ordem especificada abaixo:

1. Realizar o *CC Offset*
2. Calibração da placa (comando *Board Offset*)
3. Realizar calibração da temperatura
4. Realizar calibração da tensão
5. Realizar calibração da corrente
6. Escrever resultados na *flash*

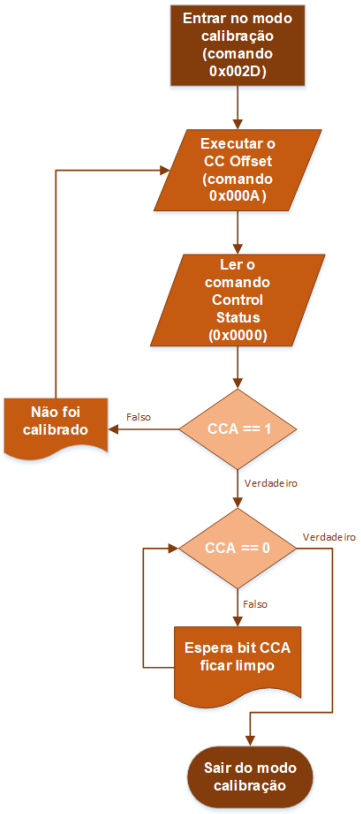
Antes de executar cada um dos processos citados acima é necessário entrar no modo de calibração e no final é preciso sair do mesmo. Este modo é ativado utilizando o comando $0x002D$[4].

3.4.3.1 CC Offset

O processo do *CC Offset* é explicado no fluxograma da Figura 3.12.

[4] O acesso aos comandos é explicado em 3.4.12.

Figura 3.12 – Realização do comando *CC Offset*

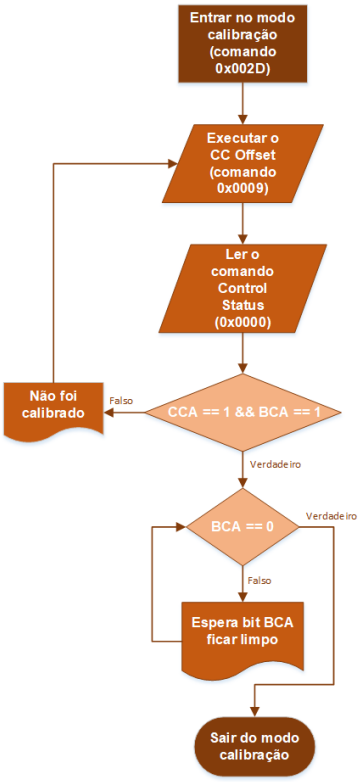


Fonte: Adaptado de (Texas Instruments, 2016).

3.4.3.2 Calibração da Placa

O processo da calibração da placa é semelhante ao do *CC Offset* e pode ser visto no fluxograma da Figura 3.13.

Figura 3.13 – Calibração da Placa

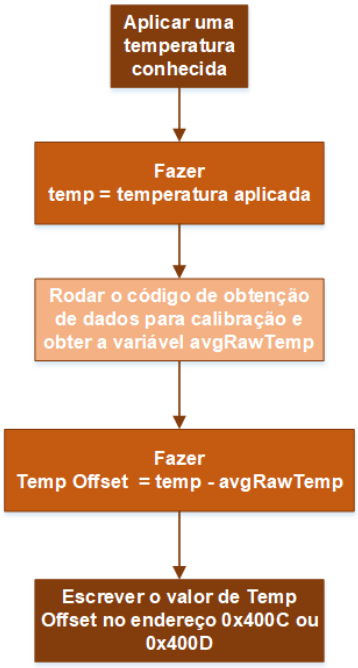


Fonte: Adaptado de (Texas Instruments, 2016).

3.4.3.3 Calibração da Temperatura

Para realizar a calibração da temperatura é necessário rodar um código para obtenção de dados (*avgRawTemp*) que é mostrado no Anexo C.1. Na Figura 3.14 é mostrado o passo a passo deste processo.

Figura 3.14 – Calibração da Temperatura



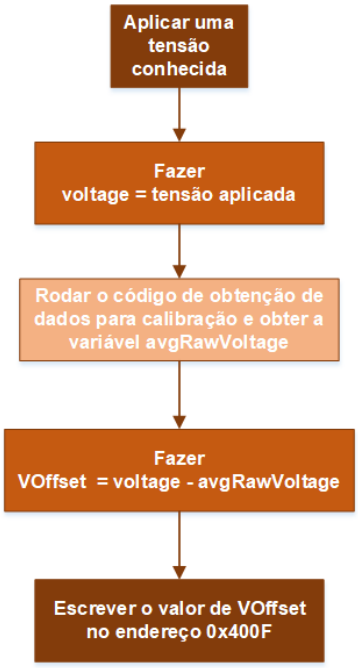
Fonte: Adaptado de (Texas Instruments, 2016).

Se for utilizado o medidor de temperatura interno, o endereço a ser atualizado na memória deve ser o $0x400C$ (*Int Temp Offset*). Caso seja utilizado o medidor externo, o endereço atualizado deve ser o $0x400D$ (*Ext Temp Offset*).

3.4.3.4 Calibração da Tensão

Quando utilizado um divisor de tensão externo ao dispositivo, é necessário colocar o ganho deste circuito no endereço $0x4010$ *Voltage Divider*. Porém antes de fazer esta configuração é necessário realizar o processo de calibração. Como na calibração na temperatura, também é necessário rodar um código para obtenção de dados (*avgRawVoltage*) mostrado no Anexo C.1. Na Figura 3.15 é mostrado o passo a passo deste processo.

Figura 3.15 – Calibração da Tensão



Fonte: Adaptado de (Texas Instruments, 2016).

Para calcular o novo valor a ser colocado no *Voltage Divider* é utilizada a Equação 3.1.

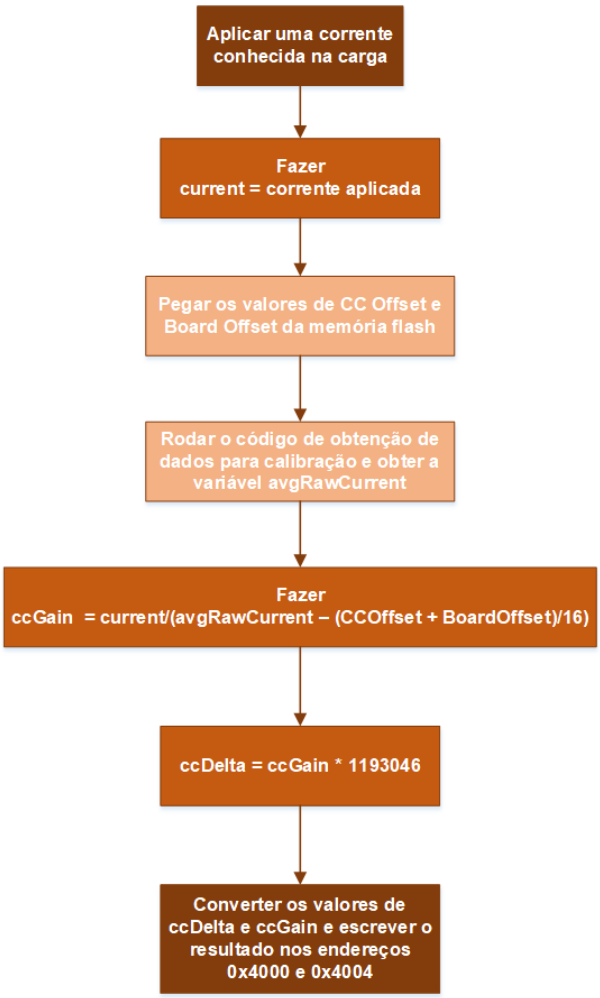
$$novo\_divisor = \frac{V_{aplicada} \cdot VoltageDivider}{avgRawVoltage} \tag{3.1}$$

O valor da variável *Voltage Divider* na equação é o valor que está armazenado no endereço da memória enquanto o programa para obtenção dos dados para calibração é rodado.

3.4.3.5 Calibração da Corrente

Para realizar a calibração da corrente também é necessário rodar o código para obtenção de dados (*avgRawCurrent*) que é mostrado no Anexo C.1. Na Figura 3.16 é mostrado o passo a passo deste processo.

Figura 3.16 – Calibração da Corrente



Fonte: Adaptado de (Texas Instruments, 2016).

Para a transformação dos valores *ccGain* e *ccDelta* é utilizado o processo mostrado em (Texas Instruments, 2016).

3.4.4 Métodos de Carga e Reconhecimento de Término de Carga

Para uma correta operação do dispositivo, é necessário o usuário especificar a tensão com a qual deseja carregar a bateria. Esta configuração pode ser feita manualmente, configurando diretamente os endereços $0x4114$ a $0x4118$. Porém o bq34110 possui vários algoritmos que quando selecionados e, corretamente configurados, fazem este ajuste automaticamente.

O primeiro algoritmo é chamado método JEITA. Este método é acionado pelo bit [JEITA] do registrador *Operation Config A*. Quando acionado, este método seleciona a corrente e tensão de carga baseado na temperatura das células. Primeiro, deve ser configurado na memória, nos endereços $0x4106$ (*T1 Temp*), $0x4108$ (*T2 Temp*), $0x410A$ (*T3 Temp*) e $0x410C$ (*T4 Temp*) quatro

níveis de temperatura. Para cada intervalo entre esses níveis é definida uma corrente (nos endereços 0*x*410*E* (*Charge Current T1 - T2*) a 0*x*4112 (*Charge Current T3 - T4*)) e uma tensão (nos endereços 0*x*410*E* (*Charge Voltage T1 - T2*) a 0*x*4118 (*Charge Voltage T3 - T4*)) para carregar a bateria. Abaixo do menor nível de temperatura e acima do maior nível de temperatura, a carga não deve ser permitida por estar fora dos limites. O ponto ótimo de operação para carga na bateria, no qual a corrente deve ser a maior configurada é o intervalo entre os terceiro e quarto níveis.

Para determinar se houve um término de carga[5], existem três métodos que podem ser selecionados pelo usuário. O primeiro é o método *Current Taper*, o qual precisa satisfazer três condições para o reconhecimento do término da carga. A primeira, durante aproximadamente dois períodos do tempo configurado no endereço 0*x*4122 (*Current Taper Window*), a corrente média medida pelo dispositivo deve ser menor que a corrente configurada no endereço 0*x*411*C* (*Taper Current*). A segunda, a mudança na capacidade acumulada deve ser maior que a mínima capacidade definida no endereço 0*x*411*E* (*Minimum Taper Capacity*). E a terceira, a tensão medida na bateria deve ser maior que a tensão de carga subtraída da tensão configurada no endereço 0*x*4120 (Taper Voltage) (Tensão medida > Tensão de carga - *Taper Voltage*). Então é reconhecido que foi completa a carga das células.

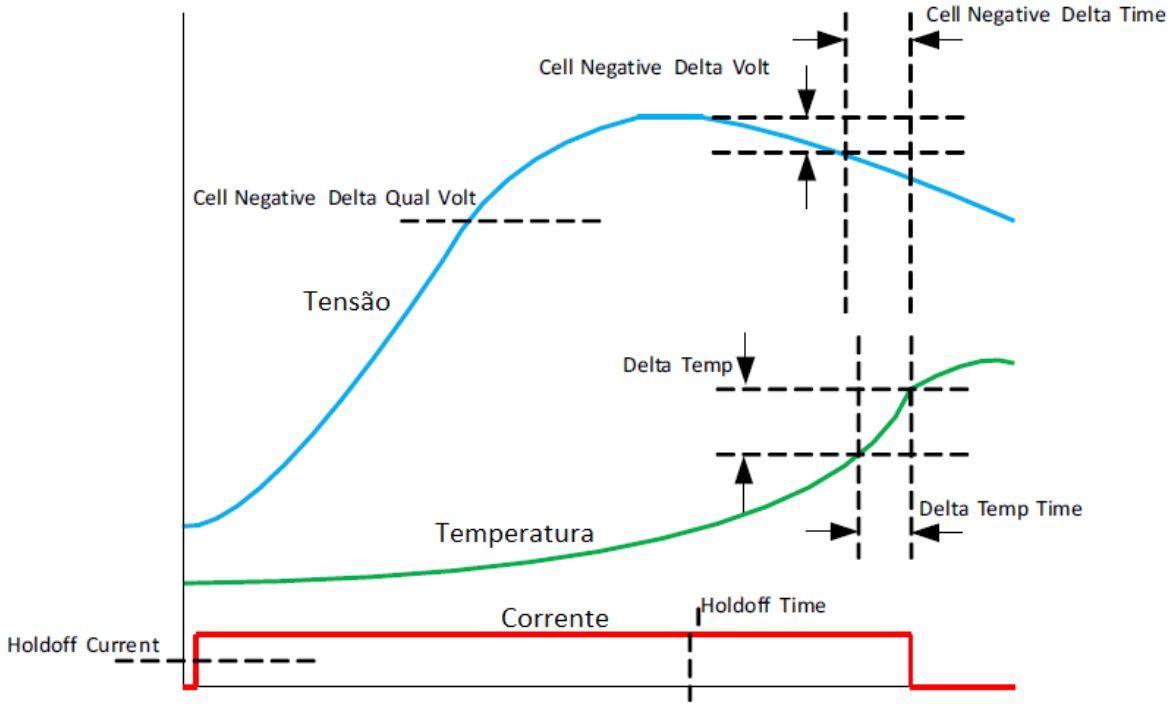
O segundo método é chamado de método da variação de temperatura ou *Delta Temperature*. Neste caso o dispositivo reconhece um aumento na temperatura durante um determinado tempo. Esta temperatura pode ser configurada no endereço 0*x*412*B* (*Delta Temperature*) e o tempo pode ser configurado no endereço 0*x*412*D* (*Delta Temperature Time*). Usualmente é escolhida a relação 1°C/minuto ou 2°C/120s ou 3°C/180s. Além desse, um segundo *timer* (*Holdoff Timer - endereço* 0*x*412*F*) inicia a contagem quando a corrente de carga da bateria ultrapassa o valor configurado no endereço 0*x*4131 (*Holdoff Current*) e a temperatura medida está acima do configurado no endereço 0*x*4133 (*Holdoff Temperature*). Até este tempo expirar, a contagem do primeiro *timer* é suspensa. Quando a corrente ou a temperatura estiverem abaixo desses níveis pré definidos, o *Holdoff Time* é resetado e reinicia quando ambas as condições forem satisfeitas novamente.

O terceiro é o método da variação negativa de tensão (*Negative Delta Voltage Method*). O dispositivo detecta o término da carga quando há uma variação negativa de tensão configurada no endereço 0*x*4135 (*Cell Negative Delta Voltage*) por um tempo determinado no endereço 0*x*4137 (Cell Negative Delta Time), juntamente com o requisito da tensão medida na bateria estar acima do valor configurado no endereço 0*x*4138 (Cell Negative Delta Qual Voltage).

[5]Por término de carga entende-se que houve o fim do processo de carregar a bateria, e não o fim da carga da bateria.

Na Figura 3.17 é possível ver as curvas de corrente, tensão e temperatura ultrapassando os níveis apresentados acima.

Figura 3.17 – Métodos para reconhecimento de término de carga



Fonte: Adaptado de (Texas Instruments, 2016)

Além dos métodos citados acima, há um outro procedimento inovador presente no BQ34110. Este método é específico para aplicações onde um determinado nível de capacidade da bateria é requerido, e a descarga das células é rara, como baterias de fontes ininterruptas de energia. Conforme passa o tempo e a bateria começa a ficar antiga, a tensão de carga necessária para manter o mesmo nível de capacidade na bateria vai aumentando. Com um sistema de controle com tensão fixa, é necessário aplicar uma sobretensão na bateria no começo de sua vida útil, para que no futuro esta mesma tensão ainda mantenha o nível mínimo requerido. Este método chamado de *WHr Charging* age de forma diferente. Vai adaptando esta tensão para sempre ter o nível necessário. O próprio algoritmo configura a tensão de carga (*ChargingVoltage*) para o número de células multiplicado pela tensão escolhida para o intervalo entre os segundo e terceiro níveis configurados pelo primeiro método apresentado neste trabalho, o método JEITA (número de células x *Cell Charge Voltage T2 - T3*).

A bateria é carregada com este nível de tensão e reconhece o término da carga baseado em três condições. A primeira é quando a tensão medida na célula da bateria é maior que a tensão de carga selecionada diminuída da tensão configurada no endereço 0*x*4120 (*Taper Voltage*) (Tensão medida > Tensão de carga - *Taper Voltage*). A segunda é que a corrente

medida deve ser menor que a corrente configurada no endereço 0*x*411*C* (*Taper Current*) por dois períodos de tempo determinados no endereço 0*x*411*E* (*Current Taper Window*). E por fim, a capacidade remanescente calculada pelo dispositivo deve ser maior que o valor configurado no endereço 0*x*4127 (*WHr Termination Capacity*) ou a tensão de carga deve ser igual à tensão determinada no endereço 0*x*4123 (*Max Charge Voltage*). Caso as duas primeiras condições sejam satisfeitas porém a capacidade remanescente não atingiu o valor mínimo necessário, então a tensão de carga é alterada para a tensão armazenada no endereço 0*x*4009 (*Last Charge Voltage T2 - T3*) somada com a tensão do endereço 0*x*4125 (WHr CV Step), resultando (*Last Charge Voltage T2 - T3* + WHr CV Step). É esta a parte do procedimento que é inovadora, pois ajusta a tensão de carga adicionando um pequeno "degrau" de tensão à tensão de carga medida no último processo de carga completo.

3.4.5 Inibição de Carga

É possível bloquear a carga das células baseado na temperatura, configurando os endereços 0*x*4100 (*Charge Inhibit Temp Low*) e 0*x*4102 (*Charge Inhibit Temp High*). Na primeira posição de memória citada é configurado um valor de temperatura abaixo do qual a carga é suspensa. Na segunda posição de memória citada é determinado um valor acima do qual a carga também é suspensa. Estes valores vem de fábrica como sendo 0ºC e 45ºC respectivamente. Quando uma dessas condições ocorre, uma *flag* é setada, o *bit* $[CHGINH]$ do registrador de leitura *Battery Status* (será abordado em detalhes mais a frente) é colocado em um. Para este *bit* ser limpo é necessário a temperatura voltar para o range permitido com uma pequena margem de segurança. Esta margem de segurança é determinada no endereço 0*x*4104 (*Temp Hys*), e é somada ao valor mínimo e diminuída do valor máximo. Ou seja, se esta margem de segurança for configurada para 5ºC por exemplo, é necessário que a temperatura suba acima de 5ºC ou caia abaixo de 40ºC para a carga ser reestabelecida.

3.4.6 Medição de Temperatura

O dispositivo tem a capacidade de avisar, através de *flags* quando a temperatura da bateria está acima ou abaixo do desejado durante uma carga ou descarga.

Se durante a carga, a temperatura medida passar do valor determinado em 0*x*4170 (*OT Chg*) por um período de tempo maior ou igual ao configurado em 0*x*4172 (*OT Chg Time*) e a corrente medida for maior que o nível mínimo de corrente necessário para caracterizar uma carga 0*x*4164 (*Charge Detection Threshold*), então uma *flag* é setada através do *bit* $[OTC]$ no registrador de leitura *Battery Status*.

O mesmo ocorre para o processo de descarga. Se a temperatura medida passar do valor determinado em $0x4175$ (*OT Dsg*) por um período de tempo maior ou igual ao configurado em $0x4177$ (*OT Dsg Time*) e a corrente medida for menor que o negativo do nível mínimo de corrente necessário para caracterizar uma descarga $0x4162$ (*Discharge Detection Threshold*) (Corrente < - *Discharge Detection Threshold*). Então uma *flag* é setada através do *bit* $[OTD]$ no registrador de leitura *Battery Status*.

Se durante a carga a temperatura medida cair abaixo do valor configurado em $0x417A$ (*UT Chg*) por um período maior ou igual ao determinado em $0x417C$ (*UT Chg Time*) e a corrente medida for maior que o nível mínimo de corrente necessário para caracterizar uma carga $0x4164$ (*Charge Detection Threshold*), então uma *flag* é setada através do *bit* $[UTC]$ no registrador de leitura *Battery Status*.

Analogamente, para a descarga se a temperatura medida cair abaixo do valor configurado em $0x417F$ (*UT Dsg*) por um período maior ou igual ao determinado em $0x4181$ (*UT Dsg Time*) e a corrente medida for menor que o negativo do nível mínimo de corrente necessário para caracterizar uma descarga $0x4162$ (*Discharge Detection Threshold*) (Corrente < -*Discharge Detection Threshold*), então uma *flag* é setada através do *bit* $[UTD]$ no registrador de leitura *Battery Status*.

3.4.7 Alertas de Condição da Bateria

O dispositivo indica através de *flags* quando a capacidade da bateria ultrapassa certos níveis configurados.

Para a indicação de que a tensão da bateria está baixa, é determinado um valor em $0x4184$ (Battery Low Set Threshold). Quando a tensão é medida abaixo deste valor por um período de tempo determinado em $0x4186$ (Battery Low Time) o bit $[BATLOW]$ é colocado em um. Quando a tensão é maior que este nível, o bit $[BATLOW]$ do registrador de leitura *Battery Status* é limpo.

Para a indicação de que a tensão da bateria está abaixo de um valor configurado, é determinado o nível desejado em $0x4189$ (*Battery High Set Threshold*). Quando a tensão é maior que este nível, o bit $[BATHIGH]$ do registrador de leitura *Battery Status* é limpo. Quando a tensão é medida abaixo deste valor por um período de tempo determinado em $0x418B$ (*Battery High Time*) o bit $[BATHIGH]$ é setado. Esta indicação é muito útil para aplicações onde as células raramente são descarregadas e devem permanecer a maior parte do tempo em um nível específico. Assim, quando este nível cair, a *flag* é setada avisando o usuário que deve aumentar a tensão de carga para voltar a atingir o nível desejado.

Também há uma indicação para quando o estado de carga cai abaixo de um valor estipulado. Este valor é determinado no endereço $0x418E$ (*SOC Low Threshold*), e quando o SOC medido pelo dispositivo estiver abaixo deste valor, o *bit* $[SOCLOW]$ no registrador de leitura *Battery Status* é colocado em um. Para limpar este *bit* é necessário que o SOC medido ultrapasse um valor de recuperação, definido em $0x418F$ (*SOC Low Recovery*).

3.4.8 Alertas de Capacidade Remanescente

O dispositivo pode ser configurado para avisar durante a carga ou descarga que a capacidade remanescente ultrapassou um nível determinado. Quando a corrente medida é positiva, ou seja, está em processo de carga, e a capacidade remanescente medida passa do valor configurado através do comando *BLTChargeSet*, uma *flag* é setada no *bit* $[BLT]$ do registrador de leitura *Battery Status*. Quando a corrente medida é negativa, ou seja, está em processo de descarga, e a capacidade remanescente medida é menor que o valor configurado através do comando *BLTDischargeSet*, a *flag* do *bit* $[BLT]$ do registrador de leitura *Battery Status* também é setada.

O comando *BLTChargeSet* pode ser configurado através dos endereços $0x36$ e $0x37$ e o comando *BLTDischargeSet* através dos endereços $0x34$ e $0x35$. Ambos conforme explicado na secção 3.4.12.

3.4.9 O Algorítmo de Medição

O algorítmo utilizado pelo dispositivo para estimar as características da bateria, acumula as medições de carga e descarga, bem como a medição de descarga própria. É necessário as células terem uma carga completa e, posteriormente uma descarga completa para que o algorítmo consiga estimar mais precisamente os dados.

O primeiro passo é informar a capacidade nominal da bateria, na posição da memória $0x41F5$ chamada *Design Capacity* em unidades de mAh.

O principal contador do algoritmo de estimação do dispositivo é o de capacidade remanescente, que pode ser lido através dos comandos $0x10$ e $0x11$ (*Remaining Capacity (RC)*). Este comando representa a carga disponível restante na bateria no momento da leitura. O RC cresce durante uma carga até atingir um valor máximo (a capacidade de bateria cheia), e decresce durante uma descarga ou descarga própria até atingir zero.

Outro contador importante é o de capacidade total, o qual pode ser lido nos comandos $0x12$ e $0x13$ (*Full Charge Capacity (FCC)*). Este valor representa a capacidade da bateria cheia como referência para a estimação da carga remanescente e estado de carga. Na inicialização, o FCC carrega o valor contido no endereço $0x40C0$ (*Learned Full Charge Capacity*) da memória

flash. Nas descargas subsequentes o dispositivo atualiza este valor para a última capacidade de descarga medida nas células.

Um contador interno, ao qual não se tem acesso e serve apenas para o dispositivo realizar suas estimações, é o contador de registro de descarga (ou Discharge Count Register - DCR). Este cresce durante a descarga da bateria, independente de RC, e é responsável por estimar a atividade de descarga, a carga da bateria e controlar o incremento da descarga própria. Este contador é inicializado com o valor da capacidade de bateria cheia diminuído da capacidade remanescente (FCC - RC). Porém este cálculo só ocorre se RC for próximo ao valor configurado como "quase cheio" na memória, no endereço $0x4298$ (*Near Full*). O DCR pára seu contador quando atinge um determinado nível de capacidade, configurado no endereço $0x427D$ (*Battery Low%*).

Através do registrador *Cycle Count*, endereço $0x40C5$ da memória, é possível saber quantos ciclos a bateria experienciou. É preciso configurar uma percentagem da carga total da bateria no endereço $0x41FB$ chamado *Cycle Count Percentage*. O valor *default* determinado é 90%. Isso significa que quando houver uma descarga de 90% da capacidade da bateria foi completo um ciclo e o contador *Cycle Count* irá acrescentar uma unidade.

3.4.9.1 Monitoramento da Tensão e Ajuste da Capacidade

O bq34110 monitora a bateria através de três níveis baixos de tensão: EDV0, EDV1 e EDV2. A definição desses níveis pode ser feita manualmente pelo usuário, ou pelo próprio dispositivo. Essa escolha é definida no bit $[EDV\_CMP]$ do registrador *CEDV Gauging Configuration* detalhados na secção 3.4.11. Quando escolhido o modo manual, é necessário configurar os valores dos três níveis nos endereços $0x425A$ *Fixed EDV0*, $0x425D$ *Fixed EDV1* e $0x4260$ *Fixed EDV2*. Quando escolhido o modo automático, o dispositivo escolhe esses níveis baseado na corrente de descarga e na temperatura. Essa detecção de nível é desabilitada quando a corrente excede o valor configurado como sobrecorrente (no endereço $0x4291$ chamado *Overload Current*).

Os níveis são associados a estimações do estado de carga relativo (é relativo pois o dispositivo estima o estado de carga como uma percentagem de FCC). O nível EDV0 é associado à 0%, EDV1 à 3% e EDV2 ao valor definido em *BatteryLow%* (citado em 3.4.9).

O circuito integrado tem seus níveis de tensão (EDVx) ajustados somente quando o valor da corrente medida é maior que três vezes a capacidade da bateria definida para o respectivo nível dividido por 32 ($Corrente > 3C/32$, onde C é a capacidade relativa ao nível analisado no momento). Nenhum nível é definido quando esta relação não é satisfeita. Se a tensão medida na bateria atinge o nível sem que a capacidade remanescente (RC) esteja no valor associado, o

contador RC é ajustado para o valor que condiz com o nível de tensão atingido. Se ocorre o contrário, a capacidade é atingida porém a tensão ainda está acima do nível configurado para àquela capacidade, então o contador RC é paralizado até o nível de tensão ser alcançado.

A equação utilizada para o cálculo dos níveis de tensão é:

$$EDV_{0,1,2} = n(EMF \cdot FBL - |I_{LOAD}| \cdot R0 \cdot FTZ) \tag{3.2}$$

Onde:

- EMF é a tensão a vazio, ou seja, sem carga conectada, e deve ser maior que o maior nível de tensão definido (EDV2). Este valor pode ser configurado no endereço $0x424D$ da memória *flash*.
- I_{LOAD} é a corrente de descarga que passa pela carga.
- n é o número de células configurado em $0x4155$ (*Number of Series Cells*).
- FBL é o fator que ajusta os níveis de tensão pela capacidade e temperatura da bateria para que sejam satisfeitas as características a vazio da bateria.

  A fórmula do fator de ajuste é uma função de 4 variáveis, como mostrado em 3.3.

$$FBL = f(C0, C + C1, T) \tag{3.3}$$

  Sendo:

- C pode ser 0%, 3% ou *BatteryLow%*, para EDV0, EDV1 e EDV2 respectivamente.
- C0 é o fator de ajuste da capacidade relativo ao nível de tensão. Este fator é armazenado no endereço $0x424F$ da memória *flash*.
- C1 é a capacidade remanescente na bateria quando a tensão medida atinge o nível de tensão mais baixo (EDV0). Este fator é armazenado no endereço $0x4258$.
- T é a temperatura medida em Kelvin.

A parcela da Equação 3.2 $R0 \cdot FTZ$ representa a resistência da célula como uma função da capacidade e temperatura. Assim, FTZ é obtido em função de cinco variáveis como mostrado em 3.4.

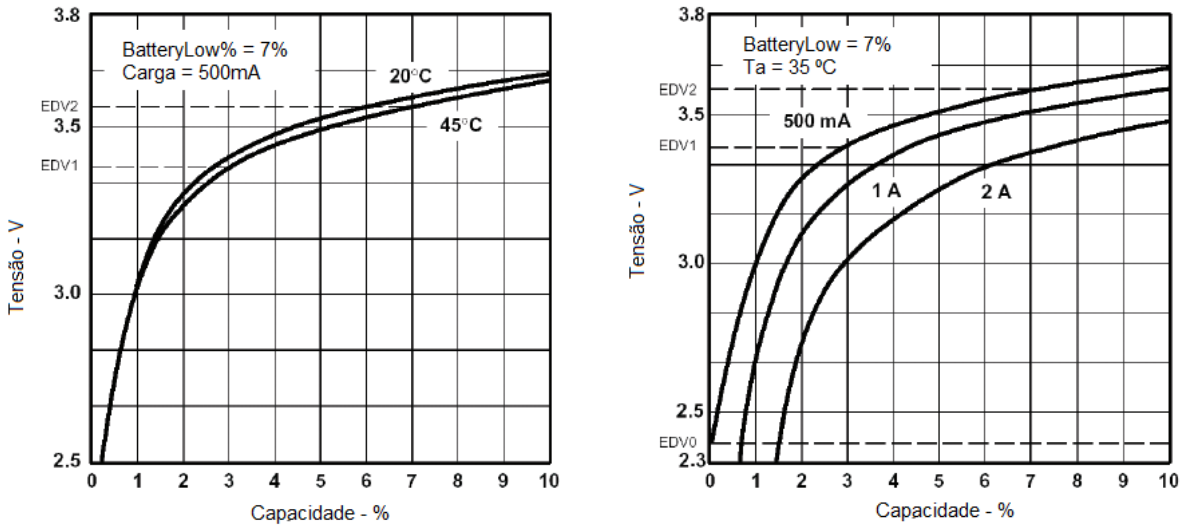
$$FTZ = f(R1, T0, C + C1, TC) \tag{3.4}$$

Onde:

- R0 é o fator de dependência de primeira ordem armazenado no endereço $0x4251$ da memória.
- R1 ajusta a variação da impedância pela capacidade da bateria e pode ser visualizado em $0x4255$.
- T0 ajusta a variação de impedância pela temperatura, armazenado no endereço $0x4253$.
- TC ajusta a variação de impedância para temperaturas consideradas muito frias (<23°C, seu valor pode ser encontrado em $0x4257$.

Nos dois gráficos da Figura 3.18 é possível notar como se comporta a relação 'EDV x Capacidade' em diferentes temperaturas ambiente e diferentes cargas. Nota-se que quando fixo o valor da tensão, com o aumento da temperatura aumenta a capacidade relacionada ao mesmo. Analogamente com o aumento da carga, fixado um nível de tensão, a capacidade configurada a este será maior quanto maior for a carga.

Figura 3.18 – Variação dos níveis de tensão com a variação de temperatura e da carga



Fonte: Adaptado de (Texas Instruments, 2016)

3.4.10 Coleta de dados

O BQ34110 dispõe de uma função para coletar e armazenar dados da vida da bateria. Esta função pode ser ativada através do *bit* $[LF\_EN]$ do registrador *ManufacturingStatus* descrito em detalhes em 3.4.11.2. A função chamada *Lifetimes* faz a coleta de dados de temperatura, tensão e corrente periodicamente numa frequência programada. Entre os dados coletados estão: a

máxima corrente de carga e descarga, máxima tensão da células, número de atualizações na memória e temperatura mínima e máxima atingida.

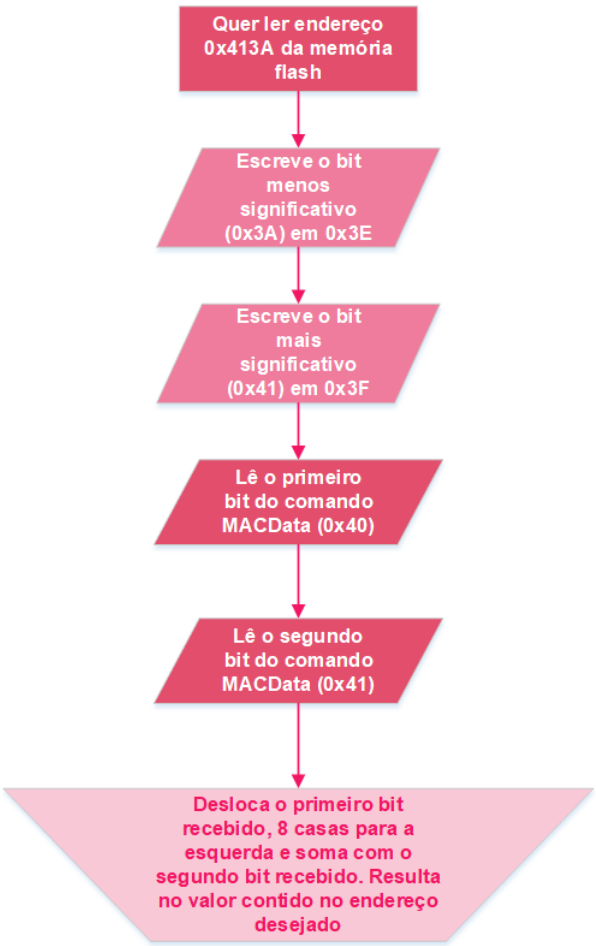
Os valores medidos são comparados com os valores armazenados. Caso o valor medido seja maior que o máximo armazenado, é visto se essa diferença é superior ao valor mínimo para se fazer uma atualização. Esses valores mínimos são configurados nos endereços $0x4190$ (*Temperature Resolution*), $0x4191$ (*Current Resolution*) e $0x4192$ (*Voltage Resolution*). Por exemplo, o valor *default* contido em *Temperature Resolution* é 10ºC, porém a unidade deste endereço é 0.1ºC. Sendo assim, quando a temperatura medida for maior que a temperatura máxima armazenada e essa diferença for superior a 1ºC ($10 \cdot 0.1 = 1$ºC), então o valor de máxima temperatura ($0x4080$ chamado *Max Temperature*) será atualizado. Além dessa diferença mínima, há outra condição para a atualização desses dados. É preciso que o valor se mantenha por pelo menos 60 segundos. Caso isso não ocorra os dados não serão atualizados. Este processo ocorre de forma análoga para os valores de mínimo.

A lista completa de dados que podem ser coletados e armazenados é encontrada em (Texas Instruments, 2016).

3.4.11 Configuração dos Registradores

Para realizar a configuração dos registradores é necessário primeiro entender como acessá-los na memória *flash*. Isso porque neste dispositivo todos os endereços da memória são compostos por dois *bytes*, como por exemplo em hexadecimal $0xFFFF$. Entretanto, como visto anteriormente, a comunicação I^2C só envia um *byte* por vez. Portanto é necessário fazer esse acesso através de um comando chamado *Manufacturer Access Control*. Este comando possui dois endereços de um *byte* cada, os quais em hexadecimal são $0x3E$ e $0x3F$. Então, para acessar um determinado endereço na *flash*, deve-se quebrar o endereço desejado (que possui dois bytes) em dois endereços de um byte. O *byte* menos significativo deve ser escrito no endereço $0x3E$ e o mais significativo no endereço $0x3F$. Quando escrito um endereço no comando, este "junta"os bytes e faz com que o ponteiro da memória se posicione no endereço escrito ao mesmo. E então, o dado contido neste endereço da memória irá aparecer em outro comando, chamado *MACData*, o qual pode ser lido através dos endereços $0x40$ a $0x41$. O fluxograma da Figura 3.19 mostra o passo a passo para fazer este processo de leitura.

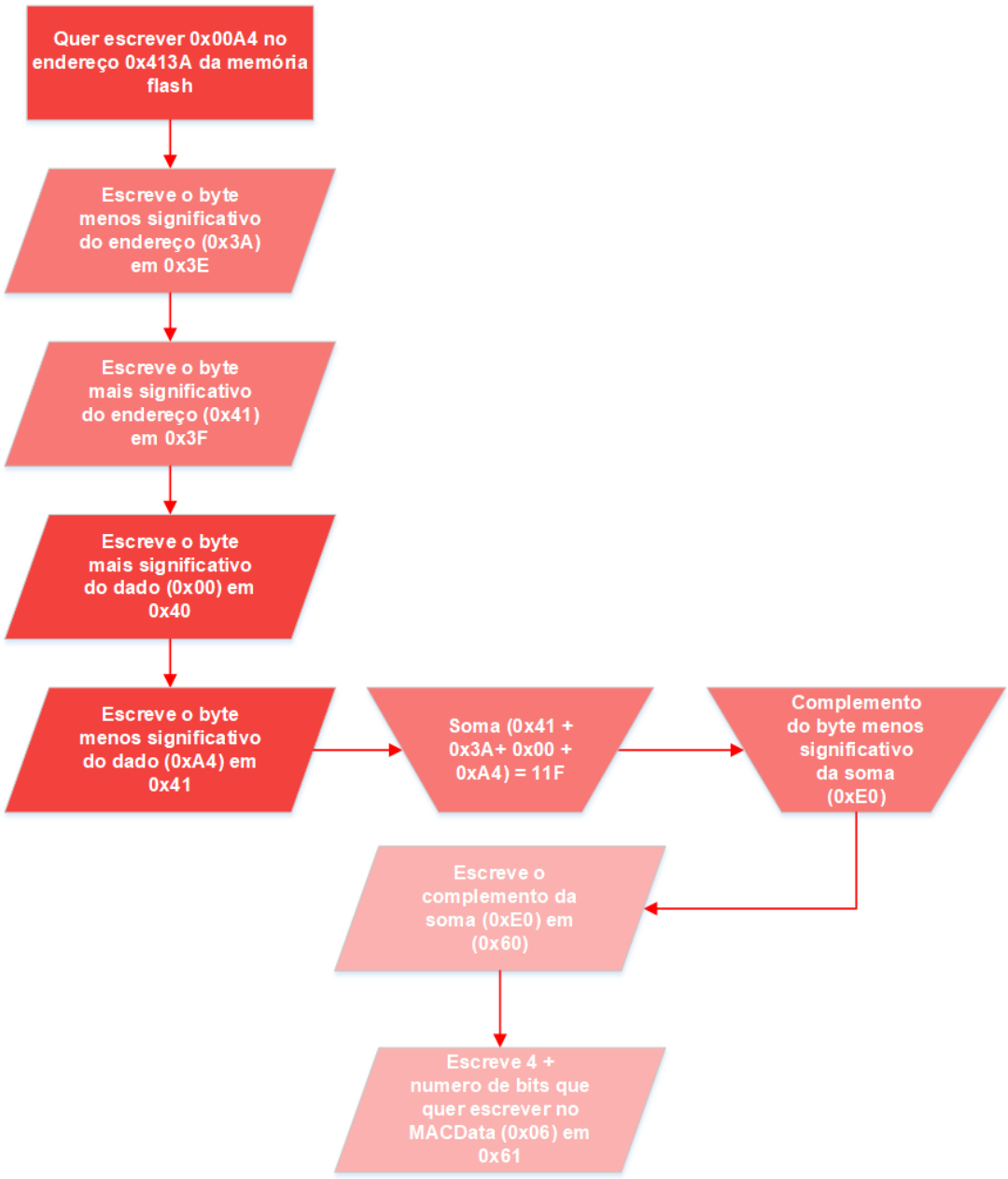
Figura 3.19 – Leitura de endereços na memória *flash*



Fonte: Próprio autor.

Para o processo de escrita em um endereço da memória *flash* é um pouco mais complicado. É necessário utilizar mais dois comandos: *MACDataSum* e *MACDataLen*. O primeiro utiliza o endereço em hexadecimal $0x60$ e neste deve ser escrito o *byte* menos significativo do complemento da soma dos *bytes* escritos no *Manufacturer Access Control* e no *MACData*. O segundo, utiliza o endereço em hexadecimal $0x61$ e neste deve ser escrito a soma do número 4 com o número de *bytes* que se quer escrever. Na Figura 3.20 encontra-se um exemplo para este processo.

Figura 3.20 – Escrita de endereços na memória *flash*



Fonte: Próprio autor.

Um detalhe de grande importância, é o fato de que a memória *flash* só pode ser atualizada (ou seja, o usuário só poderá escrever nos registradores ou endereços da memória) quando a tensão medida for maior que a tensão configurada no endereço 0x4157 (*Flash Update OK Voltage*). Com exceção se o dispositivo estiver em modo de calibração. Isso ocorre porque a programação da *flash* causa uma queda na tensão do regulador interno ao dispositivo. Sendo assim, a tensão de fábrica programada no endereço citado acima é 2800 mV. Este valor deve ser selecionado de forma que a tensão V_{cc} não caia abaixo de seu mínimo (que é 2.4V) durante configuração da *flash*. Portanto, não é aconselhável alterar este valor.

Para configurar algumas funções do dispositivo, são quatro registradores principais que devem configurados:

1. *Operation Config A*
2. *Manufacturing Status Init*
3. *Pin Control Config*
4. *CEDV Gauging Configuration*

3.4.11.1 Operation Config A

Composto por dois *bytes*[6], esse registrador configura várias funções do circuito integrado. No bit 15 [$TEMPS$] é selecionado se é desejado medir a temperatura por um termistor interno (limpa o bit colocando zero) ou externo (seta o bit colocando um). É neste registrador também que se configura se é permitido ou não o dispositivo entrar no modo *SLEEP* através do bit 9 [$SLEEP$] conforme explicado anteriormente. O bit 7 [$JEITA$] serve para ativar a escolha da tensão e corrente de carga das baterias, baseado nas especificações de temperatura pelo método *JEITA* (conforme explicado na secção 3.4.4). Também pode-se escolher qual o referencial utilizado no dispositivo, no bit 6 [$GNDSEL$]. Caso este bit seja limpo, o pino VSS é selecionado como referência, caso seja setado o dispositivo escolhe o SRN para seu referencial. Os bits 5 [$NIMH\_CHG\_EN$], 4 [NI_DT] e 3 [NI_DV] são para selecionar qual o método utilizado para reconhecer quando houve um processo de carga completa na bateria. O primeiro é para ativar o uso do algoritmo NiXX, o segundo do algoritmo $\Delta T \ / \ \Delta t$ e o último do algoritmo $-\Delta V$. Todos foram descritos em 3.4.4. Os demais *bits* são valores reservados e devem permanecer conforme *default*.

3.4.11.2 Manufacturing Status Init

Este registrador composto por dois *bytes*[7], é responsável por ativar (escrevendo 1 no bit respectivo a esta função) ou desativar (escrevendo 0 no respectivo bit) diversas funções desempenhadas pelo dispositivo. No bit 6 [WHR_EN] é possível escolher se o algoritmo *WHr Charge Termination* explicado na secção 3.4.4 deve ser habilitado ou não. O bit 5 [LF_EN] ativa a coleta de dados da vida da bateria (*Lifetime Data Collection*). O bit 4 [$PCTL\_EN$] quando ativo permite o usuário a controlar manualmente os pinos de alerta (*ALERT1* e *ALERT2*), o pino de

[6]Os bits não detalhados são reservados e não devem ser alterados
[7]Os bits não detalhados são reservados e não devem ser alterados

proteção (LEN) e o pino de redução do consumo de potência (VEN). O bit 3 [EOS_EN] ativa a função de determinar o fim da vida da bateria (*End of Service*). O bit 2 [$IGNORE\_SD\_EN$] escolhe se é ignorada (escrevendo um) a descarga própria da bateria (descarga que ocorre mesmo quando o circuito está a vazio ou a carga não esta consumindo corrente). Os bits 1 [$ACCHG\_EN$] e 0 [$ACDSG\_EN$], quando setados, permitem que sejam integrados os valores de corrente positivos durante a carga e negativos durante a descarga.

3.4.11.3 Pin Control Config

Este registrador, diferente dos anteriores, possui apenas um *byte*[8]. O bit 4 [VEN_EN] quando setado, coloca em estado "alto" o pino VEN do bq34110, ativando o circuito divisor de tensão contido na placa. Este circuito é responsável por reduzir a tensão das células até uma tensão suportável pelo pino BAT. Quando limpo, este bit habilita o divisor de tensão interno ao CI. Os bits 3 e 1 [$ALERT2\_POL$] e [$ALERT1\_POL$] controlam a polaridade do pino alerta correspondente. Ou seja, quando escrito um, o pino de alerta terá um estado "alto" quando satisfeitas suas condições. Por fim, os bits 2 e 0 [$ALERT2\_EN$] e [$ALERT1\_EN$] são utilizados para controlar se os alarmes devem ou não gerar uma interrupção (ativo quando escrito um).

A configuração dos alarmes é feita através dos endereços $0x413E$ a $0x414B$, nos quais cada bit corresponde à uma função. Para haver uma sinalização nos alarmes, é necessário ocorrer a coincidência do bit desta função estar ativado[9] nesses registradores e sua *flag* no respectivo registrador de leitura estar em um. Por exemplo, se no endereço $0x413F$ é escrito $0x40$ (em binário 0100 0000) foi ativo a função do bit 6, que é a *SOCLOW*. Portanto, quando for reconhecido que o SOC da bateria está baixo e sua *flag* no registrador de leitura *BatteryStatus* for setada, haverá uma interrupção. A lista completa das funções que podem ser ativadas nos endereços citados acima pode ser encontrada em (Texas Instruments, 2016).

3.4.11.4 CEDV Configuration

Este registrador tem suas configurações diretamente ligadas às medições e estimações feitas pelo dispositivo. Através do bit 10 [FCC_FOR_VDQ] é possível selecionar se o circuito integrado só atualiza o registrador FCC (citado em 3.4.9) quando detectada uma carga completa da bateria[10]. O bit 8 [FCC_LIMIT] define se o FCC pode ser atualizado com um valor maior que a capacidade nominal (DC) definida pelo usuário. Para implementar este limite, deve ser escrito um neste bit. O bit 5 [$FIXED\_EDV$] quando setado, usa para o mais baixo nível de

[8] Os bits não detalhados são reservados e não devem ser alterados

[9] O bit é ativado quando escrito um.

[10] Ativa essa característica setando o bit.

tensão (EDV0) o valor fixo configurado manualmente na memória *flash*. E as compensações e cálculos feitos pelo dispositivo para EDV1 e EDV2 não poderão ser menores que o valor determinado por EDV0. Quando este bit permanecer limpo, então todos os níveis de tensão serão determinados pelo bq34110. O bit 4 [SC] quando setado, muda o cálculo no contador de registro de descarga (DCR - citado em 3.4.9). Se $SC = 1$ então DCR é inicializado com $FCC - RC - (FCC/128)$. O bit 3 [EDV_CMP] quando limpo configura os níveis de tensão (EDV0,1,2) para serem determinados manualmente. Quando setado, o dispositivo faz o cálculo automaticamente. O bit 1 [$CSYNC$] define se o contador RC é igualado a FCC quando ocorre uma carga completa ou se permanece imutável. Para igualar, o bit deve ser colocado em 1. Por fim, o bit 0 [CCT] tem a ver com o contador de ciclos experienciados pela bateria (*Cycle Count* - citado em 3.4.9). Quando setado, o dispositivo verifica se ocorreu um ciclo estimando uma percentagem de FCC. Quando limpo, o dispositivo faz o mesmo porém com uma percentagem de DC (*default*).

3.4.11.5 Controle de Carga

O BQ34110 não faz simplesmente o monitoramento da carga e descarga na bateria, mas também o controle destes processos, bem como a escolha de habilitar ou não a alimentação da carga conectada ao sistema. Para este controle são utilizados dois registradores de comprimento um *byte* cada, chamados *Direct Charge Pin Control* e *Charge Level Pin Control*. Através do primeiro, o controle é feito diretamente e seus bits só escolhem qual o pino será utilizado para esta função. São quatro pinos que podem ser configurados: *ALERT1*, *ALERT2*, *VEN* e *LEN*[11]. O bit 3 [$DCHGPOL$] define a polarização dos pinos de controle. Setando este bit será gerada uma saída em estado "alto" quando o processo de carga (ou alimentação da carga) estiver liberado. Os bit 2, 1 e 0 escolhem o pino que irá realizar este controle. Para 100 nesses bits o pino selecionado é *ALERT1*, 101 *ALERT2*, 110 VEN e 111 LEN.[12]

O registrador *Charge Level Pin Control* possui um controle discreto da tensão. Primeiro, deve ser selecionado nos bits 5 [VEN_PUP] e 4 [LEN_PUP] como os pinos VEN e LEN irão gerar uma saída de estado "alto". Quando setados, os pinos usarão uma saída lógica forçada em 1 e quando limpos, é selecionada uma saída de estado de alta impedância. Então os bits 3 a 0 selecionam o pino da mesma forma que o comentado no primeiro registrador. Apenas 3 desses pinos podem ser selecionados ao mesmo tempo. A tensão de carga é definida pela combinação dos níveis dos três bits selecionados. A Tabela 3.3 mostra cada combinação de bit para a respectiva tensão determinada.

[11] Os quatro primeiros bits são reservados e não devem ser alterados.
[12] Qualquer valor diferente dos citados não ativa nenhum pino.

Tabela 3.3 – Níveis de tensão para combinação de bits

Nível	Combinação de Bits	Tensão (mV)
Charge Voltage Level A	000	3900
Charge Voltage Level B	001	3950
Charge Voltage Level C	010	4000
Charge Voltage Level D	011	4050
Charge Voltage Level E	100	4100
Charge Voltage Level F	101	4150
Charge Voltage Level G	110	4200
Charge Voltage Level H	111	4250

Fonte: (Texas Instruments, 2016)

Como apenas três bits podem ser escolhidos a combinação máxima é todos os bits unitários. A ordem do bit mais significativo para o menos segue o seguinte: *ALERT1*, *ALERT2*, VEN, LEN. Assim, se por exemplo os bits VEN e LEN são selecionados para este controle discreto, o bit VEN será o mais significativo, o LEN será o do meio e o menos significativo será sempre zero (pois apenas dois pinos foram escolhidos). Assim, os únicos níveis de tensão possíveis seriam 3900mV, 4000mV, 4100mV ou 4200mV. Quando não há oito níveis permitidos, é necessário repetir o último nível várias vezes até completar os oito níveis. Conforme mostrado na Tabela 3.4.

Tabela 3.4 – Níveis de tensão

Nível	Tensão (mV)
Charge Voltage Level A	3900
Charge Voltage Level B	4000
Charge Voltage Level C	4100
Charge Voltage Level D	4200
Charge Voltage Level E	4200
Charge Voltage Level F	4200
Charge Voltage Level G	4200
Charge Voltage Level H	4200

Fonte: (Texas Instruments, 2016)

Esses níveis de tensão podem ser alterados através dos endereços $0x4197$ (*Charge Voltage Level A*) a $0x41A5$ (*Charge Voltage Level H*) na memória *flash*.

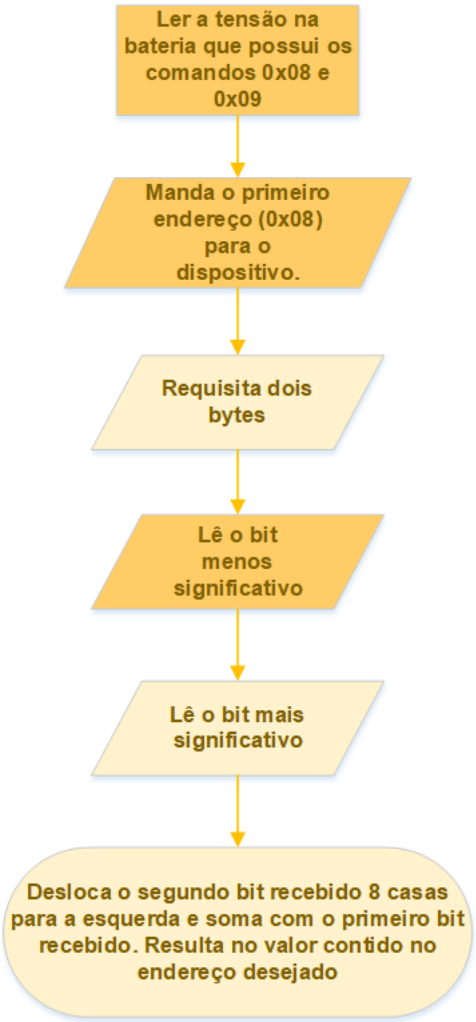
3.4.12 Executando os Comandos

Como a comunicação I^2C só faz a transmissão de um *byte* por vez, as leituras e algumas funções são executadas através de comandos. Existem dois tipos de comandos, os que são apenas para leitura das características das células e outros que são para executar alguma função.

Os de leitura são apenas endereços com comprimento de um *byte* de onde são requisitados os dados para leitura.

O processo de acesso a esses comandos é mais simples do que a leitura dos registradores. É apenas enviado o primeiro endereço dos comandos e requisitado os dados de dois *bytes*. Os dois endereços são sempre sequenciais, então enviando apenas o primeiro, o ponteiro da memória será colocado sobre este e quando requisitado dois *bytes* será enviado o dado sobre o qual estará apontando e o próximo. No fluxograma da Figura 3.21 pode ser visto a sequência das ações.

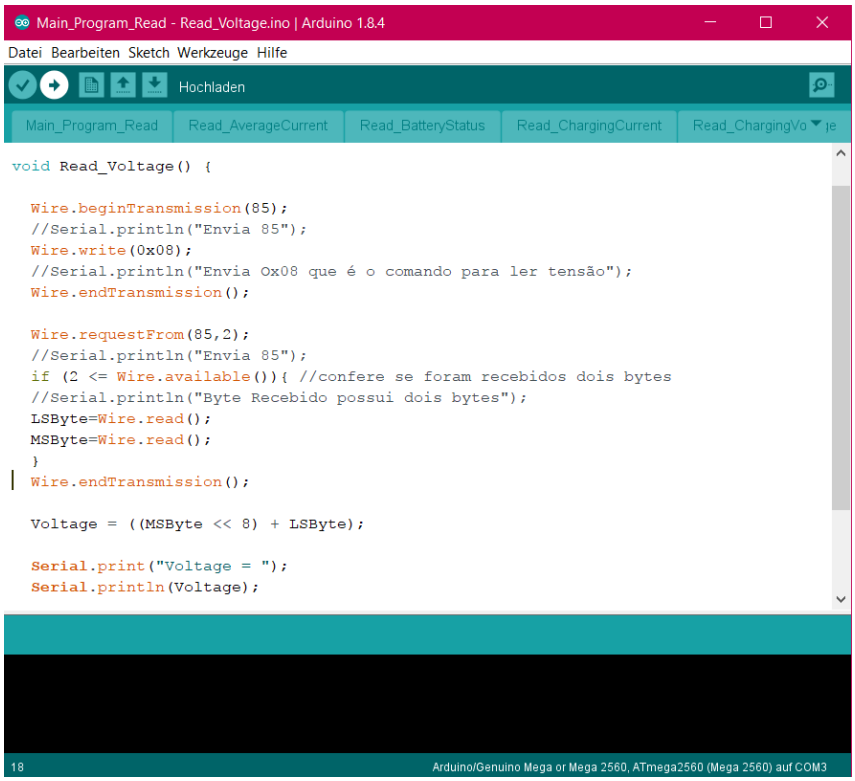
Figura 3.21 – Exemplo de Leitura da Tensão da Bateria



Fonte: Próprio autor.

Um exemplo do programa correspondente ao fluxograma acima é o apresentado na Figura 3.22.

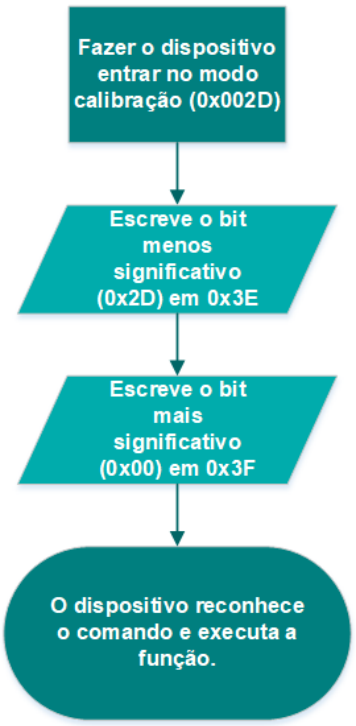
Figura 3.22 – Programa para Leitura da Tensão da Bateria



Fonte: Próprio autor.

O acesso aos comandos que executam funções é semelhante à leitura de registradores na memória *flash*. Visto que estes comandos são compostos por um endereço de dois *bytes*. Sendo assim, deve ser feito através do *Manufacturer Access Control*. O *byte* menos significativo deve ser escrito no endereço $0x3E$ e o mais significativo em $0x3F$. Então o dispositivo lê esses comandos e executa a função associada aos mesmos. O fluxograma da Figura 3.23 exemplifica este processo.

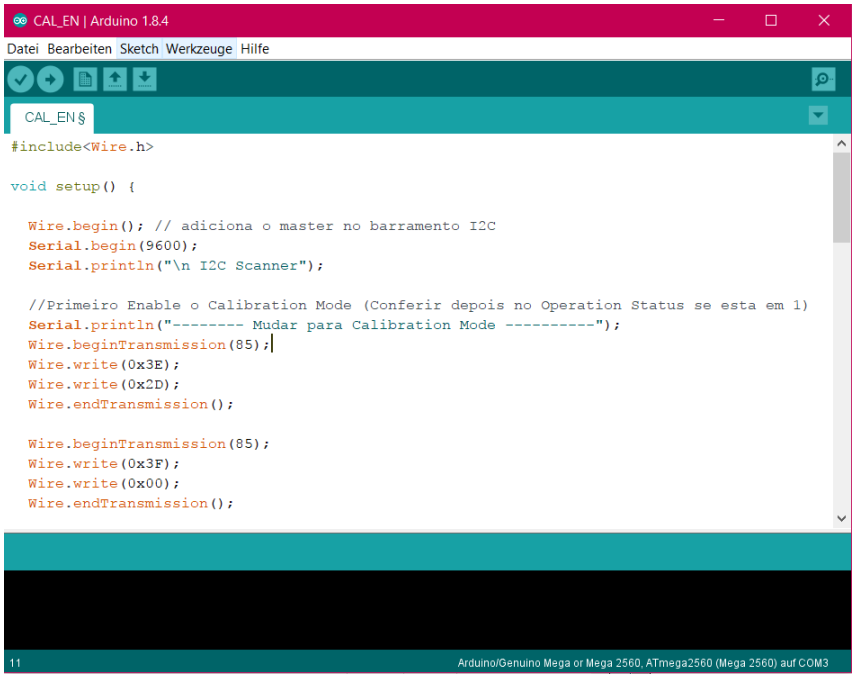
Figura 3.23 – Executar funções através de comandos



Fonte: Próprio autor.

O programa para este mesmo exemplo do fluxograma pode ser visto na Figura 3.24.

Figura 3.24 – Programa para executar funções através de comandos



Fonte: Próprio autor.

Os principais comandos estão apresentados na Tabela 3.5.

Tabela 3.5 – Principais Comandos e seus Endereços

Comando	**Endereço**	**Descrição**
Voltage	0x08 e 0x09	Tensão medida nas células
Temperature	0x06 e 0x07	Temperatura nas células
Current	0x0C e 0x0D	Corrente fluindo no resistor *shunt*
Remaining Capacity	0x12 e 0x13	Capacidade remanescente na bateria
Average Current	0x14 e 0x15	Tensão média nos terminais da bateria
Time To Empty	0x16 e 0x17	Prevê o tempo de duração da capacidade da bateria baseado na taxa de descarga
Time To Full	0x18 e 0x19	Prevê o tempo restante para completar a carga da bateria
Cycle Count	0x2A e 0x2B	Quantos ciclos a bateria experienciou
Relative State of Charge	0x2C e 0x2D	Prevê a capacidade remanescente como uma percentagem da capacidade total
State of Health	0x2E e 0x2F	Retorna o estado de saúde da bateria
Operation Status	0x3A e 0x3B	Retorna diversas características de operação
Battery Status	0x0A e 0x0B	Retorna diversas indicações do estado da bateria
Gauging Status	0x0056	Retorna diversas indicações do algoritmo de monitoramento

Fonte: Adaptado de (Texas Instruments, 2016)

A lista completa do endereçamento dos comandos pode ser encontrada em (Texas Instruments, 2016).

O comando *Battery Status* retorna uma série de indicações sobre o *status* da bateria. A descrição de seus bits está detalhada na Tabela 3.6.

Tabela 3.6 – Bits do registrador de indicações de status da Bateria

Bit	**Nome**	**Descrição**
14	SOCLOW	SOC baixo detectado
13	UTC	Temperatura baixa durante carga
12	UTD	Temperatura baixa durante descarga
11	OTC	Temperatura alta durante carga
10	OTD	Temperatura alta durante descarga
9	BATHIGH	Tensão caiu abaixo do nível definido em (0x4189)
8	BATLOW	Tensão caiu abaixo do nível definido em (0x4184)
7	SLEEP	O dispositivo está no modo SLEEP
6	CHGINH	O processo de carga está bloqueado
5	FD	Descarga completa detectada
4	FC	Carga completa detectada
3	TCA	Alerta de término do processo de carga
2	TDA	Alerta de término do processo de descarga
1	CHG	Corrente de carga detectada
0	DSG	Corrente de descarga detectada

Fonte: Adaptado de (Texas Instruments, 2016)

O comando *Operation Status* retorna uma série de indicações sobre o *status* das operações. A descrição de seus bits está detalhada na Tabela 3.7.

Tabela 3.7 – Bits do registrador de indicações de status das operações

Bit	Nome	Descrição
8	AUTH	Autenticação em progresso
7	BLT	Este bit é setado conforme descrito em 3.4.8
6	SMTH	Função *smooth* ativa
5	ACTHR	A capacidade de carga acumulada ultrapassou o nível configurado em $0x416C$
4	VDQ	Ciclo de descarga detectado
3	EDV2	Tensão da célula está abaixo de EDV2
2	SEC1	Informa status de segurança
1	SEC0	Informa status de segurança
0	CALMD	Modo de calibração ativo

Fonte: Adaptado de (Texas Instruments, 2016)

O comando *Gauging Status* retorna uma série de indicações sobre o algoritmo de monitoramento. A descrição de seus bits está detalhada na Tabela 3.8.

Tabela 3.8 – Bits do registrador de indicações de status do algoritmo de monitoramento

Bit	Nome	Descrição
15	VDQ	Descarga detectada
14	EDV2	Nível EDV2 foi atingido durante descarga
13	EDV1	Nível EDV1 foi atingido durante descarga
10	FCCX	Registrador FCC foi atualizado
8	REST	Dispositivo está no modo relaxado
7	CF	Erro no monitoramento
6	DSG	Processo de carga não foi detectado
5	EDV0	Nível EDV0 foi atingido durante descarga
3	TC	Fim do processo de carga detectado
2	TD	Fim do processo de descarga detectado
1	FC	Carga completa detectada
0	FD	Descarga completa detectada

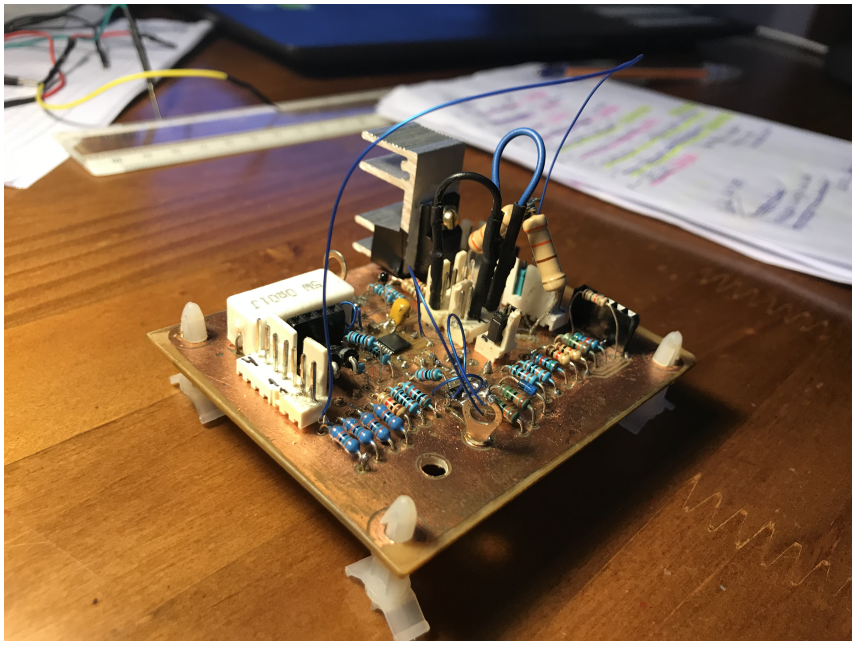
Fonte: Adaptado de (Texas Instruments, 2016)

3.5 RESULTADOS E CONFIGURAÇÕES DE LABORATÓRIO

3.5.1 O protótipo

Para a realização de testes em laboratório, foi confeccionada uma placa de tamanho 8.5cm x 6.5cm, a qual pode ser vista na Figura 3.25. Os *layers* da placa estão mostrados no Anexo A.

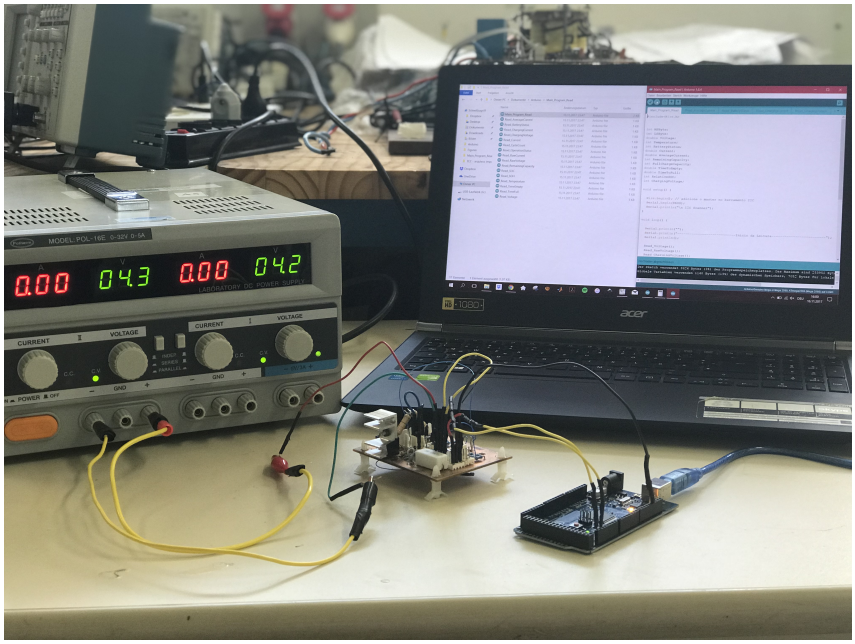
Figura 3.25 – Placa desenvolvida em Laboratório



Fonte: Próprio autor.

As células foram simuladas por uma fonte de tensão, e os processos de carga e descarga foram representados pelo aumento e diminuição da tensão da fonte. O circuito implementado em laboratório pode ser visto na Figura 3.26.

Figura 3.26 – Circuito implementado em Laboratório



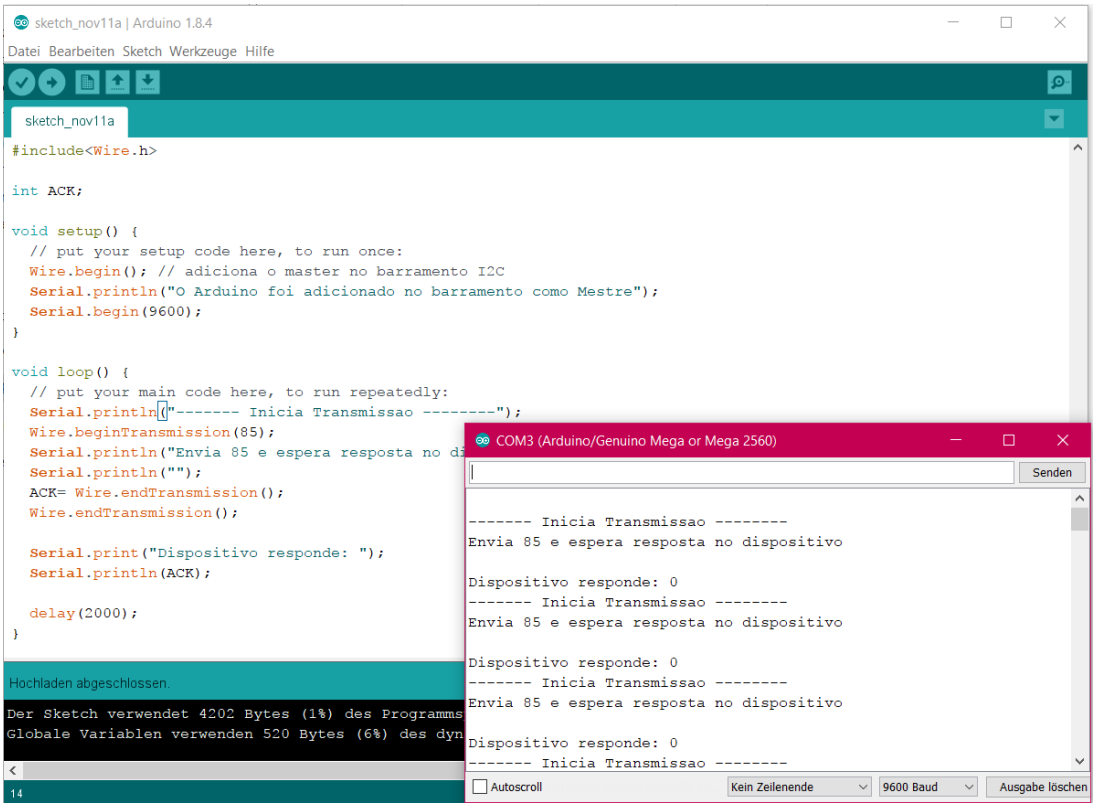
Fonte: Próprio autor.

A comunicação I^2C entre o computador e o dispositivo foi realizada através do microcontrolador arduíno, onde todos os códigos foram implementados em linguagem C.

3.5.2 Resultados Alcançados

Para estabelecer a comunicação foi necessário fazer uma adaptação à placa. Os zeners conectados ao barramento de dados e de *clock* foram retirados, visto que a comunicação estava programada para uma frequência de 100kHz[13] e os zeners eram muito lentos e acabavam cortando o sinal. Após retirados, a comunicação foi estabelecida. Na Figura 3.27 é mostrado ao fundo o código executado e na frente o display de acesso do arduíno com a resposta do dispositivo.

Figura 3.27 – Comunicação estabelecida e Resposta do dispositivo



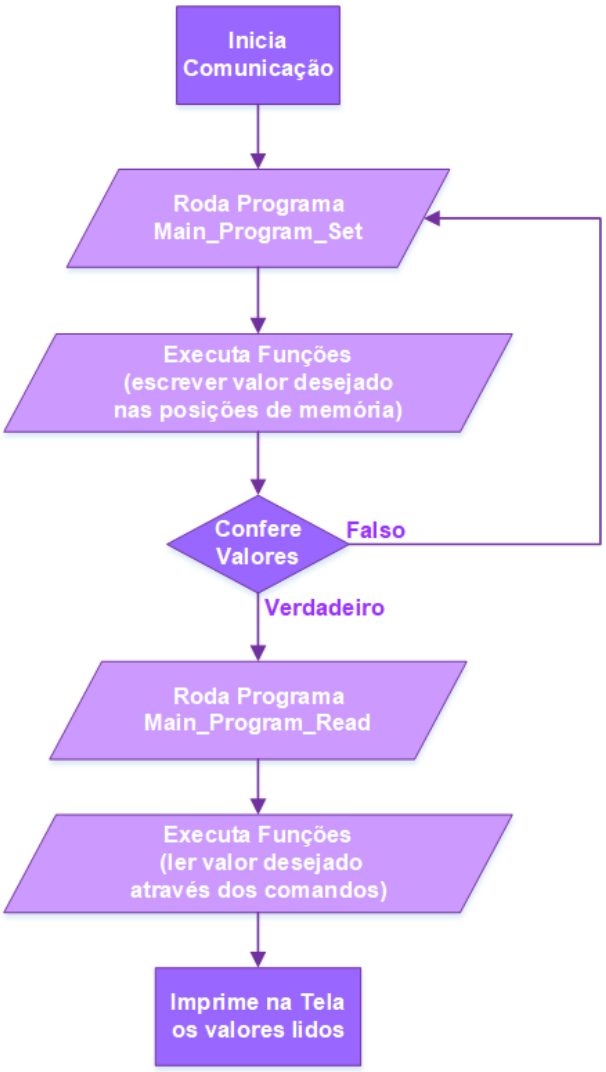
Fonte: Próprio autor.

O dispositivo respondeu zero, o que conforme o fluxograma apresentado na Figura 3.8 significa "dado transmitido com sucesso".

Ao conectar a placa ao arduíno, é preciso rodar o programa chamado *Main_Program_Set*. No qual são chamadas diversas funções para realizar a configuração dos registradores e posições da memória. Realizada esta configuração com sucesso, então roda-se o programa chamado *Main_Program_Read*. Este programa também chama funções, porém desta vez para realizar a leitura dos comandos desejados. Ambos os programas podem ser encontrados no Anexo C. Na Figura 3.28 encontra-se um fluxograma que detalha este processo.

[13] O dispositivo pode chegar até 400kHz.

Figura 3.28 – Passos a executar os programas gerais de configuração e de leitura.

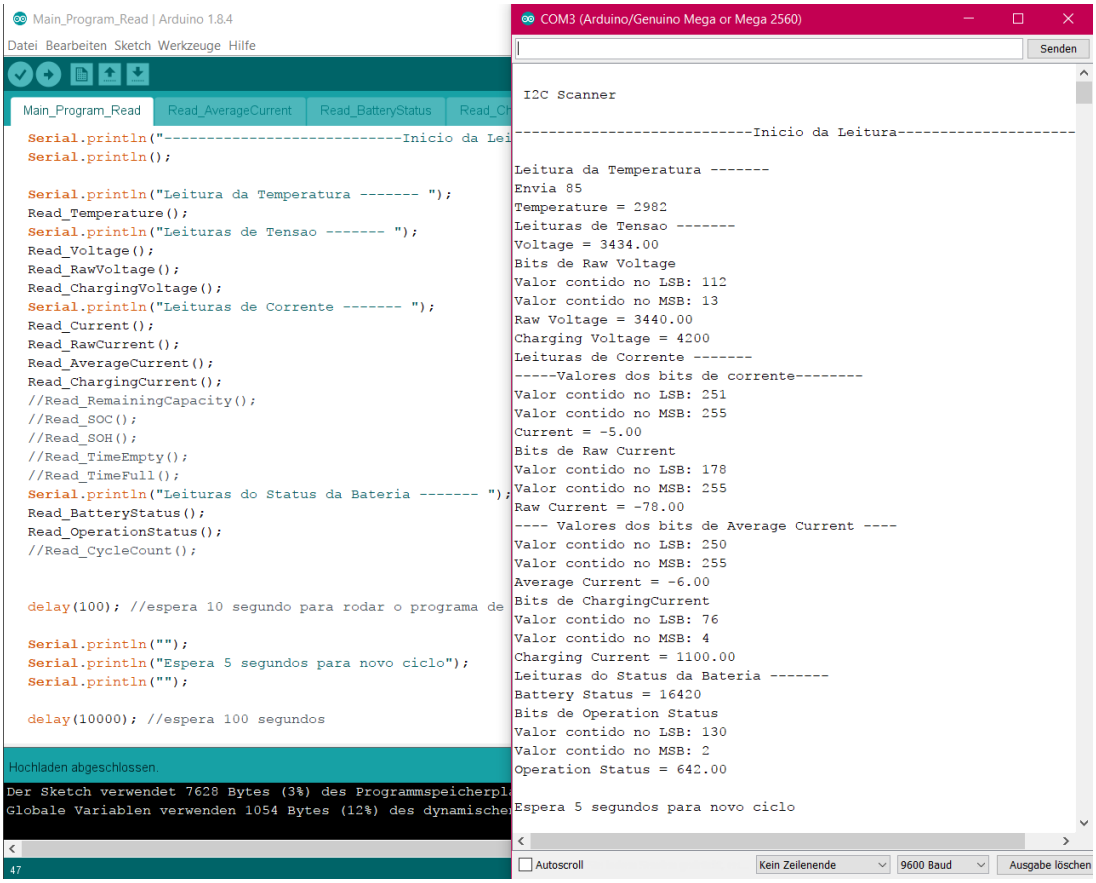


Fonte: Próprio autor.

No registrador *Op Config A* foi escrito o valor 0x00A4, o qual habilita a medição de temperatura pelo termistor interno ao dispositivo, usa o pino VSS como referência, habilita a definição da tensão de carga (*ChargingVoltage*) e da corrente de carga (ChargingCurrent) através do método JEITA e o reconhecimento do término do processo de carga é feito pelo algoritmo NiXX.

Então foi aplicado 3.4V no pino BAT, simulando apenas uma célula que possua uma tensão abaixo de 5V. Na Figura 3.29 pode ser visto os resultados obtidos pela leitura.

Figura 3.29 – Resultados obtidos através da Leitura do dispositivo



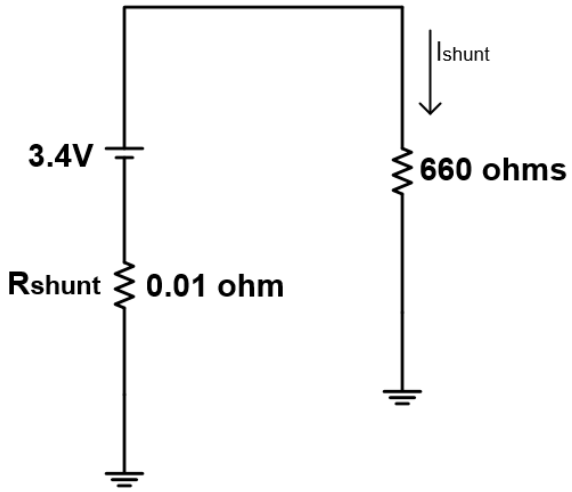
Fonte: Próprio autor.

O valor de temperatura retornado foi 2982, porém sua unidade é 0.1K, o que gera 298.2K. Este valor deve ser diminuído de 273 quando transformado para graus *Celsius*, resultando em 25.2°C.

Pelo método JEITA, foi configurado que quando a temperatura medida estivesse no intervalo $10°C < temp < 45°C$ a tensão de carga seria 4200mV. Exatamente o mostrado no display, comprovando que este processo está fazendo a análise correta. A tensão medida foi 3434mV, atingindo também o esperado visto que a tensão aplicada foi de 3.4V.

A corrente lida retorna um valor negativo, o qual indicaria uma possível descarga, porém a corrente mínima para detectar uma descarga foi configurada em $0x4162$ *Discharge Detection Threshold* como 60mA e o valor lido está abaixo disso. O valor de 5mA confere com a teoria, o circuito equivalente seria o mostrado na Figura 3.30.

Figura 3.30 – Circuito Equivalente para Calcular Corrente



Fonte: Próprio autor.

Desprezando o resistor *shunt*, que tem uma resistência muito baixa, a corrente é calculada na Equação 3.5.

$$I_{shunt} = \frac{3.4}{660} = 0.00515A = 5.15mA \tag{3.5}$$

O valor retornado em *RawCurrent* vem do periférico *Coulomb Counter*, e está indiretamente relacionado à corrente medida. O valor determinado em *ChargingCurrent* também está dentro do esperado, pois com uma temperatura de 25.2°C o valor da corrente de carga configurado em $0x4110$ é 1100mA.

O valor lido no *Battery Status* é 16420, e indica que o estado de carga está baixo, ou seja, a *flag* SOCLOW está setada pois o estado de carga foi reconhecido como abaixo de 10% (esta percentagem é definida em $0x418E$). Como a corrente fluindo é mínima, o dispositivo reconheceu como se tivesse ocorrido uma descarga completa e então indicou através das *flags* *Full Discharge Detected* e *Terminate Charge Alarm* setadas.

O valor lido no *Operation Status* é 642 e indica que o dispositivo está no modo de segurança "full access", e a *flag* BLT está setada. Isso ocorreu porque a corrente é menor que zero e a capacidade remanescente medida é menor que 150mAh. Este resultado condiz pois a carga máxima (FCC e DC) está configurada para 300mAh, o que faria 150mAh ter uma capacidade remanescente de 50%. Como em função dos resultados lidos no *Battery Status* é sabido que o estado de carga está abaixo de 10% a condição é satisfeita para o bit BLT ser setado.

Até o presente momento, somente foi validado o teste e medições para a configuração abaixo de 5V pois esta não requer calibração. Para uma tensão acima deste valor é necessário

uma série de configurações adicionais, as quais ainda não foram devidamente ajustadas. Além do processo de calibração das variáveis (tensão, corrente, temperatura, offset da placa, entre outras) que ainda não foi realizado com sucesso.

3.6 CONCLUSÃO DO CAPÍTULO

Este trabalho teve como objetivo o estudo e implementação de um sistema de monitoramento e controle de baterias de chumbo-ácido. Para este, foi escolhido o circuito integrado BQ34110 da *Texas Instruments*, suas funcionalidades estudadas e um projeto implementado. Portanto, foi confeccionada uma placa com o circuito auxiliar sugerido pelo fabricante e testes realizados. A comunicação foi feita através do protocolo I^2C com o microcontrolador arduíno. Foi elaborada a programação do circuito integrado na linguagem C, para seu adequado funcionamento. Foram encontradas diversas dificuldades durante sua execução como por exemplo os diodos utilizados para proteção do barramento de comunicação eram muito lentos e acabavam danificando a mesma. Alguns *mosfets* trabalham na região linear, e isto foi percebido apenas perto da data limite do trabalho, portanto suas substituições não foram felizes e dois deles não executaram suas funções corretamente, sendo necessário utilizar *jumpers* para garantir o correto funcionamento do circuito. Entretanto a maior dificuldade foi o entendimento do funcionamento do circuito integrado bem como realizar sua configuração. O manual de referência do fabricante é direcionado para a utilização de um *software* também desenvolvido por eles e então os comandos desta ferramenta tiveram que ser "desenvolvidos" passo a passo no algoritmo implementado neste trabalho. A implementação da rotina de calibração foi realizada e revisada porém não foi implementada com sucesso. A sugestão dada pelo fabricante seria implementar o código feito na placa original (da *Texas Instruments*), validar este código verificando o corrente funcionamento e medições dos parâmetros, para posteriormente implementar na placa confeccionada em laboratório. Por fim, é comprovado que não é preciso um circuito muito complexo, nem a utilização de uma quantidade muito elevada de componentes para a elaboração deste sistema de controle. A complexidade encontra-se na devida programação do dispositivo, sendo necessária a análise e configuração de inúmeros fatores para que as estimações sejam corretamente realizadas.

4 CONSIDERAÇÕES FINAIS

O objetivo principal das ideias contidas neste trabalho era destacar a importância de monitorar e controlar os eventos de carga e descarga de baterias, bem como a alimentação das cargas a elas conectadas. As vantagens geradas pelo uso adequado das baterias são inúmeras, sendo a principal delas o prolongamento de sua vida útil.

No capítulo 2 foram analisados aspectos construtivos de vários tipos de baterias. Foi então escolhida a bateria de chumbo-ácido por ainda ser a mais difundida no Brasil, e apesar de seu volume e peso elevados, ter o menor custo. Então, foram apresentados métodos de carga com suas vantagens e desvantagens, porém todos com um objetivo em comum: realizar o processo de carga sem gerar sobretensões, evitando temperaturas elevadas e picos de corrente. A seguir, foram introduzidas técnicas para a modelagem da bateria de chumbo-ácido, sendo o modelo matemático o mais usual. Logo após, é descrita a razão da necessidade de uma precisa estimação do estado de carga das baterias. Além de incontáveis vantagens para o usuário, o qual pode programar melhor a utilização do equipamento, os fabricantes de baterias atingem uma maior sofisticação do controle não sendo mais preciso sobredimensionar o projeto para garantir o funcionamento nas condições desejadas. Ao longo dos anos, muitos métodos para medição do SOC foram desenvolvidos, porém os estimadores recursivos, como o Filtro de Kalman, se destacaram no uso de sistemas onde há não-linearidades.

No capítulo 3 foram apresentados os circuitos integrados que efetuam a teoria proposta anteriormente, o monitoramento e controle dos processos de baterias. O dispositivo escolhido foi o BQ34110, o qual pode ser aplicado para cinco tipos de bateria, porém este trabalho foi direcionado ao chumbo-ácido. Foi então exposto o circuito da placa a ser implementada. Como o circuito integrado utiliza a comunicação I^2C para a transferência de dados, é explicado brevemente como ela funciona e como foi aplicada neste trabalho. Por último, o dispositivo tem suas funções detalhadas e todas as configurações necessárias para sua utilização comentadas.

Assim sendo, o objetivo deste trabalho era comprovar a importância do controle da bateria, estimando seu estado de carga e estado de saúde e mostrar a implementação de um sistema de monitoramento. No desenvolvimento teórico os motivos estão muito bem retratados, entretanto na prática seriam necessários mais testes para concluir com sucesso a realização deste sistema. Acredita-se que alguns resultados não foram obtidos em função da rotina de calibração não conseguir sem implementada corretamente. Uma alternativa sugerida pelo fabricante seria a aquisição da placa da *Texas Instruments* para validar o código de programação, e posteriormente transferí-lo para a placa confeccionada e validar a mesma.

4.1 MELHORIAS NESTE TRABALHO

Nesta secção encontram-se algumas possibilidades de melhorias a realizar neste trabalho para um melhor desempenho.

A primeira sugestão seria refazer o *layout* da placa, colocando os conectores que devem ser ligados por *jumpers* mais próximos um do outro. Bem como tirar os conectores que se encontram no meio da placa passando-os para as extremidades. Tirar a malha de terra do *layer* superior da placa, deixando apenas no *layer* inferior por questões estéticas.

Por fim, seria interessante validar o funcionamento do "conversor dc-dc"(composto pelo conjunto mosfet (Q1), zenner (D1) e resistor (R1)) conectado à entrada do regulador de tensão interno ao dispositivo também não foi possível. Portanto, como sugestão fica fazer testes e validar essa teoria.

4.2 TRABALHOS FUTUROS

Nesta secção são deixadas sugestões para dar continuidade ao projeto iniciado neste trabalho.

Por questão de tempo, foi realizado apenas o monitoramento das características da bateria. Portanto, sugere-se realizar o controle da alimentação da carga conectada ao sistema. Utilizando os registradores de controle e os pinos de proteção, os quais são responsáveis por permitir ou bloquear a corrente que flui para a carga.

Outra possibilidade que este trabalho deixa para trabalhos futuros seria o estudo e utilização da função *End-of-Service* contida no BQ34110. Esta é responsável pela estimação do tempo de vida útil da bateria baseado em diversos fatores e é configurada por uma classe de registradores com inúmeras funções.

REFERÊNCIAS BIBLIOGRÁFICAS

AGUIRRE, L. **Introdução à Identificação de Sistemas - Técnicas Lineares e Nào-Lineares Aplicadas a Sistemas Reais**. 3. ed. Belo Horizonte, Brasil: Editora UFMG, 2007.

ARAUJO, J. F.; HARTMANN, L. V.; CORREA, M.; LIMA, A. Lead-acid battery modeling and state of charge monitoring. **Applied Power Electronics Conference and Exposition (APEC)**, 2010. 2010.

BARROS, B. **Algoritmo aprimorado de rastreamento de máxima potência para sistemas fotovoltaicos.** Dissertação (TCC) — Universidade do Estado de Santa Catarina, Joinville, Brazil, 2017.

BASTOS, R. F. **Sistema de Gerenciamento para Carga e Descarga de Baterias (Chumbo-Ácido) e para Busca do Ponto de Máxima Potência Gerada em Painéis Fotovoltaicos Empregados em Sistemas de Geração Distribuída**. Dissertação (Mestrado) — Universidade de São Paulo, São Carlos, Brasil, 2013.

CERAOLO, M. Dynamical models of lead-acid batteries. **IEEE Transactions on Power Electronics**, 2000. v. 15, n. 4, 2000.

CHAGAS, M. **Novas Tecnologias para Avaliação de Baterias**. Dissertação (Mestrado) — IEP/LACTEC, Curitiba, Brazil, 2007.

COELHO, K. F. **Estudo de uma Fonte Ininterrupta de Corrente Contínua de Baixa Potência gerenciada por um Microcontrolador**. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, Florianópolis, Brazil, 2001.

DEKKA, A.; GHAFFARI, R.; VENKATESH, B.; WU, B. A survey on energy storage technologies in power systems. **Electrical Power and Energy Conference**, 2015. 2015.

DRAGICEVIC, T.; SUCIC, S.; GUERRERO, J. Battery state-of-charge and parameter estimation algorithm based on kalman filter. 2013. 2013.

FAKHAM, H.; LU, D.; FRANCOIS, B. Power control design of battery charger in a hybrid active pv generator for load-following applications. **IEEE Transactions on Industrial Electronics**, 2011. v. 58, n. 1, Janeiro 2011.

FRAGA, J. R. C. P. **Análise do Comportamento da Bateria Utilizada em Sistemas Fotovoltaicos de Pequeno Porte**. Tese (Doutorado) — Universidade Estadual Paulista Júlio de Mesquita Filho, Botucatu, Brazil, 2009.

GUO, S. **The Application of Genetic Algorithms to Parameter Estimation in Lead-Acid Battery Equivalent Circuit Models**. Dissertação (Mestrado) — University of Birmingham, Birmingham, England, 2010.

JOSEPH, A.; M.SHAHIDEHPOUR. Battery storage systems in electric power systems. **Power Engineering Society General Meeting**, 2006. 2006.

MOURA. **Moura Manual Técnico Acumuladores**. [S.l.], 2011.

PILLER, S.; PERRIN, M.; JOSSEN, A. Methods for state-of-charge determination and their applications. 2001. 2001.

POP, V.; BERGVELD, H.; DANILOV, P. R. D.; NOTTEN, P. **Battery Management Systems - Accurate State-of-Charge Indication for Battery-Powered Applications**. 9. ed. Belo Horizonte, Brasil: Springer, 2008.

REIS, V. R. dos. **I2C – Protocolo de Comunicação**. 2017. November 11, 2017. Disponível em: <{http://www.arduinobr.com/arduino/i2c-protocolo-de-comunicacao/}>.

REMES, C.; OLIVEIRA, S. Modelagem de baterias e estimação de indicadores soc e soh. **Seminário Estudos Dirigidos**, 2014. 2014.

Texas Instruments. **BQ34Z100-g1 - Application Report**. 2015. Available in: <http://www.ti.com/lit/an/slua760/slua760.pdf>.

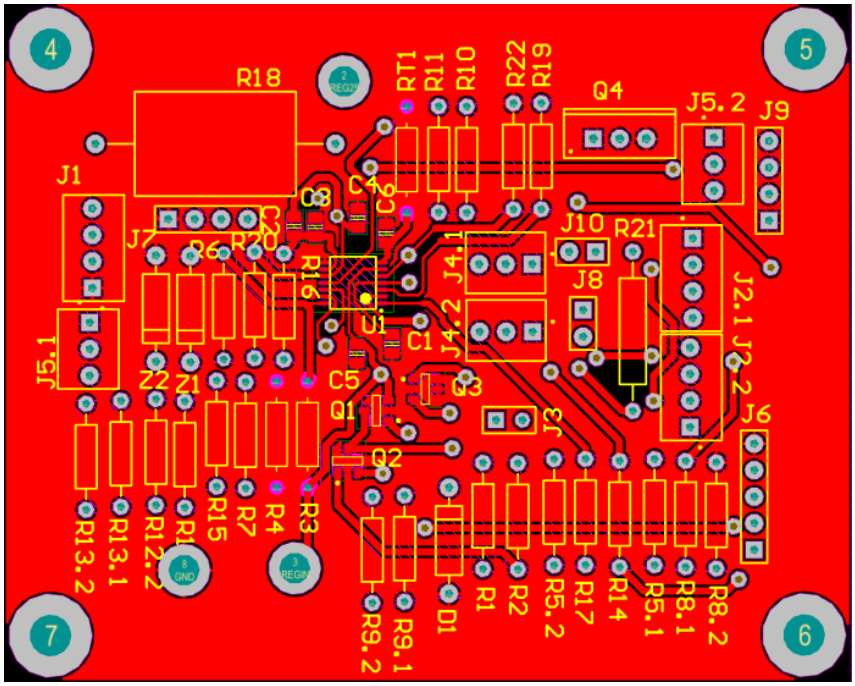
Texas Instruments. **BQ34110 - Technical Reference Manual**. 2016. Available in: <http://www.ti.com/lit/ug/sluubf7/sluubf7.pdf>.

UNKNOWN. **I2C – Protocolo de Comunicação**. 2017. November 11, 2017. Disponível em: <{http://microcontrolandos.blogspot.com.br/2012/12/comunicacao-i2c.html}>.

ANEXO A – Layers da Placa

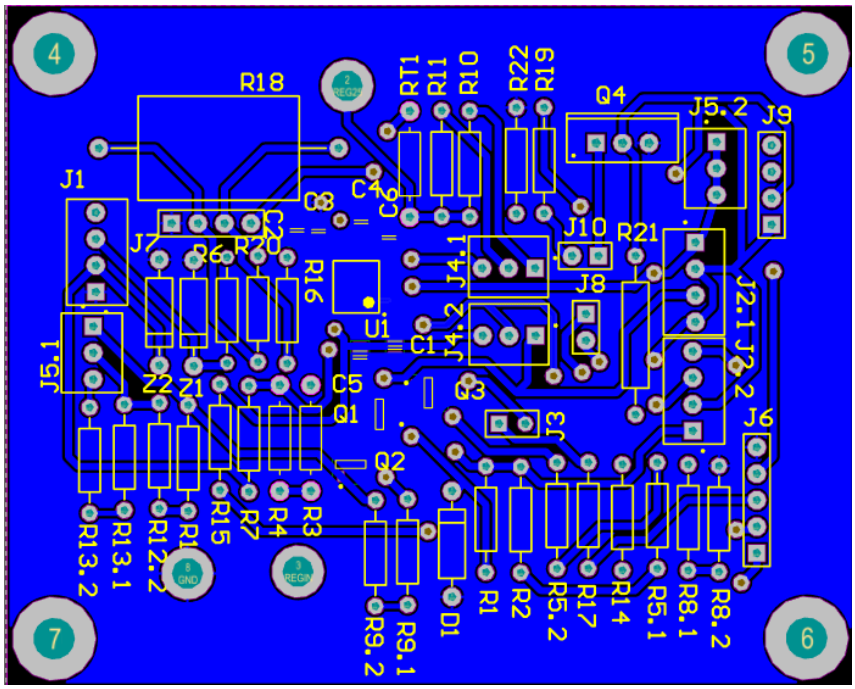
O esquemático e o projeto pcb da placa foram desenvolvidos no Altium. Neste Anexo encontram-se o *bottom layer* e o *top layer*.

Figura A.1 – Top Layer



Fonte: Próprio autor.

Figura A.2 – Bottom Layer



Fonte: Próprio autor.

ANEXO B – Lista de Componentes

Neste anexo, encontra-se a lista de componentes utilizados para a implementação do projeto apresentado.

Tabela B.1 – Níveis de tensão

Descrição do Componente	**Quantidade**	**Nome no Esquemático**
Capacitor cerâmico 3300pF smd0805	1	C1
Capacitor cerâmico 0.1uF smd0805	4	C2, C3, C4 e C5
Capacitor multicamada 1uF 50V	1	C6
Diodo Zenner 5.6V 1/4W	1	D1
Diodo Zener 5.6V 5W	2	Z1 e Z2
Mosfet 60V 0.17 (2n7002) smd0805	1	Q1
Mosfet 240V 0.1A (BSS131) smd0805	1	Q3
Mosfet 45V 0.5A (BC807) smd0805	1	Q2
Mosfet 60V 15A (2SK1419	1	Q4
Resistor 10k 5% 0.1W	3	R1, R3 e R4
Resistor 100k 1% 0.1W	3	R2, R10 e R11
Resistor 150k 1% 0.1W	7	R9.1, R9.2, R12.1, R12.2, R13.1, R13.2 e R5.1
Resistor 15k 1% 0.1W	2	R5.2 e R8.1
Resistor 100 ohms 1% 0.1W	4	R16, R20, R6 e R7
Resistor 1.5k 1% 0.1W	1	R8.2
Resistor 1k 5% 0.1W	4	R14, R15, R17 e R19
Resistor 330 ohms 5% 3W	2	Colocado mais tarde no lugar de R21
Resistor 1M 5% 0.1W	1	R22
Termistor NTC 10k	1	RT1
Conectores	13	J1, J2.1, J2.2, J3, J4.1, J4.2, J5.1, J5.2, J6, J7, J8, J9 e J10

Fonte: (Texas Instruments, 2016)

ANEXO C – Códigos Desenvolvidos para a Implementação

Neste anexo, encontram-se que os códigos utilizados para realizar a implementação do circuito apresentado neste trabalho. Os códigos estão na linguagem C e foram implementados através do arduíno.

C.1 OBTENÇÃO DOS DADOS PARA CALIBRAÇÃO

Nesta seção, apresenta-se a rotina desenvolvida.

```
#include<Wire.h>

int rawDataSum = 0;
int samplesToAvg = 50;
int counterNow;
int counterPrev;


void setup() {
  Wire.begin(); // adiciona o master no barramento I2C
  Serial.begin(9600);
  Serial.println("\n I2C Scanner");

  //Primeiro Enable o Calibration Mode (Conferir depois no
  Operation Status se esta em 1)
  Serial.println("-------- Mudar para Calibration Mode ----------");
  Wire.beginTransmission(85);
  Wire.write(0x3E);
  Wire.write(0x2D);
  Wire.endTransmission();

  Wire.beginTransmission(85);
  Wire.write(0x3F);
  Wire.write(0x00);
  Wire.endTransmission();
}

void loop() {

  for (int loopCount = 0; loopCount <= samplesToAvg; loopCount++){
  Wire.beginTransmission(85);
  Wire.write(0x79);
  Wire.endTransmission();

  Wire.requestFrom(85,1);
  counterNow = Wire.read();
  counterPrev = counterNow;
  Wire.endTransmission();
```

```
  delay(200);

  Wire.beginTransmission(85);
  Wire.write(0x79);
  Wire.endTransmission();

  Wire.requestFrom(85,1);
  counterNow = Wire.read();
  Wire.endTransmission();

  if(counterNow == counterPrev)
  {
    delay (200);

    Wire.beginTransmission(85);
    Wire.write(0x79);
    Wire.endTransmission();

    Wire.requestFrom(85,1);
    counterNow = Wire.read();
    Wire.endTransmission();
  }
  else
  {
    Wire.write(0x7E); //7A pra corrente e 7C pra tensao

    Wire.requestFrom(85,2);
    int LSB = Wire.read();
    int MSB = Wire.read();
    Wire.endTransmission();

    rawDataSum = ((MSB<<8) + LSB);
    counterPrev = counterNow;
  }
  }
  int avgRawData = rawDataSum/samplesToAvg;
  }
```

C.2 OPERAÇÃO DE LEITURA

Nesta secção, apresenta-se o código para realizar as leituras da bateria. Foram criadas funções para cada leitura e todas elas são chamadas no programa principal.

```
#include<Wire.h>
```

```
int MSByte;
int LSByte;
double Voltage;
int Temperature;
int BatteryStatus;
double Current;
double AverageCurrent;
int RemainingCapacity;
int FullChargeCapacity;
double TimeToEmpty;
double TimeToFull;
int RelativeSOC;
int ChargingVoltage;

void setup() {

  Wire.begin(); // adiciona o master no barramento I2C
  Serial.begin(9600);
  Serial.println("\n I2C Scanner");
}

void loop() {

  Serial.println("");
  Serial.println("------------------Inicio da Leitura------------");
  Serial.println();

  Read_Voltage();
  Read_RawVoltage();
  Read_ChargingVoltage();
  Read_Current();
  Read_RawCurrent();
  Read_AverageCurrent();
  Read_ChargingCurrent();
  Read_RemainingCapacity();
  Read_SOC();
  Read_SOH();
  Read_TimeEmpty();
  Read_TimeFull();
  Read_BatteryStatus();
  Read_OperationStatus();
  Read_CycleCount();
  Read_Temperature();

  delay(100); //espera 10 segundo para rodar o programa de novo

  Serial.println("");
  Serial.println("Espera 5 segundos para novo ciclo");
```

```
  Serial.println("");

  delay(10000); //espera 100 segundos

}
```

C.2.1 Função Read_Voltage

```
#include<Wire.h>

void Read_Voltage() {

  Wire.beginTransmission(85);
  //Serial.println("Envia 85");
  Wire.write(0x08);
  //Serial.println("Envia 0x08 que é o comando para ler tensão");
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  //Serial.println("Envia 85");
  if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
  //Serial.println("Byte Recebido possui dois bytes");
  LSByte=Wire.read();
  MSByte=Wire.read();
  }
  Wire.endTransmission();

  Voltage = ((MSByte << 8) + LSByte);

  Serial.print("Voltage = ");
  Serial.println(Voltage);

  float tensao_real = ((Voltage / 1000) * 916.5) / (16.5);
}
```

C.2.2 Função Read_RawVoltage

```
#include<Wire.h>

void Read_RawVoltage() {

  Wire.beginTransmission(85);
  Wire.write(0x7C);
  Wire.endTransmission();

  Wire.requestFrom(85,2);
```

```
  //Serial.println("Envia 85");
  if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
  //Serial.println("Byte Recebido possui dois bytes");
  LSByte=Wire.read();
  MSByte=Wire.read();
  }

  Serial.println("Bits de Raw Voltage");
  Serial.print("Valor contido no LSB: ");
  Serial.println(LSByte);

  Serial.print("Valor contido no MSB: ");
  Serial.println(MSByte);
  Wire.endTransmission();

  double RawVoltage = ((MSByte << 8) + LSByte);

  Serial.print("Raw Voltage = ");
  Serial.println(RawVoltage);
  }
```

C.2.3 Função Read_ChargingVoltage

```
#include<Wire.h>

void Read_ChargingVoltage() {

  Wire.beginTransmission(85);
  Wire.write(0x30);
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  //Serial.println("Envia 85");
  if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
  //Serial.println("Byte Recebido possui dois bytes");
  LSByte=Wire.read();
  MSByte=Wire.read();
  }
  Wire.endTransmission();

  ChargingVoltage = ((MSByte << 8) + LSByte);

  Serial.print("Charging Voltage = ");
  Serial.println(ChargingVoltage);
}
```

C.2.4 Função Read_Current

```
#include<Wire.h>

void Read_Current() {


  Wire.beginTransmission(85);
  Wire.write(0x0C);
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  //Serial.println("Envia 85");
  if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
  //Serial.println("Byte Recebido possui dois bytes");
  LSByte=Wire.read();
  MSByte=Wire.read();
  }

  Serial.println("-----Valores dos bits de corrente--------");
  Serial.print("Valor contido no LSB: ");
  Serial.println(LSByte);

  Serial.print("Valor contido no MSB: ");
  Serial.println(MSByte);
  Wire.endTransmission();

  Current = ((MSByte << 8) + LSByte);

  Serial.print("Current = ");
  Serial.println(Current);
}
```

C.2.5 Função Read_RawCurrent

```
#include<Wire.h>

void Read_RawCurrent() {

  Wire.beginTransmission(85);
  Wire.write(0x7A);
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  //Serial.println("Envia 85");
  if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
  //Serial.println("Byte Recebido possui dois bytes");
```

```
  LSByte=Wire.read();
  MSByte=Wire.read();
  }

  Serial.println("Bits de Raw Current");
  Serial.print("Valor contido no LSB: ");
  Serial.println(LSByte);

  Serial.print("Valor contido no MSB: ");
  Serial.println(MSByte);
  Wire.endTransmission();

  double RawCurrent = ((MSByte << 8) | LSByte);

  Serial.print("Raw Current = ");
  Serial.println(RawCurrent);
}
```

C.2.6 Função Read_AverageCurrent

```
#include<Wire.h>

void Read_AverageCurrent() {

//Serial.println("------ Inicio da Leitura de Average Current -------");
  Wire.beginTransmission(85);
  Wire.write(0x14);
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  //Serial.println("Envia 85");
  if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
  //Serial.println("Byte Recebido possui dois bytes");
  LSByte=Wire.read();
  MSByte=Wire.read();
  }

  Serial.println("---- Valores dos bits de Average Current ----");
  Serial.print("Valor contido no LSB: ");
  Serial.println(LSByte);

  Serial.print("Valor contido no MSB: ");
  Serial.println(MSByte);
  Wire.endTransmission();

  AverageCurrent = ((MSByte << 8) + LSByte);
```

```
  Serial.print("Average Current = ");
  Serial.println(AverageCurrent);
}
```

C.2.7 Função Read_ChargingCurrent

```
#include<Wire.h>

void Read_ChargingCurrent() {


  Wire.beginTransmission(85);
  Wire.write(0x32);
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  //Serial.println("Envia 85");
  if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
  //Serial.println("Byte Recebido possui dois bytes");
  LSByte=Wire.read();
  MSByte=Wire.read();
  }

  Serial.println("Bits de ChargingCurrent");
  Serial.print("Valor contido no LSB: ");
  Serial.println(LSByte);

  Serial.print("Valor contido no MSB: ");
  Serial.println(MSByte);
  Wire.endTransmission();

  double ChargingCurrent = ((MSByte << 8) + LSByte);

  Serial.print("Charging Current = ");
  Serial.println(ChargingCurrent);
}
```

C.2.8 Função Read_RemainingCapacity

```
#include<Wire.h>

void Read_RemainingCapacity() {

  Wire.beginTransmission(85);
  Wire.write(0x10);
  Wire.endTransmission();
```

```
  Wire.requestFrom(85,2);
  //Serial.println("Envia 85");
  if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
  //Serial.println("Byte Recebido possui dois bytes");
  LSByte=Wire.read();
  MSByte=Wire.read();
  }

  Serial.println("---- Bits de Remaining Capacity ----");
  Serial.print("Valor contido no LSB: ");
  Serial.println(LSByte);

  Serial.print("Valor contido no MSB: ");
  Serial.println(MSByte);
  Wire.endTransmission();

  RemainingCapacity = ((MSByte << 8) + LSByte);

  Serial.print("Remaining Capacity = ");
  Serial.println(RemainingCapacity);
}
```

C.2.9 Função Read_SOC

```
#include<Wire.h>

void Read_SOC() {

//Serial.println("------ Inicio da Leitura Relative SOC ------");
  Wire.beginTransmission(85);
  Wire.write(0x2C);
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  //Serial.println("Envia 85");
  if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
  //Serial.println("Byte Recebido possui dois bytes");
  LSByte=Wire.read();
  MSByte=Wire.read();
  }
  Wire.endTransmission();

  RelativeSOC = ((MSByte << 8) + LSByte);

  Serial.print("SOC = ");
  Serial.println(RelativeSOC);
```

```
}
```

C.2.10 Função Read_SOH

```
#include<Wire.h>

void Read_SOH() {

  Wire.beginTransmission(85);
  Wire.write(0x2E);
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  //Serial.println("Envia 85");
  if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
  //Serial.println("Byte Recebido possui dois bytes");
  LSByte=Wire.read();
  MSByte=Wire.read();
  }
  Serial.println("Bits de SOH");
  Serial.print("Valor contido no LSB: ");
  Serial.println(LSByte);

  Serial.print("Valor contido no MSB: ");
  Serial.println(MSByte);
  Wire.endTransmission();

  double SOH = ((MSByte << 8) + LSByte);

  Serial.print("SOH = ");
  Serial.println(SOH);
}
```

C.2.11 Função Read_TimeToEmpty

```
#include<Wire.h>

void Read_TimeEmpty() {

  //Serial.println("----- Inicio da Leitura Time to Empty -----");
  Wire.beginTransmission(85);
  Wire.write(0x16);
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  //Serial.println("Envia 85");
```

```
  if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
  //Serial.println("Byte Recebido possui dois bytes");
  LSByte=Wire.read();
  MSByte=Wire.read();
  }
  Serial.println("Bits de Time to Empty");
  Serial.print("Valor contido no LSB: ");
  Serial.println(LSByte);

  Serial.print("Valor contido no MSB: ");
  Serial.println(MSByte);
  Wire.endTransmission();

  TimeToEmpty = ((MSByte << 8) + LSByte);

  Serial.print("Time to Empty ");
  Serial.println(TimeToEmpty);
}
```

C.2.12 Função Read_TimeToFull

```
#include<Wire.h>

void Read_TimeFull() {

  Wire.beginTransmission(85);
  Wire.write(0x18);
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  //Serial.println("Envia 85");
  if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
  //Serial.println("Byte Recebido possui dois bytes");
  LSByte=Wire.read();
  MSByte=Wire.read();
  }
  Serial.println("---- Bits de Time to Full ----");
  Serial.print("Valor contido no LSB: ");
  Serial.println(LSByte);

  Serial.print("Valor contido no MSB: ");
  Serial.println(MSByte);
  Wire.endTransmission();

  TimeToFull = ((MSByte << 8) + LSByte);

  Serial.print("Time to Full = ");
```

```
  Serial.println(TimeToFull);
}
```

C.2.13 Função Read_BatteryStatus

```
#include<Wire.h>

void Read_BatteryStatus() {

  Wire.beginTransmission(85);
  Wire.write(0x0A);
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  //Serial.println("Envia 85");
  if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
  //Serial.println("Byte Recebido possui dois bytes");
  LSByte=Wire.read();
  MSByte=Wire.read();
  }
  Wire.endTransmission();

  BatteryStatus = ((MSByte << 8) + LSByte);

  Serial.print("Battery Status = ");
  Serial.println(BatteryStatus);

}
```

C.2.14 Função Read_OperatioStatus

```
#include<Wire.h>

void Read_OperationStatus() {

  Wire.beginTransmission(85);
  Wire.write(0x3A);
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  //Serial.println("Envia 85");
  if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
  //Serial.println("Byte Recebido possui dois bytes");
  LSByte=Wire.read();
  MSByte=Wire.read();
  }
```

```
  Serial.println("Bits de Operation Status");
  Serial.print("Valor contido no LSB: ");
  Serial.println(LSByte);

  Serial.print("Valor contido no MSB: ");
  Serial.println(MSByte);
  Wire.endTransmission();

  double OperationStatus = ((MSByte << 8) + LSByte);

  Serial.print("Operation Status = ");
  Serial.println(OperationStatus);
}
```

C.2.15 Função Read_CycleCount

```
#include<Wire.h>

void Read_CycleCount() {


  Wire.beginTransmission(85);
  Wire.write(0x2A);
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  //Serial.println("Envia 85");
  if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
  //Serial.println("Byte Recebido possui dois bytes");
  LSByte=Wire.read();
  MSByte=Wire.read();
  }

  Serial.println("Bits de Cycle Count");
  Serial.print("Valor contido no LSB: ");
  Serial.println(LSByte);

  Serial.print("Valor contido no MSB: ");
  Serial.println(MSByte);
  Wire.endTransmission();

  double CycleCount = ((MSByte << 8) + LSByte);

  Serial.print("Cycle Count = ");
  Serial.println(CycleCount);
}
```

C.2.16 Função Read_Temperature

```
#include<Wire.h>

void Read_Temperature() {

  Wire.beginTransmission(85);
  Wire.write(0x06);
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  Serial.println("Envia 85");
  if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
  //Serial.println("Byte Recebido possui dois bytes");
  LSByte=Wire.read();
  MSByte=Wire.read();
  }
  Wire.endTransmission();

  Temperature = ((MSByte << 8) + LSByte);

  Serial.print("Temperature = ");
  Serial.println(Temperature);
}
```

C.3 CONFIGURAÇÃO DOS REGISTRADORES

Nesta secção, apresenta-se o código para configurar os registradores da bateria. Foram criadas funções para cada leitura e todas elas são chamadas no programa principal.

```
#include<Wire.h>

void setup() {
  Wire.begin(); // adiciona o master no barramento I2C
  Serial.begin(9600);
  Serial.println("\n I2C Scanner");
}

void loop() {
  Set_Op_Config();
  Set_Pin_Control_Config();
  Set_MSI();
  Set_CEDV_Gauging_Config();
  Set_Accumulated_Charge();
  Set_FCC();

  delay(10000); //espera 10 segundo para rodar o programa de novo
```

```
  Serial.println("");
  Serial.println("Espera 5 segundos para novo ciclo");
  Serial.println("");

  delay(1000); //espera 1 segundo
 }
```

C.3.1 Função Set_Op_Config

```
#include<Wire.h>


int MACaccess1_1;
int MACaccess1_2;
int MACdata1_1;
int MACdata1_2;
int MACdatasum1_1;
int MACdatasum1_2;
int MACdatalen1_1;
int MACdatalen1_2;
long Op_Config;


void Set_Op_Config() {
  Serial.println("---------------- Inicio da Transmissao -----------------");

  Wire.beginTransmission(85);
  Serial.println("Envia 85");
  Wire.write(0x3E);
  Serial.println("Aponta para o endereço Manufacturer Access Control 0x3E");
  Wire.write(0x3A);
  Serial.println("Escreve LSB do endereço no Manufacturer Access Control 0x3E");
  Wire.endTransmission();

  Wire.beginTransmission(85);
  Serial.println("Envia 85");
  Wire.write(0x3F);
  Serial.println("Aponta para o endereço Manufacturer Access Control 0x3F");
  Wire.write(0x41);
  Serial.println("Escreve MSB do endereço no Manufacturer Access Control 0x3F");
  Wire.endTransmission();

  Serial.println("");
  Serial.println("--- Termina de Escrever no Manufacturer Access Control ---");
  Serial.println("");
```

```
Serial.println("--- Ler o Manufacturer Access Control para Conferir ---");
Wire.beginTransmission(85);
Wire.write(0x3E);
Serial.println("Aponta para o endereço que quer ler, no caso 0x3E");
Wire.endTransmission();

Wire.requestFrom(85,2);
Serial.println("Envia 85 e requisita 2 bytes");
if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
Serial.println("Byte Recebido possui dois bytes");
MACaccess1_1=Wire.read();
MACaccess1_2=Wire.read();
}
Serial.print("Valor contido no MACaccess1: ");
Serial.println(MACaccess1_1);

Serial.print("Valor contido no MACaccess2: ");
Serial.println(MACaccess1_2);
Wire.endTransmission();

Serial.println("");
Serial.println("--- Escrever no MACData ---");
Serial.println("");

Wire.beginTransmission(85);
Wire.write(0x40);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x40
que é MACData");
Wire.write(0x00);
Serial.println("Escreve MSB no endereço 0x40");
Wire.endTransmission();

Wire.beginTransmission(85);
Wire.write(0x41);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x41
que é MACData");
Wire.write(0x14);
Serial.println("Escreve LSB no endereço 0x41");
Wire.endTransmission();

Serial.println("--- Escrever no MACDataSum ---");

Wire.beginTransmission(85);
Wire.write(0x60);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x60
que é MACDataSum");
Wire.write(0x70);//E0 para 0x00A4
Serial.println("Escreve complemto da soma no endereço 0x60");
```

```
Wire.endTransmission();

Serial.println("--- Escrever no MACDataLen ---");

Wire.beginTransmission(85);
Wire.write(0x61);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x61
que é MACDataLen");
Wire.write(0x06);
Serial.println("Escreve 4+numero de bits no endereço 0x61");
Wire.endTransmission();

Serial.println("---------- Termina o processo de escrever ------------");

Serial.println();
Serial.println("---------- LER O QUE ESCREVEU ------------");
Serial.println();

Wire.beginTransmission(85);
Serial.println("Envia 85");
Wire.write(0x3E);
Serial.println("Aponta para o endereço Manufacturer Access Control 0x3E");
Wire.write(0x3A);
Serial.println("Escreve LSB do endereço no Manufacturer Access Control 0x3E");
Wire.endTransmission();

Wire.beginTransmission(85);
Serial.println("Envia 85");
Wire.write(0x3F);
Serial.println("Aponta para o endereço Manufacturer Access Control 0x3F");
Wire.write(0x41);
Serial.println("Escreve MSB do endereço no Manufacturer Access Control 0x3F");
Wire.endTransmission();

Serial.println("");
Serial.println("--- Termina de Escrever no Manufacturer Access Control ---");
Serial.println("");


Serial.println("--- Ler o Manufacturer Access Control para conferir ---");
Wire.beginTransmission(85);
Wire.write(0x3E);
Serial.println("Aponta para o endereço que quer ler, no caso 0x3E");
Wire.endTransmission();

Wire.requestFrom(85,2);
Serial.println("Envia 85 e requisita 2 bytes");
if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
```

```
Serial.println("Byte Recebido possui dois bytes");
MACaccess1_1=Wire.read();
MACaccess1_2=Wire.read();
}
Serial.print("Valor contido no MACaccess1: ");
Serial.println(MACaccess1_1);

Serial.print("Valor contido no MACaccess2: ");
Serial.println(MACaccess1_2);
Wire.endTransmission();


Serial.println("");
Serial.println("--- Ler o MacData para ver se escreveu certo ---");
Serial.println("");

Wire.beginTransmission(85);
Serial.println("Envia 85 para apontar para MACData");
Wire.write(0x40);
Serial.println("Aponta para o endereço de MACData");
Wire.endTransmission();

Wire.requestFrom(85,2);
Serial.println("Envia 85 e requisita 2 bytes");
if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
Serial.println("Byte Recebido possui dois bytes");
MACdata1_1=Wire.read();
MACdata1_2=Wire.read();
}
Serial.print("Valor contido no MACdata1: ");
Serial.println(MACdata1_1);

Serial.print("Valor contido no MACdata2: ");
Serial.println(MACdata1_2);

Wire.endTransmission();

Op_Config = ((MACdata1_1 << 8) + MACdata1_2);

Serial.print("Valor contido no Registrador: ");
Serial.println(Op_Config);

Serial.println("--- Ler o MACDataSum ---");

Wire.beginTransmission(85);
Wire.write(0x60);
Serial.println("Aponta para o endereço que quer ler, no caso 0x60
que é MACDatasum");
```

```
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  Serial.println("Envia 85 e requisita 2 bytes");
  if (2 <= Wire.available()){ //confere se foram recebidos um bytes
  Serial.println("Byte Recebido possui um bytes");
  MACdatasum1_1=Wire.read();
  MACdatasum1_2=Wire.read();
  }
  Serial.print("Valor contido no MACdatasum1: ");
  Serial.println(MACdatasum1_1);
  Serial.print("Valor contido no MACdatasum2: ");
  Serial.println(MACdatasum1_2);
  Wire.endTransmission();

  Serial.println("--- Ler o MACDataLen ---");

  Wire.beginTransmission(85);
  Wire.write(0x61);
  Serial.println("Aponta para o endereco que quer ler, no caso 0x61
  que é MACDataLen");
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  Serial.println("Envia 85 e requisita 2 bytes");
  if (2 <= Wire.available()){ //confere se foram recebidos um bytes
  Serial.println("Byte Recebido possui um bytes");
  MACdatalen1_1=Wire.read();
  MACdatalen1_2=Wire.read();
  }
  Serial.print("Valor contido no MACdatalen1: ");
  Serial.println(MACdatalen1_1);
  Serial.print("Valor contido no MACdatalen2: ");
  Serial.println(MACdatalen1_2);
  Wire.endTransmission();


  Serial.println("---------------- Fim da Transmissao -----------------");
  delay(1000); //espera 1 segundo para rodar o programa de novo

  Serial.println("");
  Serial.println("Espera 5 segundos para novo ciclo");
  Serial.println("");

  delay(1000);
}
```

C.3.2 Função Set_Pin_Control_Config

```
#include<Wire.h>


int MACaccess2_1;
int MACaccess2_2;
int MACdata2_1;
int MACdata2_2;
int MACdatasum2_1;
int MACdatasum2_2;
int MACdatalen2_1;
int MACdatalen2_2;
long Pin_Control_Config;


void Set_Pin_Control_Config() {
  Serial.println("---------------- Inicio da Transmissao -----------------");

  Wire.beginTransmission(85);
  Serial.println("Envia 85");
  Wire.write(0x3E);
  Serial.println("Aponta para o endereço Manufacturer Access Control 0x3E");
  Wire.write(0x3D);
  Serial.println("Escreve LSB do endereço no Manufacturer Access Control 0x3E");
  Wire.endTransmission();

  Wire.beginTransmission(85);
  Serial.println("Envia 85");
  Wire.write(0x3F);
  Serial.println("Aponta para o endereço Manufacturer Access Control 0x3F");
  Wire.write(0x41);
  Serial.println("Escreve MSB do endereço no Manufacturer Access Control 0x3F");
  Wire.endTransmission();

  Serial.println("");
  Serial.println("--- Termina de Escrever no Manufacturer Access Control ---");
  Serial.println("");

  Serial.println("--- Ler o Manufacturer Access Control para Conferir ---");
  Wire.beginTransmission(85);
  Wire.write(0x3E);
  Serial.println("Aponta para o endereço que quer ler, no caso 0x3E");
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  Serial.println("Envia 85 e requisita 2 bytes");
  if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
```

```
Serial.println("Byte Recebido possui dois bytes");
MACaccess2_1=Wire.read();
MACaccess2_2=Wire.read();
}
Serial.print("Valor contido no MACaccess1: ");
Serial.println(MACaccess2_1);

Serial.print("Valor contido no MACaccess2: ");
Serial.println(MACaccess2_2);
Wire.endTransmission();

Serial.println("");
Serial.println("--- Escrever no MACData ---");
Serial.println("");

Wire.beginTransmission(85);
Wire.write(0x40);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x40
que é MACData");
Wire.write(0x0A);
Serial.println("Escreve MSB no endereço");
Wire.endTransmission();

Serial.println("--- Escrever no MACDataSum ---");

Wire.beginTransmission(85);
Wire.write(0x60);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x60
que é MACDataSum");
Wire.write(0x77);//67 para 1A
Serial.println("Escreve complemto da soma no endereço 0x60");
Wire.endTransmission();

Serial.println("--- Escrever no MACDataLen ---");

Wire.beginTransmission(85);
Wire.write(0x61);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x61
que é MACDataLen");
Wire.write(0x05);
Serial.println("Escreve 4+numero de bits no endereço 0x61");
Wire.endTransmission();

Serial.println("---------- Termina o processo de escrever ------------");

Serial.println();
Serial.println("---------- LER O QUE ESCREVEU ------------");
Serial.println();
```

```
Wire.beginTransmission(85);
Serial.println("Envia 85");
Wire.write(0x3E);
Serial.println("Aponta para o endereço Manufacturer Access Control 0x3E");
Wire.write(0x3D);
Serial.println("Escreve LSB do endereço no Manufacturer Access Control 0x3E");
Wire.endTransmission();

Wire.beginTransmission(85);
Serial.println("Envia 85");
Wire.write(0x3F);
Serial.println("Aponta para o endereço Manufacturer Access Control 0x3F");
Wire.write(0x41);
Serial.println("Escreve MSB do endereço no Manufacturer Access Control 0x3F");
Wire.endTransmission();

Serial.println("");
Serial.println("--- Termina de Escrever no Manufacturer Access Control ---");
Serial.println("");


Serial.println("--- Ler o Manufacturer Access Control para conferir ---");
Wire.beginTransmission(85);
Wire.write(0x3E);
Serial.println("Aponta para o endereço que quer ler, no caso 0x3E");
Wire.endTransmission();

Wire.requestFrom(85,2);
Serial.println("Envia 85 e requisita 2 bytes");
if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
Serial.println("Byte Recebido possui dois bytes");
MACaccess2_1=Wire.read();
MACaccess2_2=Wire.read();
}
Serial.print("Valor contido no MACaccess1: ");
Serial.println(MACaccess2_1);

Serial.print("Valor contido no MACaccess2: ");
Serial.println(MACaccess2_2);
Wire.endTransmission();


Serial.println("");
Serial.println("--- Ler o MacData para ver se escreveu certo ---");
Serial.println("");

Wire.beginTransmission(85);
```

```
Serial.println("Envia 85 para apontar para MACData");
Wire.write(0x40);
Serial.println("Aponta para o endereço de MACData");
Wire.endTransmission();

Wire.requestFrom(85,1);
Serial.println("Envia 85 e requisita 2 bytes");
if (1 <= Wire.available()){ //confere se foram recebidos dois bytes
Serial.println("Byte Recebido possui dois bytes");
MACdata2_1=Wire.read();
}
Serial.print("Valor contido no MACdata1: ");
Serial.println(MACdata2_1);
Wire.endTransmission();

Pin_Control_Config = MACdata2_1;

Serial.print("Valor contido no Registrador: ");
Serial.println(Pin_Control_Config);

Serial.println("--- Ler o MACDataSum ---");

Wire.beginTransmission(85);
Wire.write(0x60);
Serial.println("Aponta para o endereço que quer ler, no caso 0x60
que é MACDatasum");
Wire.endTransmission();

Wire.requestFrom(85,2);
Serial.println("Envia 85 e requisita 2 bytes");
if (2 <= Wire.available()){ //confere se foram recebidos um bytes
Serial.println("Byte Recebido possui um bytes");
MACdatasum2_1=Wire.read();
MACdatasum2_2=Wire.read();
}
Serial.print("Valor contido no MACdatasum1: ");
Serial.println(MACdatasum2_1);
Serial.print("Valor contido no MACdatasum2: ");
Serial.println(MACdatasum2_2);
Wire.endTransmission();

Serial.println("--- Ler o MACDataLen ---");

Wire.beginTransmission(85);
Wire.write(0x61);
Serial.println("Aponta para o endereco que quer ler, no caso 0x61
que é MACDataLen");
Wire.endTransmission();
```

```
  Wire.requestFrom(85,2);
  Serial.println("Envia 85 e requisita 2 bytes");
  if (2 <= Wire.available()){ //confere se foram recebidos um bytes
  Serial.println("Byte Recebido possui um bytes");
  MACdatalen2_1=Wire.read();
  MACdatalen2_2=Wire.read();
  }
  Serial.print("Valor contido no MACdatalen1: ");
  Serial.println(MACdatalen2_1);
  Serial.print("Valor contido no MACdatalen2: ");
  Serial.println(MACdatalen2_2);
  Wire.endTransmission();


  Serial.println("---------------- Fim da Transmissao -----------------");
  delay(1000); //espera 1 segundo para rodar o programa de novo

  Serial.println("");
  Serial.println("Espera 5 segundos para novo ciclo");
  Serial.println("");

  delay(1000);
}
```

C.3.3 Função Set_MSI

```
#include<Wire.h>


int MACaccess3_1;
int MACaccess3_2;
int MACdata3_1;
int MACdata3_2;
int MACdatasum3_1;
int MACdatasum3_2;
int MACdatalen3_1;
int MACdatalen3_2;
long Manufacturing_Status_Init;



void Set_MSI() {
   Serial.println("---------------- Inicio da Transmissao -----------------");

  Wire.beginTransmission(85);
  Serial.println("Envia 85");
```

```
Wire.write(0x3E);
Serial.println("Aponta para o endereço Manufacturer Access Control 0x3E");
Wire.write(0xD7);
Serial.println("Escreve LSB do endereço no Manufacturer Access Control 0x3E");
Wire.endTransmission();

Wire.beginTransmission(85);
Serial.println("Envia 85");
Wire.write(0x3F);
Serial.println("Aponta para o endereço Manufacturer Access Control 0x3F");
Wire.write(0x40);
Serial.println("Escreve MSB do endereço no Manufacturer Access Control 0x3F");
Wire.endTransmission();

Serial.println("");
Serial.println("--- Termina de Escrever no Manufacturer Access Control ---");
Serial.println("");

Serial.println("--- Ler o Manufacturer Access Control para Conferir ---");
Wire.beginTransmission(85);
Wire.write(0x3E);
Serial.println("Aponta para o endereço que quer ler, no caso 0x3E");
Wire.endTransmission();

Wire.requestFrom(85,2);
Serial.println("Envia 85 e requisita 2 bytes");
if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
Serial.println("Byte Recebido possui dois bytes");
MACaccess3_1=Wire.read();
MACaccess3_2=Wire.read();
}
Serial.print("Valor contido no MACaccess1: ");
Serial.println(MACaccess3_1);

Serial.print("Valor contido no MACaccess2: ");
Serial.println(MACaccess3_2);
Wire.endTransmission();

Serial.println("");
Serial.println("--- Escrever no MACData ---");
Serial.println("");

Wire.beginTransmission(85);
Wire.write(0x40);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x40
que é MACData");
Wire.write(0x00);
Serial.println("Escreve MSB no endereço 0x40");
```

```
Wire.endTransmission();

Wire.beginTransmission(85);
Wire.write(0x41);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x41
que é MACData");
Wire.write(0x7B);
Serial.println("Escreve LSB no endereço 0x41");
Wire.endTransmission();

Serial.println("--- Escrever no MACDataSum ---");

Wire.beginTransmission(85);
Wire.write(0x60);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x60
que é MACDataSum");
Wire.write(0x6D); //2D
Serial.println("Escreve complemto da soma no endereço 0x60");
Wire.endTransmission();

Serial.println("--- Escrever no MACDataLen ---");

Wire.beginTransmission(85);
Wire.write(0x61);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x61
que é MACDataLen");
Wire.write(0x06);
Serial.println("Escreve 4+numero de bits no endereço 0x61");
Wire.endTransmission();

Serial.println("---------- Termina o processo de escrever ------------");

Serial.println();
Serial.println("---------- LER O QUE ESCREVEU ------------");
Serial.println();

Wire.beginTransmission(85);
Serial.println("Envia 85");
Wire.write(0x3E);
Serial.println("Aponta para o endereço Manufacturer Access Control 0x3E");
Wire.write(0xD7);
Serial.println("Escreve LSB do endereço no Manufacturer Access Control 0x3E");
Wire.endTransmission();

Wire.beginTransmission(85);
Serial.println("Envia 85");
Wire.write(0x3F);
Serial.println("Aponta para o endereço Manufacturer Access Control 0x3F");
```

```
Wire.write(0x40);
Serial.println("Escreve MSB do endereço no Manufacturer Access Control 0x3F");
Wire.endTransmission();

Serial.println("");
Serial.println("--- Termina de Escrever no Manufacturer Access Control ---");
Serial.println("");


Serial.println("--- Ler o Manufacturer Access Control para conferir ---");
Wire.beginTransmission(85);
Wire.write(0x3E);
Serial.println("Aponta para o endereço que quer ler, no caso 0x3E");
Wire.endTransmission();

Wire.requestFrom(85,2);
Serial.println("Envia 85 e requisita 2 bytes");
if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
Serial.println("Byte Recebido possui dois bytes");
MACaccess3_1=Wire.read();
MACaccess3_2=Wire.read();
}
Serial.print("Valor contido no MACaccess1: ");
Serial.println(MACaccess3_1);

Serial.print("Valor contido no MACaccess2: ");
Serial.println(MACaccess3_2);
Wire.endTransmission();


Serial.println("");
Serial.println("--- Ler o MacData para ver se escreveu certo ---");
Serial.println("");

Wire.beginTransmission(85);
Serial.println("Envia 85 para apontar para MACData");
Wire.write(0x40);
Serial.println("Aponta para o endereço de MACData");
Wire.endTransmission();

Wire.requestFrom(85,2);
Serial.println("Envia 85 e requisita 2 bytes");
if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
Serial.println("Byte Recebido possui dois bytes");
MACdata3_1=Wire.read();
MACdata3_2=Wire.read();
}
Serial.print("Valor contido no MACdata1: ");
```

```
Serial.println(MACdata3_1);

Serial.print("Valor contido no MACdata2: ");
Serial.println(MACdata3_2);

Wire.endTransmission();

Manufacturing_Status_Init = ((MACdata3_1 << 8) + MACdata3_2);

Serial.print("Valor contido no Registrador: ");
Serial.println(Manufacturing_Status_Init);

Serial.println("--- Ler o MACDataSum ---");

Wire.beginTransmission(85);
Wire.write(0x60);
Serial.println("Aponta para o endereço que quer ler, no caso 0x60
que é MACDatasum");
Wire.endTransmission();

Wire.requestFrom(85,2);
Serial.println("Envia 85 e requisita 2 bytes");
if (2 <= Wire.available()){ //confere se foram recebidos um bytes
Serial.println("Byte Recebido possui um bytes");
MACdatasum3_1=Wire.read();
MACdatasum3_2=Wire.read();
}
Serial.print("Valor contido no MACdatasum1: ");
Serial.println(MACdatasum3_1);
Serial.print("Valor contido no MACdatasum2: ");
Serial.println(MACdatasum3_2);
Wire.endTransmission();

Serial.println("--- Ler o MACDataLen ---");

Wire.beginTransmission(85);
Wire.write(0x61);
Serial.println("Aponta para o endereco que quer ler, no caso 0x61
que é MACDataLen");
Wire.endTransmission();

Wire.requestFrom(85,2);
Serial.println("Envia 85 e requisita 2 bytes");
if (2 <= Wire.available()){ //confere se foram recebidos um bytes
Serial.println("Byte Recebido possui um bytes");
MACdatalen3_1=Wire.read();
MACdatalen3_2=Wire.read();
}
```

```
  Serial.print("Valor contido no MACdatalen1: ");
  Serial.println(MACdatalen3_1);
  Serial.print("Valor contido no MACdatalen2: ");
  Serial.println(MACdatalen3_2);
  Wire.endTransmission();


  Serial.println("---------------- Fim da Transmissao -----------------");
  delay(1000); //espera 1 segundo para rodar o programa de novo

  Serial.println("");
  Serial.println("Espera 5 segundos para novo ciclo");
  Serial.println("");

  delay(1000);
}
```

C.3.4 Função Set_CEDV_Gauging_Config

```
#include<Wire.h>


int MACaccess4_1;
int MACaccess4_2;
int MACdata4_1;
int MACdata4_2;
int MACdatasum4_1;
int MACdatasum4_2;
int MACdatalen4_1;
int MACdatalen4_2;
long CEDV_Gauging_Config;


void Set_CEDV_Gauging_Config() {
   Serial.println("---------------- Inicio da Transmissao -----------------");

  Wire.beginTransmission(85);
  Serial.println("Envia 85");
  Wire.write(0x3E);
  Serial.println("Aponta para o endereço Manufacturer Access Control 0x3E");
  Wire.write(0x4B);
  Serial.println("Escreve LSB do endereço no Manufacturer Access Control 0x3E");
  Wire.endTransmission();

  Wire.beginTransmission(85);
  Serial.println("Envia 85");
  Wire.write(0x3F);
```

```
Serial.println("Aponta para o endereço Manufacturer Access Control 0x3F");
Wire.write(0x42);
Serial.println("Escreve MSB do endereço no Manufacturer Access Control 0x3F");
Wire.endTransmission();

Serial.println("");
Serial.println("--- Termina de Escrever no Manufacturer Access Control ---");
Serial.println("");

Serial.println("--- Ler o Manufacturer Access Control para Conferir ---");
Wire.beginTransmission(85);
Wire.write(0x3E);
Serial.println("Aponta para o endereço que quer ler, no caso 0x3E");
Wire.endTransmission();

Wire.requestFrom(85,2);
Serial.println("Envia 85 e requisita 2 bytes");
if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
Serial.println("Byte Recebido possui dois bytes");
MACaccess4_1=Wire.read();
MACaccess4_2=Wire.read();
}
Serial.print("Valor contido no MACaccess1: ");
Serial.println(MACaccess4_1);

Serial.print("Valor contido no MACaccess2: ");
Serial.println(MACaccess4_2);
Wire.endTransmission();

Serial.println("");
Serial.println("--- Escrever no MACData ---");
Serial.println("");

Wire.beginTransmission(85);
Wire.write(0x40);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x40
que é MACData");
Wire.write(0x11);
Serial.println("Escreve MSB no endereço 0x40");
Wire.endTransmission();

Wire.beginTransmission(85);
Wire.write(0x41);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x41
que é MACData");
Wire.write(0x0A);
Serial.println("Escreve LSB no endereço 0x41");
Wire.endTransmission();
```

```
Serial.println("--- Escrever no MACDataSum ---");

Wire.beginTransmission(85);
Wire.write(0x60);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x60
que é MACDataSum");
Wire.write(0x57);
Serial.println("Escreve complemto da soma no endereço 0x60");
Wire.endTransmission();

Serial.println("--- Escrever no MACDataLen ---");

Wire.beginTransmission(85);
Wire.write(0x61);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x61
que é MACDataLen");
Wire.write(0x06);
Serial.println("Escreve 4+numero de bits no endereço 0x61");
Wire.endTransmission();

Serial.println("---------- Termina o processo de escrever ------------");

Serial.println();
Serial.println("---------- LER O QUE ESCREVEU ------------");
Serial.println();

Wire.beginTransmission(85);
Serial.println("Envia 85");
Wire.write(0x3E);
Serial.println("Aponta para o endereço Manufacturer Access Control 0x3E");
Wire.write(0x4B);
Serial.println("Escreve LSB do endereço no Manufacturer Access Control 0x3E");
Wire.endTransmission();

Wire.beginTransmission(85);
Serial.println("Envia 85");
Wire.write(0x3F);
Serial.println("Aponta para o endereço Manufacturer Access Control 0x3F");
Wire.write(0x42);
Serial.println("Escreve MSB do endereço no Manufacturer Access Control 0x3F");
Wire.endTransmission();

Serial.println("");
Serial.println("--- Termina de Escrever no Manufacturer Access Control ---");
Serial.println("");
```

```
Serial.println("--- Ler o Manufacturer Access Control para conferir ---");
Wire.beginTransmission(85);
Wire.write(0x3E);
Serial.println("Aponta para o endereço que quer ler, no caso 0x3E");
Wire.endTransmission();

Wire.requestFrom(85,2);
Serial.println("Envia 85 e requisita 2 bytes");
if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
Serial.println("Byte Recebido possui dois bytes");
MACaccess4_1=Wire.read();
MACaccess4_2=Wire.read();
}
Serial.print("Valor contido no MACaccess1: ");
Serial.println(MACaccess4_1);

Serial.print("Valor contido no MACaccess2: ");
Serial.println(MACaccess4_2);
Wire.endTransmission();


Serial.println("");
Serial.println("--- Ler o MacData para ver se escreveu certo ---");
Serial.println("");

Wire.beginTransmission(85);
Serial.println("Envia 85 para apontar para MACData");
Wire.write(0x40);
Serial.println("Aponta para o endereço de MACData");
Wire.endTransmission();

Wire.requestFrom(85,2);
Serial.println("Envia 85 e requisita 2 bytes");
if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
Serial.println("Byte Recebido possui dois bytes");
MACdata4_1=Wire.read();
MACdata4_2=Wire.read();
}
Serial.print("Valor contido no MACdata1: ");
Serial.println(MACdata4_1);

Serial.print("Valor contido no MACdata2: ");
Serial.println(MACdata4_2);

Wire.endTransmission();

CEDV_Gauging_Config = ((MACdata4_1 << 8) + MACdata4_2);
```

```
Serial.print("Valor contido no Registrador: ");
Serial.println(CEDV_Gauging_Config);

Serial.println("--- Ler o MACDataSum ---");

Wire.beginTransmission(85);
Wire.write(0x60);
Serial.println("Aponta para o endereço que quer ler, no caso 0x60
que é MACDatasum");
Wire.endTransmission();

Wire.requestFrom(85,2);
Serial.println("Envia 85 e requisita 2 bytes");
if (2 <= Wire.available()){ //confere se foram recebidos um bytes
Serial.println("Byte Recebido possui um bytes");
MACdatasum4_1=Wire.read();
MACdatasum4_2=Wire.read();
}
Serial.print("Valor contido no MACdatasum1: ");
Serial.println(MACdatasum4_1);
Serial.print("Valor contido no MACdatasum2: ");
Serial.println(MACdatasum4_2);
Wire.endTransmission();

Serial.println("--- Ler o MACDataLen ---");

Wire.beginTransmission(85);
Wire.write(0x61);
Serial.println("Aponta para o endereco que quer ler, no caso 0x61
que é MACDataLen");
Wire.endTransmission();

Wire.requestFrom(85,2);
Serial.println("Envia 85 e requisita 2 bytes");
if (2 <= Wire.available()){ //confere se foram recebidos um bytes
Serial.println("Byte Recebido possui um bytes");
MACdatalen4_1=Wire.read();
MACdatalen4_2=Wire.read();
}
Serial.print("Valor contido no MACdatalen1: ");
Serial.println(MACdatalen4_1);
Serial.print("Valor contido no MACdatalen2: ");
Serial.println(MACdatalen4_2);
Wire.endTransmission();


Serial.println("---------------- Fim da Transmissao -----------------");
delay(1000); //espera 1 segundo para rodar o programa de novo
```

```
  Serial.println("");
  Serial.println("Espera 5 segundos para novo ciclo");
  Serial.println("");

  delay(1000);
}
```

C.3.5 Função Set_Accumulated_Charge

```
#include<Wire.h>


int MACaccess5_1;
int MACaccess5_2;
int MACdata5_1;
int MACdata5_2;
int MACdatasum5_1;
int MACdatasum5_2;
int MACdatalen5_1;
int MACdatalen5_2;
long Accumulated_Charge;


void Set_Accumulated_Charge() {
  Serial.println("---------------- Inicio da Transmissao -----------------");

  Wire.beginTransmission(85);
  Serial.println("Envia 85");
  Wire.write(0x3E);
  Serial.println("Aponta para o endereço Manufacturer Access Control 0x3E");
  Wire.write(0x6C);
  Serial.println("Escreve LSB do endereço no Manufacturer Access Control 0x3E");
  Wire.endTransmission();

  Wire.beginTransmission(85);
  Serial.println("Envia 85");
  Wire.write(0x3F);
  Serial.println("Aponta para o endereço Manufacturer Access Control 0x3F");
  Wire.write(0x41);
  Serial.println("Escreve MSB do endereço no Manufacturer Access Control 0x3F");
  Wire.endTransmission();

  Serial.println("");
  Serial.println("--- Termina de Escrever no Manufacturer Access Control ---");
  Serial.println("");
```

```
Serial.println("--- Ler o Manufacturer Access Control para Conferir ---");
Wire.beginTransmission(85);
Wire.write(0x3E);
Serial.println("Aponta para o endereço que quer ler, no caso 0x3E");
Wire.endTransmission();

Wire.requestFrom(85,2);
Serial.println("Envia 85 e requisita 2 bytes");
if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
Serial.println("Byte Recebido possui dois bytes");
MACaccess5_1=Wire.read();
MACaccess5_2=Wire.read();
}
Serial.print("Valor contido no MACaccess1: ");
Serial.println(MACaccess5_1);

Serial.print("Valor contido no MACaccess2: ");
Serial.println(MACaccess5_2);
Wire.endTransmission();

Serial.println("");
Serial.println("--- Escrever no MACData ---");
Serial.println("");

Wire.beginTransmission(85);
Wire.write(0x40);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x40
que é MACData");
Wire.write(0x03);
Serial.println("Escreve MSB no endereço 0x40");
Wire.endTransmission();

Wire.beginTransmission(85);
Wire.write(0x41);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x40
que é MACData");
Wire.write(0xE8);
Serial.println("Escreve LSB no endereço 0x40");
Wire.endTransmission();

Serial.println("--- Escrever no MACDataSum ---");

Wire.beginTransmission(85);
Wire.write(0x60);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x60
que é MACDataSum");
Wire.write(0x67);
Serial.println("Escreve complemto da soma no endereço 0x60");
```

```
Wire.endTransmission();

Serial.println("--- Escrever no MACDataLen ---");

Wire.beginTransmission(85);
Wire.write(0x61);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x61
que é MACDataLen");
Wire.write(0x06);
Serial.println("Escreve 4+numero de bits no endereço 0x61");
Wire.endTransmission();

Serial.println("---------- Termina o processo de escrever ------------");

Serial.println();
Serial.println("---------- LER O QUE ESCREVEU ------------");
Serial.println();

Wire.beginTransmission(85);
Serial.println("Envia 85");
Wire.write(0x3E);
Serial.println("Aponta para o endereço Manufacturer Access Control 0x3E");
Wire.write(0x6C);
Serial.println("Escreve LSB do endereço no Manufacturer Access Control 0x3E");
Wire.endTransmission();

Wire.beginTransmission(85);
Serial.println("Envia 85");
Wire.write(0x3F);
Serial.println("Aponta para o endereço Manufacturer Access Control 0x3F");
Wire.write(0x41);
Serial.println("Escreve MSB do endereço no Manufacturer Access Control 0x3F");
Wire.endTransmission();

Serial.println("");
Serial.println("--- Termina de Escrever no Manufacturer Access Control ---");
Serial.println("");


Serial.println("--- Ler o Manufacturer Access Control para conferir ---");
Wire.beginTransmission(85);
Wire.write(0x3E);
Serial.println("Aponta para o endereço que quer ler, no caso 0x3E");
Wire.endTransmission();

Wire.requestFrom(85,2);
Serial.println("Envia 85 e requisita 2 bytes");
if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
```

```
Serial.println("Byte Recebido possui dois bytes");
MACaccess5_1=Wire.read();
MACaccess5_2=Wire.read();
}
Serial.print("Valor contido no MACaccess1: ");
Serial.println(MACaccess5_1);

Serial.print("Valor contido no MACaccess2: ");
Serial.println(MACaccess5_2);
Wire.endTransmission();


Serial.println("");
Serial.println("--- Ler o MacData para ver se escreveu certo ---");
Serial.println("");

Wire.beginTransmission(85);
Serial.println("Envia 85 para apontar para MACData");
Wire.write(0x40);
Serial.println("Aponta para o endereço de MACData");
Wire.endTransmission();

Wire.requestFrom(85,2);
Serial.println("Envia 85 e requisita 2 bytes");
if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
Serial.println("Byte Recebido possui dois bytes");
MACdata5_1=Wire.read();
MACdata5_2=Wire.read();
}
Serial.print("Valor contido no MACdata1: ");
Serial.println(MACdata5_1);

Serial.print("Valor contido no MACdata2: ");
Serial.println(MACdata5_2);
Wire.endTransmission();

Accumulated_Charge = ((MACdata5_1 << 8) + MACdata5_2);

Serial.print("Valor contido no Registrador: ");
Serial.println(Accumulated_Charge);

Serial.println("--- Ler o MACDataSum ---");

Wire.beginTransmission(85);
Wire.write(0x60);
Serial.println("Aponta para o endereço que quer ler, no caso 0x60
que é MACDatasum");
Wire.endTransmission();
```

```
  Wire.requestFrom(85,2);
  Serial.println("Envia 85 e requisita 2 bytes");
  if (2 <= Wire.available()){ //confere se foram recebidos um bytes
  Serial.println("Byte Recebido possui um bytes");
  MACdatasum5_1=Wire.read();
  MACdatasum5_2=Wire.read();
  }
  Serial.print("Valor contido no MACdatasum1: ");
  Serial.println(MACdatasum5_1);
  Serial.print("Valor contido no MACdatasum2: ");
  Serial.println(MACdatasum5_2);
  Wire.endTransmission();

  Serial.println("--- Ler o MACDataLen ---");

  Wire.beginTransmission(85);
  Wire.write(0x61);
  Serial.println("Aponta para o endereco que quer ler, no caso 0x61
  que é MACDataLen");
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  Serial.println("Envia 85 e requisita 2 bytes");
  if (2 <= Wire.available()){ //confere se foram recebidos um bytes
  Serial.println("Byte Recebido possui um bytes");
  MACdatalen5_1=Wire.read();
  MACdatalen5_2=Wire.read();
  }
  Serial.print("Valor contido no MACdatalen1: ");
  Serial.println(MACdatalen5_1);
  Serial.print("Valor contido no MACdatalen2: ");
  Serial.println(MACdatalen5_2);
  Wire.endTransmission();

  Serial.println("---------------- Fim da Transmissao -----------------");
  delay(1000); //espera 1 segundo para rodar o programa de novo

  Serial.println("");
  Serial.println("Espera 5 segundos para novo ciclo");
  Serial.println("");

  delay(1000);
}
```

C.3.6 Função Set_FCC

```
#include<Wire.h>


int MACaccess6_1;
int MACaccess6_2;
int MACdata6_1;
int MACdata6_2;
int MACdatasum6_1;
int MACdatasum6_2;
int MACdatalen6_1;
int MACdatalen6_2;
long FCC;


void Set_FCC() {
   Serial.println("---------------- Inicio da Transmissao -----------------");

  Wire.beginTransmission(85);
  Serial.println("Envia 85");
  Wire.write(0x3E);
  Serial.println("Aponta para o endereço Manufacturer Access Control 0x3E");
  Wire.write(0xC0);
  Serial.println("Escreve LSB do endereço no Manufacturer Access Control 0x3E");
  Wire.endTransmission();

  Wire.beginTransmission(85);
  Serial.println("Envia 85");
  Wire.write(0x3F);
  Serial.println("Aponta para o endereço Manufacturer Access Control 0x3F");
  Wire.write(0x40);
  Serial.println("Escreve MSB do endereço no Manufacturer Access Control 0x3F");
  Wire.endTransmission();

  Serial.println("");
  Serial.println("--- Termina de Escrever no Manufacturer Access Control ---");
  Serial.println("");

  Serial.println("--- Ler o Manufacturer Access Control para Conferir ---");
  Wire.beginTransmission(85);
  Wire.write(0x3E);
  Serial.println("Aponta para o endereço que quer ler, no caso 0x3E");
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  Serial.println("Envia 85 e requisita 2 bytes");
  if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
```

```
Serial.println("Byte Recebido possui dois bytes");
MACaccess6_1=Wire.read();
MACaccess6_2=Wire.read();
}
Serial.print("Valor contido no MACaccess1: ");
Serial.println(MACaccess6_1);

Serial.print("Valor contido no MACaccess2: ");
Serial.println(MACaccess6_2);
Wire.endTransmission();

Serial.println("");
Serial.println("--- Escrever no MACData ---");
Serial.println("");

Wire.beginTransmission(85);
Wire.write(0x40);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x40
que é MACData");
Wire.write(0x01);
Serial.println("Escreve MSB no endereço 0x40");
Wire.endTransmission();

Wire.beginTransmission(85);
Wire.write(0x41);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x41
que é MACData");
Wire.write(0x3C);
Serial.println("Escreve LSB no endereço 0x41");
Wire.endTransmission();

Serial.println("--- Escrever no MACDataSum ---");

Wire.beginTransmission(85);
Wire.write(0x60);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x60
que é MACDataSum");
Wire.write(0xD2);
Serial.println("Escreve complemto da soma no endereço 0x60");
Wire.endTransmission();

Serial.println("--- Escrever no MACDataLen ---");

Wire.beginTransmission(85);
Wire.write(0x61);
Serial.println("Aponta para o endereço que quer escrever, no caso 0x61
que é MACDataLen");
Wire.write(0x06);
```

```
Serial.println("Escreve 4+numero de bits no endereço 0x61");
Wire.endTransmission();

Serial.println("---------- Termina o processo de escrever ------------");

Serial.println();
Serial.println("---------- LER O QUE ESCREVEU ------------");
Serial.println();

Wire.beginTransmission(85);
Serial.println("Envia 85");
Wire.write(0x3E);
Serial.println("Aponta para o endereço Manufacturer Access Control 0x3E");
Wire.write(0xC0);
Serial.println("Escreve LSB do endereço no Manufacturer Access Control 0x3E");
Wire.endTransmission();

Wire.beginTransmission(85);
Serial.println("Envia 85");
Wire.write(0x3F);
Serial.println("Aponta para o endereço Manufacturer Access Control 0x3F");
Wire.write(0x40);
Serial.println("Escreve MSB do endereço no Manufacturer Access Control 0x3F");
Wire.endTransmission();

Serial.println("");
Serial.println("--- Termina de Escrever no Manufacturer Access Control ---");
Serial.println("");


Serial.println("--- Ler o Manufacturer Access Control para conferir ---");
Wire.beginTransmission(85);
Wire.write(0x3E);
Serial.println("Aponta para o endereço que quer ler, no caso 0x3E");
Wire.endTransmission();

Wire.requestFrom(85,2);
Serial.println("Envia 85 e requisita 2 bytes");
if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
Serial.println("Byte Recebido possui dois bytes");
MACaccess6_1=Wire.read();
MACaccess6_2=Wire.read();
}
Serial.print("Valor contido no MACaccess1: ");
Serial.println(MACaccess6_1);

Serial.print("Valor contido no MACaccess2: ");
Serial.println(MACaccess6_2);
```

```
Wire.endTransmission();


Serial.println("");
Serial.println("--- Ler o MacData para ver se escreveu certo ---");
Serial.println("");

Wire.beginTransmission(85);
Serial.println("Envia 85 para apontar para MACData");
Wire.write(0x40);
Serial.println("Aponta para o endereço de MACData");
Wire.endTransmission();

Wire.requestFrom(85,2);
Serial.println("Envia 85 e requisita 2 bytes");
if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
Serial.println("Byte Recebido possui dois bytes");
MACdata6_1=Wire.read();
MACdata6_2=Wire.read();
}
Serial.print("Valor contido no MACdata1: ");
Serial.println(MACdata6_1);

Serial.print("Valor contido no MACdata2: ");
Serial.println(MACdata6_2);

Wire.endTransmission();

FCC = ((MACdata6_1 << 8) + MACdata6_2);

Serial.print("Valor contido no Registrador: ");
Serial.println(FCC);

Serial.println("--- Ler o MACDataSum ---");

Wire.beginTransmission(85);
Wire.write(0x60);
Serial.println("Aponta para o endereço que quer ler, no caso 0x60
que é MACDatasum");
Wire.endTransmission();

Wire.requestFrom(85,2);
Serial.println("Envia 85 e requisita 2 bytes");
if (2 <= Wire.available()){ //confere se foram recebidos um bytes
Serial.println("Byte Recebido possui um bytes");
MACdatasum6_1=Wire.read();
MACdatasum6_2=Wire.read();
}
```

```
  Serial.print("Valor contido no MACdatasum1: ");
  Serial.println(MACdatasum6_1);
  Serial.print("Valor contido no MACdatasum2: ");
  Serial.println(MACdatasum6_2);
  Wire.endTransmission();

  Serial.println("--- Ler o MACDataLen ---");

  Wire.beginTransmission(85);
  Wire.write(0x61);
  Serial.println("Aponta para o endereco que quer ler, no caso 0x61
  que é MACDataLen");
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  Serial.println("Envia 85 e requisita 2 bytes");
  if (2 <= Wire.available()){ //confere se foram recebidos um bytes
  Serial.println("Byte Recebido possui um bytes");
  MACdatalen6_1=Wire.read();
  MACdatalen6_2=Wire.read();
  }
  Serial.print("Valor contido no MACdatalen1: ");
  Serial.println(MACdatalen6_1);
  Serial.print("Valor contido no MACdatalen2: ");
  Serial.println(MACdatalen6_2);
  Wire.endTransmission();


  Serial.println("---------------- Fim da Transmissao -----------------");
  delay(1000); //espera 1 segundo para rodar o programa de novo

  Serial.println("");
  Serial.println("Espera 5 segundos para novo ciclo");
  Serial.println("");

  delay(1000);
}
```

C.4 CALIBRAÇÃO

Nesta secção, apresenta-se o código para realizar a calibração da placa. Foram criadas funções para cada calibração e todas elas são chamadas no programa principal.

```
#include<Wire.h>


void setup() {
  Wire.begin(); // adiciona o master no barramento I2C
```

```
  Serial.begin(9600);
  Serial.println("\n I2C Scanner");
}

void loop() {

  CAL_EN; //entra no modo calibracao
  cc_offset;
  board_offset;
  calibracao_temp;
  calibracao_tensao;
  calibracao_corrente;

  delay(1000);
}
```

C.4.1 CAL_EN

```
#include<Wire.h>


void CAL_EN() {

  Wire.begin(); // adiciona o master no barramento I2C
  Serial.begin(9600);
  Serial.println("\n I2C Scanner");

  //Primeiro Enable o Calibration Mode
  (Conferir depois no Operation Status se esta em 1)
  Serial.println("-------- Mudar para Calibration Mode ----------");
  Wire.beginTransmission(85);
  Wire.write(0x3E);
  Wire.write(0x2D);
  Wire.endTransmission();

  Wire.beginTransmission(85);
  Wire.write(0x3F);
  Wire.write(0x00);
  Wire.endTransmission();

  delay(1000);
}
```

C.4.2 CC_Offset

```
#include<Wire.h>

int MSByte;
int LSByte;
int MACaccess1;
int MACaccess2;
int MACdata1;
int MACdata2;
int MACdatasum1;
int MACdatalen1;
long ControlStatus;
int ManufacturingStatus;

void cc_offset() {

  Wire.begin(); // adiciona o master no barramento I2C
  Serial.begin(9600);
  Serial.println("\n I2C Scanner");

  Serial.println("-------- Comando 0057 - Ler Manufacturer Status  ----------");
  Wire.beginTransmission(85);
  //Serial.println("Envia 85");
  Wire.write(0x3E);
  //Serial.println("Aponta para o endereço Manufacturer Access Control 0x3E");
  Wire.write(0x57);
  //Serial.println("Escreve 0x3A no endereço Manufacturer Access Control 0x3E");
  Wire.endTransmission();

  Wire.beginTransmission(85);
  //Serial.println("Envia 85");
  Wire.write(0x3F);
  //Serial.println("Aponta para o endereço Manufacturer Access Control 0x3F");
  Wire.write(0x00);
  //Serial.println("Escreve 0x41 no endereço Manufacturer Access Control 0x3F");
  Wire.endTransmission();

  Wire.beginTransmission(85);
  Wire.write(0x40);
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
  //Serial.println("Byte Recebido possui dois bytes");
  LSByte=Wire.read();
  MSByte=Wire.read();
  }
  Serial.print("Valor contido no LSB: ");
  Serial.println(LSByte);
```

```
Serial.print("Valor contido no MSB: ");
Serial.println(MSByte);
Wire.endTransmission();

ManufacturingStatus = ((MSByte << 8) + LSByte);

Serial.print("Valor contido no Manufacturing Status: ");
Serial.println(ManufacturingStatus);

Serial.println("");
Serial.println("--- Termina de Ler o Manufacturer Access Control ---");
Serial.println("");


Serial.println("-------- Fazer o CC Offset ----------");
Wire.beginTransmission(85);
Wire.write(0x3E);
Wire.write(0x0A);
Wire.endTransmission();

Wire.beginTransmission(85);
//Serial.println("Envia 85");
Wire.write(0x3F);
//Serial.println("Aponta para o endereço Manufacturer Access Control 0x3F");
Wire.write(0x00);
//Serial.println("Escreve 0x41 no endereço Manufacturer Access Control 0x3F");
Wire.endTransmission();

Serial.println("");
Serial.println("--- Termina de Escrever no Manufacturer Access Control ---");
Serial.println("");

delay(100);

Serial.println("-------- Comando 0000 Ler Control Status  ----------");
Wire.beginTransmission(85);
Wire.write(0x3E);
Wire.write(0x00);
Wire.endTransmission();

Wire.beginTransmission(85);
Wire.write(0x3F);
Wire.write(0x00);
Wire.endTransmission();

Wire.beginTransmission(85);
Wire.write(0x40);
```

```
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
  //Serial.println("Byte Recebido possui dois bytes");
  LSByte=Wire.read();
  MSByte=Wire.read();
  }
  Serial.print("Valor contido no LSB: ");
  Serial.println(LSByte);
  Serial.print("Valor contido no MSB: ");
  Serial.println(MSByte);
  Wire.endTransmission();

  ControlStatus = ((MSByte << 8) + LSByte);

  Serial.print("Valor contido no Control Status: ");
  Serial.println(ControlStatus);

  Serial.println("");
  Serial.println("Espera 5 segundos para novo ciclo");
  Serial.println("");

  delay(2000);
}
```

C.4.3 Board_Offset

```
#include<Wire.h>

int MSByte;
int LSByte;
int MACaccess1;
int MACaccess2;
int MACdata1;
int MACdata2;
int MACdatasum1;
int MACdatalen1;
long ControlStatus;
int ManufacturingStatus;

void board_offset() {

  Wire.begin(); // adiciona o master no barramento I2C
  Serial.begin(9600);
  Serial.println("\n I2C Scanner");
```

```
//Primeiro Enable o Calibration Mode (Conferir depois no Operation Status se esta
Serial.println("-------- Mudar para Calibration Mode ----------");
Wire.beginTransmission(85);
//Serial.println("Envia 85");
Wire.write(0x3E);
//Serial.println("Aponta para o endereço Manufacturer Access Control 0x3E");
Wire.write(0x2D);
//Serial.println("Escreve 0x3A no endereço Manufacturer Access Control 0x3E");
Wire.endTransmission();

Wire.beginTransmission(85);
//Serial.println("Envia 85");
Wire.write(0x3F);
//Serial.println("Aponta para o endereço Manufacturer Access Control 0x3F");
Wire.write(0x00);
//Serial.println("Escreve 0x41 no endereço Manufacturer Access Control 0x3F");
Wire.endTransmission();

Serial.println("");
Serial.println("--- Termina de Escrever no Manufacturer Access Control ---");
Serial.println("");


Serial.println("-------- Comando 0057 - Ler Manufacturer Status  ----------");
Wire.beginTransmission(85);
//Serial.println("Envia 85");
Wire.write(0x3E);
//Serial.println("Aponta para o endereço Manufacturer Access Control 0x3E");
Wire.write(0x57);
//Serial.println("Escreve 0x3A no endereço Manufacturer Access Control 0x3E");
Wire.endTransmission();

Wire.beginTransmission(85);
//Serial.println("Envia 85");
Wire.write(0x3F);
//Serial.println("Aponta para o endereço Manufacturer Access Control 0x3F");
Wire.write(0x00);
//Serial.println("Escreve 0x41 no endereço Manufacturer Access Control 0x3F");
Wire.endTransmission();

Wire.beginTransmission(85);
Wire.write(0x40);
Wire.endTransmission();

Wire.requestFrom(85,2);
if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
//Serial.println("Byte Recebido possui dois bytes");
```

```
LSByte=Wire.read();
MSByte=Wire.read();
}
Serial.print("Valor contido no LSB: ");
Serial.println(LSByte);

Serial.print("Valor contido no MSB: ");
Serial.println(MSByte);
Wire.endTransmission();

ManufacturingStatus = ((MSByte << 8) + LSByte);

Serial.print("Valor contido no Manufacturing Status: ");
Serial.println(ManufacturingStatus);

Serial.println("");
Serial.println("--- Termina de Ler o Manufacturer Access Control ---");
Serial.println("");

Serial.println("-------- Fazer o Board Offset ----------");
Wire.beginTransmission(85);
Wire.write(0x3E);
Wire.write(0x09);
Wire.endTransmission();

Wire.beginTransmission(85);
//Serial.println("Envia 85");
Wire.write(0x3F);
//Serial.println("Aponta para o endereço Manufacturer Access Control 0x3F");
Wire.write(0x00);
//Serial.println("Escreve 0x41 no endereço Manufacturer Access Control 0x3F");
Wire.endTransmission();

Serial.println("");
Serial.println("--- Termina de Escrever no Manufacturer Access Control ---");
Serial.println("");

delay(100);

Serial.println("-------- Comando 0000 Ler Control Status  ----------");
Wire.beginTransmission(85);
Wire.write(0x3E);
Wire.write(0x00);
Wire.endTransmission();

Wire.beginTransmission(85);
Wire.write(0x3F);
Wire.write(0x00);
```

```
  Wire.endTransmission();

  Wire.beginTransmission(85);
  Wire.write(0x40);
  Wire.endTransmission();

  Wire.requestFrom(85,2);
  if (2 <= Wire.available()){ //confere se foram recebidos dois bytes
  //Serial.println("Byte Recebido possui dois bytes");
  LSByte=Wire.read();
  MSByte=Wire.read();
  }
  Serial.print("Valor contido no LSB: ");
  Serial.println(LSByte);
  Serial.print("Valor contido no MSB: ");
  Serial.println(MSByte);
  Wire.endTransmission();

  ControlStatus = ((MSByte << 8) + LSByte);

  Serial.print("Valor contido no Control Status: ");
  Serial.println(ControlStatus);

  Serial.println("");
  Serial.println("Espera 5 segundos para novo ciclo");
  Serial.println("");

  delay(2000);
}
```

C.4.4 Calibração da Temperatura

```
#include<Wire.h>

int temp=0;
#define valor_temperatura 40;
int avgRawTemp

void calibracao_temp() {
  temp = valor_temperatura;
  Raw_Calibration_Data; //roda o programa de aquisicao de dados

  int tOffset = (temp - avgRawTemp);

  Wire.beginTransmission(85);
  Wire.write(0x3E);
  Wire.write(0x0D); //0x0E se for escolhido termistor externo
```

```
  Wire.endTransmission();

  Wire.beginTransmission(85);
  Wire.write(0x3F);
  Wire.write(0x40);
  Wire.endTransmission();

  Wire.beginTransmission(85);
  Wire.write(0x40);
  Serial.println("Aponta para o endereço que quer escrever, no caso 0x40
  que é MACData");
  Wire.write(tOffset); //tOffset considerado 1
  Serial.println("Escreve tOffset no endereço 0x40");
  Wire.endTransmission();

  Serial.println("--- Escrever no MACDataSum ---");

  Wire.beginTransmission(85);
  Wire.write(0x60);
  Serial.println("Aponta para o endereço que quer escrever, no caso 0x60
  que é MACDataSum");
  Wire.write(0xB1);
  Serial.println("Soma 0x40 + 0x0D + tOffset, faz o complemento e coloca
  em 0x60");
  Wire.endTransmission();

  Serial.println("--- Escrever no MACDataLen ---");

  Wire.beginTransmission(85);
  Wire.write(0x61);
  Serial.println("Aponta para o endereço que quer escrever, no caso 0x61
  que é MACDataLen");
  Wire.write(0x05);
  Serial.println("Escreve 0x05 no endereço 0x61");
  Wire.endTransmission();

  Serial.println("---------------- Fim da Transmissao -----------------");
  delay(1000); //espera 1 segundo para rodar o programa de novo
}
```

C.4.5 Calibração da Tensão

```
#include<Wire.h>

int voltage=0;
#define valor_tensao ;
int avgRawVoltage
```

```
void calibracao_tensao() {
  voltage = valor_temperatura;
  Raw_Calibration_Data; //roda o programa de aquisicao de dados

  int VOffset = (voltage - avgRawVoltage);

  Wire.beginTransmission(85);
  Wire.write(0x3E);
  Wire.write(0x0F);
  Wire.endTransmission();

  Wire.beginTransmission(85);
  Wire.write(0x3F);
  Wire.write(0x40);
  Wire.endTransmission();

  Wire.beginTransmission(85);
  Wire.write(0x40);
  Serial.println("Aponta para o endereço que quer escrever, no caso 0x40
  que é MACData");
  Wire.write(VOffset); //VOffset considerado 100
  Serial.println("Escreve tOffset no endereço 0x40");
  Wire.endTransmission();

  Serial.println("--- Escrever no MACDataSum ---");

  Wire.beginTransmission(85);
  Wire.write(0x60);
  Serial.println("Aponta para o endereço que quer escrever, no caso 0x60
  que é MACDataSum");
  Wire.write(0x4C);
  Serial.println("Soma 0x40 + 0x0F + VOffset, faz o complemento e coloca
  em 0x60");
  Wire.endTransmission();

  Serial.println("--- Escrever no MACDataLen ---");

  Wire.beginTransmission(85);
  Wire.write(0x61);
  Serial.println("Aponta para o endereço que quer escrever, no caso 0x61
  que é MACDataLen");
  Wire.write(0x05);
  Serial.println("Escreve 0x05 no endereço 0x61");
  Wire.endTransmission();

  Serial.println("---------------- Fim da Transmissao -----------------");
  delay(1000); //espera 1 segundo para rodar o programa de novo
```

```
}
```

C.4.6 Calibração da Corrente

```
#include<Wire.h>

int corrente=0;
#define valor_corrente 1000;
int avgRawVoltage;
int ccGain;
int ccDelta;
#define ccOffset -1432; //atualizar com valor contido em 0x4008
#define boardOffset 0; //atualizar com o valor contido em 0x400C

void calibracao_corrente() {
  corrente = valor_corrente;
  Raw_Calibration_Data; //roda o programa de aquisicao de dados

  int ccGain = corrente/(avgRawCurrent-(ccOffset + boardOffset)/16);
  double ccDelta = (ccGain * 1193046);

  Floating_Point; //roda o programa de conversao

  Wire.beginTransmission(85);
  Wire.write(0x3E);
  Wire.write(0x00);
  Wire.endTransmission();

  Wire.beginTransmission(85);
  Wire.write(0x3F);
  Wire.write(0x40);
  Wire.endTransmission();

  Wire.beginTransmission(85);
  Wire.write(0x40);
  Serial.println("Aponta para o endereço que quer escrever, no caso 0x40
  que é MACData");
  Wire.write(VOffset); //VOffset considerado 100
  Serial.println("Escreve tOffset no endereço 0x40");
  Wire.endTransmission();

  Serial.println("--- Escrever no MACDataSum ---");

  Wire.beginTransmission(85);
  Wire.write(0x60);
  Serial.println("Aponta para o endereço que quer escrever, no caso 0x60
  que é MACDataSum");
```

```
  Wire.write(0x4C);
  Serial.println("Soma 0x40 + 0x0F + VOffset, faz o complemento e
  coloca em 0x60");
  Wire.endTransmission();

  Serial.println("--- Escrever no MACDataLen ---");

  Wire.beginTransmission(85);
  Wire.write(0x61);
  Serial.println("Aponta para o endereço que quer escrever, no caso 0x61
  que é MACDataLen");
  Wire.write(0x05);
  Serial.println("Escreve 0x05 no endereço 0x61");
  Wire.endTransmission();

  Serial.println("---------------- Fim da Transmissao -----------------");
  delay(1000); //espera 1 segundo para rodar o programa de novo
}
```

C.4.7 Aquisição de Dados

```
#include<Wire.h>


int rawDataSum = 0;
int samplesToAvg = 50;
int counterNow;
int counterPrev;


void Raw_Calibration_Data() {

  Wire.begin(); // adiciona o master no barramento I2C
  Serial.begin(9600);
  Serial.println("\n I2C Scanner");

  //Primeiro Enable o Calibration Mode
  (Conferir depois no Operation Status se esta em 1)
  Serial.println("-------- Mudar para Calibration Mode ----------");
  Wire.beginTransmission(85);
  Wire.write(0x3E);
  Wire.write(0x2D);
  Wire.endTransmission();

  Wire.beginTransmission(85);
  Wire.write(0x3F);
  Wire.write(0x00);
```

```
Wire.endTransmission();

for (int loopCount = 0; loopCount <= samplesToAvg; loopCount++){
Wire.beginTransmission(85);
Wire.write(0x79);
Wire.endTransmission();

Wire.requestFrom(85,1);
counterNow = Wire.read();
counterPrev = counterNow;
Wire.endTransmission();

delay(200);

Wire.beginTransmission(85);
Wire.write(0x79);
Wire.endTransmission();

Wire.requestFrom(85,1);
counterNow = Wire.read();
Wire.endTransmission();

if(counterNow == counterPrev)
{
  delay (200);

  Wire.beginTransmission(85);
  Wire.write(0x79);
  Wire.endTransmission();

  Wire.requestFrom(85,1);
  counterNow = Wire.read();
  Wire.endTransmission();
}
else
{
  Wire.write(0x7E); //7A pra corrente e 7C pra tensao

  Wire.requestFrom(85,2);
  int LSB = Wire.read();
  int MSB = Wire.read();
  Wire.endTransmission();

  rawDataSum = ((MSB<<8) + LSB);
  counterPrev = counterNow;
}
}
// descomentar a variavel que se deseja adquirir
```

```
int avgRawTemperature = rawDataSum/samplesToAvg;
//int avgRawCurrent = rawDataSum/samplesToAvg;
//int avgRawVoltage = rawDataSum/samplesToAvg;
}
```