CENTRO UNIVERSITÁRIO UNIVATES

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

CURSO DE ENGENHARIA DA COMPUTAÇÃO

JONAS FRONCHETTI

AVALIAÇÃO E ANÁLISE DE CONSUMO DE ENERGIA MULTINÍVEL EM HARDWARE MODULAR

Lajeado

2015

JONAS FRONCHETTI

AVALIAÇÃO E ANÁLISE DE CONSUMO DE ENERGIA MULTINÍVEL EM HARDWARE MODULAR

Trabalho de Conclusão de Curso apresentado ao Centro de Ciências Exatas e Tecnológicas do Centro Universitário UNIVATES, como parte dos requisitos para a obtenção do título de bacharel em Engenharia da Computação.
Área de concentração: Engenharia da computação

ORIENTADOR: Marcelo de G. Malheiros
CO-ORIENTADOR: Pedro A. M. de Campos Velho

Lajeado

2015

JONAS FRONCHETTI

AVALIAÇÃO E ANÁLISE DE CONSUMO DE ENERGIA MULTINÍVEL EM HARDWARE MODULAR:

Este trabalho foi julgado adequado para a obtenção do título de bacharel em Engenharia da Computação do CETEC e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____
Prof. Marcelo de G. Malheiros, UNIVATES
Mestre pela UNICAMP - Campinas, Brasil

Banca Examinadora:

Prof. Marcelo de G. Malheiros, UNIVATES
Mestre pela UNICAMP - Campinas, Brasil

Prof. Alexandre Stürmer Wolf, UNIVATES
Mestre pela PUC-Rio - Rio de Janeiro, Brasil

Prof. Anderson A. Giacomolli, UNIVATES
Mestre pela UFRGS – Porto Alegre, Brasil

Coordenador do Curso de Engenharia da Computação: _____
Prof. Marcelo de G. Malheiros

Lajeado, julho de 2015.

AGRADECIMENTOS

Agradeço a minha esposa que teve paciência e compreensão durante os dias e noites de estudos que estive ausente da sua companhia, sempre me apoiando para poder seguir sempre em frete.

Aos meus pais e minha família que me deram conselhos possibilitando realizar as escolhas certas e pelo seu apoio durante as dificuldades, sabendo que não estaria sozinho para poder enfrenta-las.

Aos professores que foram fundamentais ao realizar a minha orientação e o acompanhamento durante a realização desse trabalho, também pela sua amizade e paciência, acreditando no meu potencial.

A dedicação dos professores que em minha vida acadêmica passaram um pouco do seu conhecimento durante as aulas e foram muito importantes para minha formação acadêmica.

A todos os amigos que durante o desenvolvimento desse trabalho ajudaram a tirar dúvidas e deram sua contribuição de conhecimento.

A Deus que me deu saúde e força para superar as dificuldades

RESUMO

O Brasil teve aumento exponencial no consumo de energia elétrica nos últimos anos. Além da natural expansão e construção de novas usinas, é importante haver uma transformação no modo de consumir energia, reduzindo riscos ambientais. Portanto, se torna essencial reduzir a utilização da energia, e para isso, é necessário desenvolver formas de monitoramento e medição do consumo energético. Este trabalho tem como objetivo o desenvolvimento de um sistema para medição do consumo energético multinível em sistemas computacionais, possibilitando uma análise do consumo em três níveis de uma plataforma modular de hardware. Para este desenvolvimento foi utilizada a linguagem de programação C, apoiada por hardware externo dedicado, permitindo medições entre a fonte e a tomada, entre a fonte e a placa principal, e internamente na placa para seus componentes. O foco é obter dados precisos de voltagem e corrente em vários pontos do sistema, escondendo a complexidade e apresentando uma API simples para o programador.

Palavras-chave: Consumo de Energia, Eficiência Energética, Odroid, OpenEnergyMonitor.

ABSTRACT

Brazil had exponential increase in energy consumption in recent years. Besides the natural expansion and construction of new plants, it is important to change the way we use energy, reducing environmental risks. Therefore, it becomes essential to reduce energy use, and it is also necessary to develop ways of monitoring and measuring energy consumption. This work aims to develop a system for measuring the multi-level energy consumption in computer systems, enabling an analysis of consumption at three levels of a modular hardware platform. For this development the C programming language was used, supported by dedicated external hardware, allowing measurements between the source and the outlet, between the source and the motherboard, and internally on the board to its components. The focus is to obtain accurate data for voltage and current at various points in the system, hiding complexity and presenting a simple API to the programmer.

Keywords: Consumption of Energy, Energy Efficiency, Odroid, OpenEnergyMonitor.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AC:	Alternating Current
CCI:	Cache Coherent Interconnect
CI:	Circuito Integrado
CPU:	Central Processing Unit
DC:	Direct Current
DRAM:	Dynamic Random Access Memory
EEPROM:	Electrically-Erasable Programmable Read-Only Memory
eMMC:	embedded Multi-Media Controller
FLOPs:	Floating Point Operations
GCC:	Gnu Compiler Collection
GIC:	Generic Interrupt Controller
GND:	Ground
GPL:	GNU General Public License
GPU:	Graphics Processing Unit
HDMI:	High-Definition Multimedia Interface
HPL:	High Performance LINPACK
IDE:	Integrated Development Environment
LAN:	Local Area Network
LED:	Light Emitting Diode
MME:	Ministério de Minas e Energia
MSRs:	Model-Specific Registers
OEM:	OpenEnergyMonitor
OTG:	USB On-The-Go
PCB:	Proccess Control Block

RALF:	Running Average Power Limit
SoC:	System on Chip
SRAM:	Static Random Access Memory
TRS:	Tip-Ring-Sleeve
UFS:	Universidade Federal de Sergipe
USB:	Universal Serial Bus

SUMÁRIO

1 INTRODUÇÃO

O setor energético Brasileiro teve aumento exponencial no consumo de energia elétrica nos últimos anos, onde o consumo aumentou 3,6% em 2013 chegando a 530 TWh (Terawatts-hora). Até 2022 a previsão é que essa medida chegue a 785 TWh. Uma solução proposta, levando em consideração o aumento do consumo energético brasileiro, é a expansão e construção de novas termoelétricas e hidroelétricas. Pensando nesses dados, está planejada a construção de mais de 80 usinas até 2022 (MME, 2013).

O aumento da produção de energia elétrica consequentemente gera diversos danos à natureza, sendo que os principais impactos gerados são os ambientais, como desflorestamento e a poluição, causados pela construção de novas fontes e a queima de combustíveis fósseis. Para diminuir esses efeitos muitos estudos estão sendo feitos com foco em formas de obtenção da energia e no uso eficiente da energia produzida (INATOMI, 2000).

O investimento em maior produção de energia renovável, intensificando a oferta, que por consequência teria um aumento momentâneo na disponibilidade de energia elétrica, está longe de ser a melhor solução para amenizar o problema. É importante haver uma transformação no modo de consumir energia, caso contrário o risco ambiental irá persistir. Se países em desenvolvimento (com grande quantidade populacional e que estão começando a ter acesso a bens de consumo) começarem a consumir energia com os mesmos padrões de países ricos, o atual cenário de danos à natureza e aumento do consumo ampliará os problemas ambientais (ADELLE; MARC; PALLEMAERTS, 2009).

Pensando na necessidade de maior processamento com maior eficiência energética em plataformas computacionais, se torna essencial maximizar a utilização da energia, e para isso, são desenvolvidas formas de monitoramento e medição do consumo energético (FILHO, 2011).

A medição e o monitoramento do consumo energético pode ser feito através de várias técnicas, sendo que as mais conhecidas são: medir o consumo de forma teórica, medir com uso de um multímetro, ou medir utilizando um medidor de potência elétrica.

Poderia ser medido a potência utilizando um multímetro, onde são obtidos a tensão e a corrente,

podendo ser calculada a potência atual. Caso esta mantenha os mesmos valores durante o tempo, bastaria multiplicar a potência pelo tempo, mas como normalmente há variações, essa medição necessita ser feita em pequenos intervalos de tempo, não sendo uma opção viável (BRAGA, 2013). Devido à baixa amostragem, o multímetro não é uma ferramenta muito indicada para medir o consumo, podendo ser utilizado para uma comparação se o valor da potência está aproximado do valor obtido através de outras formas (ALMEIDA, 2011).

Atualmente, diferente de medir o tempo em uma aplicação, que é algo fácil, medir o consumo real de energia direto na aplicação é uma tarefa complicada. Faltam bibliotecas específicas que facilitem a obtenção de dados do consumo e componentes dedicados a efetuar tais medições.

Este trabalho tem como objetivo o desenvolvimento de uma biblioteca para medição do consumo energético multinível em plataformas de hardware modular, possibilitando uma análise do consumo em diferentes níveis de uma plataforma computacional, tornando a medição dos dados de energia tão simples quanto medir o tempo.

Para poder realizar esse trabalho será utilizada uma placa Odroid XU+E rodando um sistema operacional derivado do Linux, como exemplo de hardware modular. Ligado ao Odroid por USB estará um Arduino Uno, conectado ao Arduino estará uma placa do projeto OpenEnergyMonitor, a EmonTx Shield V2, para facilitar a obtenção dos dados pelos sensores. Também conectado ao Arduino terá um sensor de efeito hall, o ASC712, e conectado à placa EmonTx terá um sensor de corrente não invasivo e um transformador de tensão.

A biblioteca implementada será executada no sistema operacional da placa Odroid. Através deste terá acesso a todos os sensores que serão configurados pelo usuário, sendo que a biblioteca obterá os dados de tensão e corrente dos sensores realizando o cálculo de consumo instantâneo e também o consumo dentro do tempo desejado. A biblioteca terá alguns parâmetros para sua inicialização e após ser inicializada estará executando em uma *thread* separada, possibilitando utilizar suas outras funções ao mesmo tempo que ela realiza coleta dos dados e cálculos internos.

Para realizar a validação da biblioteca foi utilizado multímetro, realizando todas as comparações possíveis com os dados obtidos pelos sensores, tanto de tensão quanto de corrente. Ao total serão realizados quatro diferentes ensaios, obtendo o consumo em diferentes níveis da plataforma computacional. Os ensaios que serão realizados são: consumo energético com o processamento da placa Odroid totalmente ocioso, consumo desta com o processamento em 50% do total, consumo desta com processamento em 100% e o seu consumo durante a execução de um *benchmark*.

Dentro dos resultados obtidos neste trabalho pode-se observar alguns detalhes muito

importantes para um menor consumo de energia elétrica. Em particular, observou-se que a fonte transformadora de energia alternada para energia continua é a principal consumidora de energia elétrica, tendo um consumo maior que 60% do consumo total. Mesmo a placa estando com seu processamento totalmente ocioso, o que em teoria diminuiria seu consumo, o mesmo não vale para a fonte que mantém um alto consumo mesmo sem demanda de processamento interno. Por sua vez o consumo no processador enquanto ocioso foi totalmente satisfatório, utilizando apenas o necessário para se manter ligado. Por exemplo, durante a medição de cinco minutos ele não chegou a utilizar 5 W; em comparação, a fonte utilizou 2910 W nesse mesmo intervalo de tempo.

Foi possível verificar analisando todos os ensaios realizados que o aumento do consumo foi relativamente proporcional ao aumento do processamento, nos sensores integrados na placa e nos sensores entre a placa e a fonte.

Dentro do contexto apresentado anteriormente, no Capítulo 2 será descrito um referencial teórico destinado a apresentar os conceitos fundamentais para o desenvolvimento e aplicação deste trabalho.

O Capítulo 3 apresenta a descrição do desenvolvimento do trabalho, descrevendo a organização e montagem do hardware, assim como os principais detalhes de configuração do software e implementação da biblioteca. A seguir, no Capítulo 4 são descritos os ensaios feitos para validar o sistema implementado, assim como discussões sobre os resultados obtidos. Finalmente, no Capítulo 5, são apresentadas as conclusões do presente trabalho, destacando as principais contribuições e sugerindo possibilidades de continuidade para o mesmo.

2 REFERENCIAL TEÓRICO

Esta seção apresenta uma análise detalhada, identificando técnicas, procedimentos, e informações relevantes relacionados com a área de atuação deste projeto. Ao final do capitulo serão também descritos trabalhos relacionados a esta monografia.

2.1 Plataformas Modulares

As mudanças no cenário internacional caracterizado pelo menor ciclo de vida dos produtos, pelo rápido desenvolvimento tecnológico e pela globalização dos mercados, estão definindo uma nova abordagem nos projetos de desenvolvimento de produtos (PIETRO; AUGUSTO; MIGUEL, 2011).

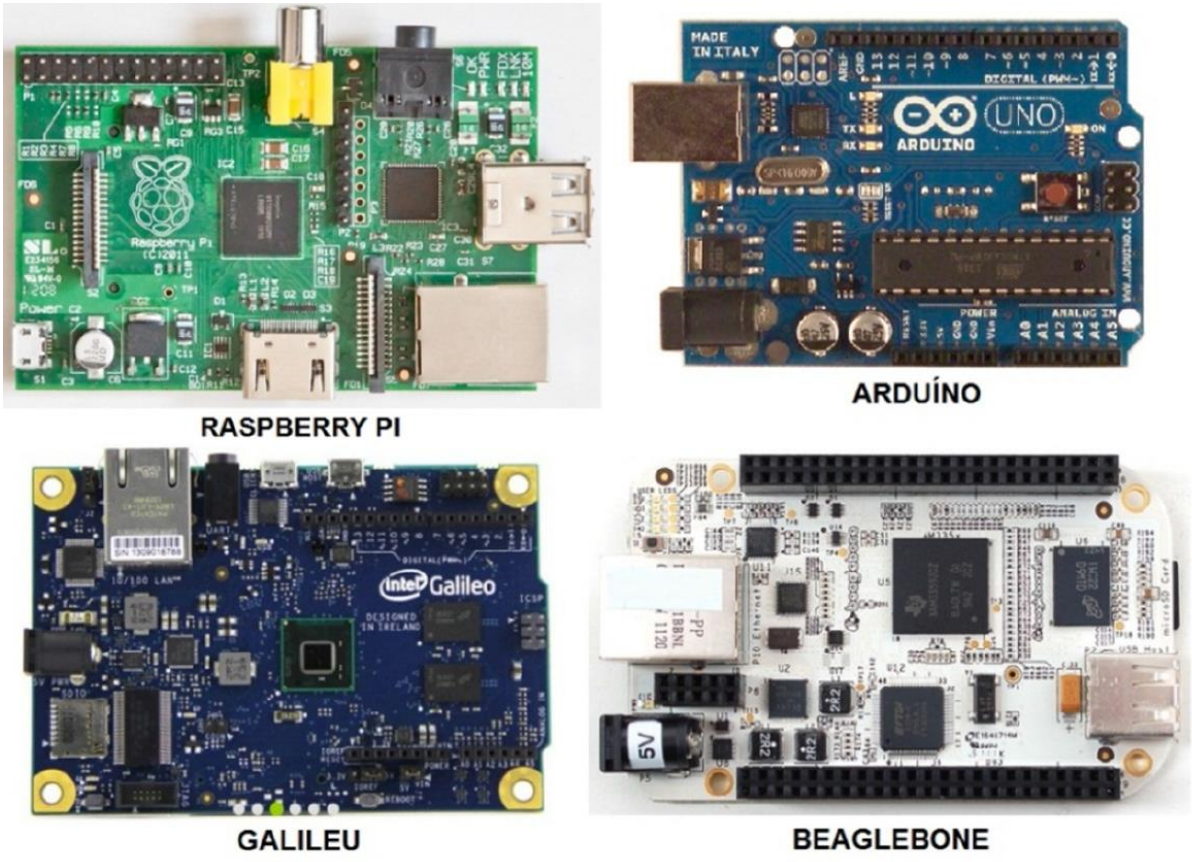
Segundo Régis (2014) uma plataforma modular é um projeto de desenvolvimento genérico, ou seja, que se adapte a várias aplicações sendo um dispositivo versátil. Esse tipo de plataforma é composto pelo módulo principal (modulo de controle), e módulos secundários que possuem características especificas, o que permite que o projeto seja configurado de acordo com suas necessidades.

As plataformas modulares possibilitam a criação de diversos projetos eletrônicos para o aprendizado ou para desenvolver protótipos. Elas também possibilitam um rápido desenvolvimento de produtos.

Um protótipo é a implementação de um projeto onde são estabelecidas as primeiras funcionalidades. A plataforma de prototipagem é o ambiente onde os protótipos são testados, que é constituída por vários módulos. Estes módulos combinados entre si aumentam o potencial da plataforma, permitindo um desenvolvimento mais rápido e eficiente, sendo possível a qualquer momento adicionar ou remover um módulo sem comprometer a plataforma (HULLER, 2014).

Existem diversas plataformas de prototipagem, sendo que as mais populares são: Galileo, Beaglebone, Raspberry Pi e Arduino. Essas plataformas estão apresentadas na Figura 1.

Figura 1 - Plataformas de prototipagem



Fonte: Elaborado pelo autor.

2.2 Instrumentação

A ciência que estuda, desenvolve e aplica instrumentos de medição e controle de processos é definida por Instrumentação.

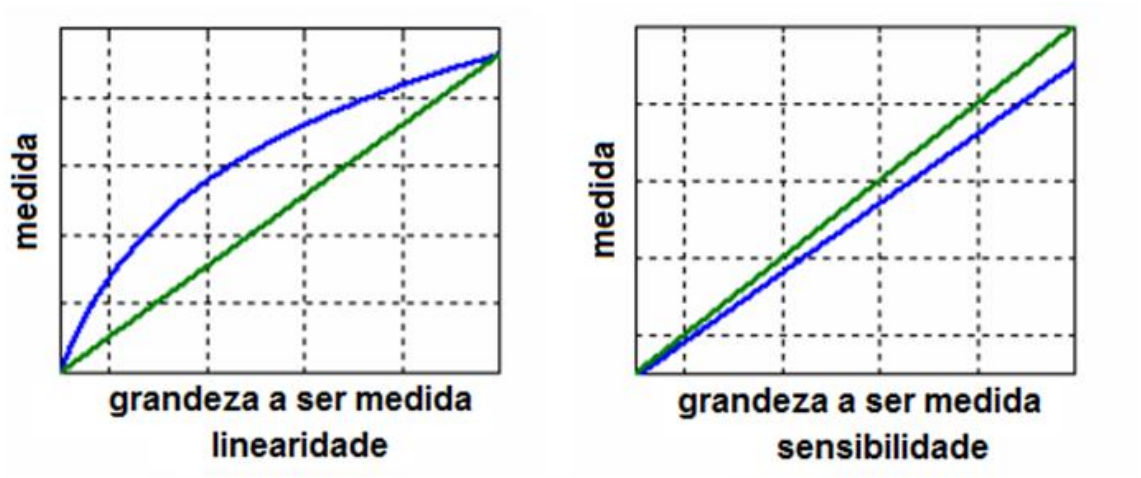
Na maioria dos processos automatizados é necessária uma realimentação do processo para que o dispositivo de controle conheça a saída do sistema. É justamente a instrumentação que proporciona esse processo de realimentação. Como existem inúmeros processos envolvendo diversas grandezas físicas que podem ser monitoradas, foram desenvolvidos distintos instrumentos de medição. Seja na indústria, na área médica, em pontos comerciais ou na área de transportes, as informações quanto ao andamento dos processos devem estar sempre disponíveis para serem controladas. Essas informações são coletadas por meio dos mais diversos tipos de sensores existentes (THOMAZINI; ALBUQUERQUE, 2007).

Os instrumentos de medição têm uma série de características importantes que devem ser

analisadas. Em particular, elas descrevem o desvio do comportamento real em relação ao comportamento ideal do sensor. As características normalmente são informadas no *data-sheet* do sensor. Exemplificamos algumas dessas características a seguir, segundo (BEGA, 2011).

- **Faixa de medida (*range*)**: É definida como faixa ou conjunto de valores da variável medida. Inclui todos os níveis de amplitude da grandeza física medida nos quais se supõe que o sensor pode operar dentro da precisão especificada. Por exemplo, um sensor de corrente pode ser fabricado para operar de 0A até 30 A.
- **Alcance (*span*):** É a diferença algébrica entre o valor superior e inferior da faixa de medição do instrumento. Por exemplo, um instrumento com *range* de 10°C a 50°C possui um *span* igual a 40°C.
- **Sensibilidade**: A sensibilidade é a relação entre a variação do sinal elétrico entregue na saída e a variação da grandeza física medida, como exemplifica a Figura 2.
- **Exatidão ou erro**: A exatidão ou erro é a diferença absoluta entre o valor do sinal de saída entregue pelo sensor e o valor do sinal ideal que o sensor deveria fornecer.
- **Precisão**: É o erro relativo máximo que um instrumento possa ter ao longo de sua faixa de medição, quando utilizado em condições normais de serviço. Todo sensor possui um erro de leitura, podendo ser maior ou menor conforme o tipo de grandeza medida, as condições de operação do sensor, o estado de conservação e a sua qualidade.
- **Linearidade**: Tendo um determinado sensor como referência, a grandeza física medida podem incluir variações no sinal entregue. Se o comportamento do sensor for ideal, o gráfico obtido é representado por uma reta, como ilustra a Figura 2.

Figura 2 - Linearidade e sensibilidade



Fonte: (BEGA, 2011).

2.3 Como medir consumo?

O resultado da medição do consumo de energia em um equipamento é obtido através da integral da potência (*P*) deste equipamento. Potência é a taxa de transferência instantânea de energia entregue a esse equipamento. De acordo com a Lei de Ohm, a potência é definida como um produto da corrente (*I*) pela tensão (*V*). Sua alimentação geralmente é medida em Watts (*W*), enquanto a corrente é medida em amperes (*A*) e tensão é medida em volts (*V*). Desse modo, uma maneira de obter a potência é medir corrente e tensão, como mostra a Equação 1. A Lei de Ohm também afirma que a corrente depende também da resistência, de modo que a corrente pode ser calculada ligando um resistor com valores conhecidos ao circuito, usando a Equação 2, determinado a corrente atual (IRWIN, 2013).

$$P = V \times I \tag{1}$$

$$I = \frac{V}{R} \tag{2}$$

Quando temos tensão e corrente podemos calcular a potência instantânea. Sendo assim, a potência é a quantidade de energia por unidade de tempo (joules/s). Então o resultado da medição do consumo de energia pode ser obtido através da integral da potência em relação ao tempo (Equação 3) (LEMOS, 2014).

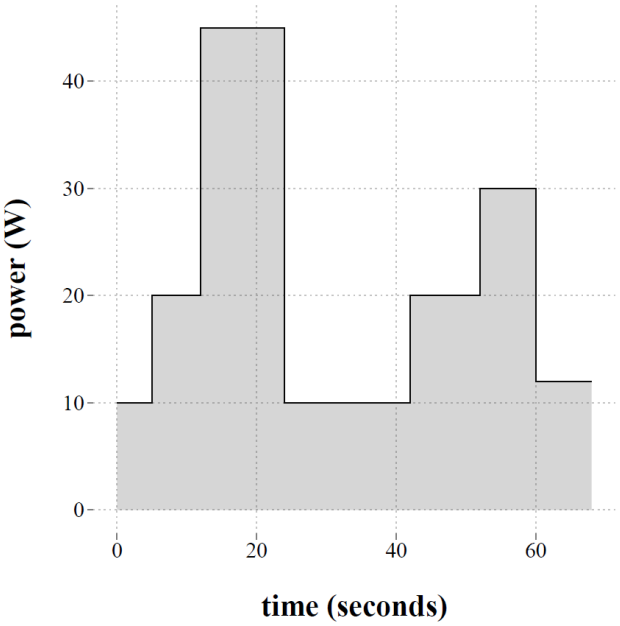
$$E = \int P(t)dt \tag{3}$$

Quando a potência for variável, é necessário fazer a média da potência, multiplicada pelo tempo que está esteve presente. Se ela for constante, o cálculo se resume à potência multiplicada pelo tempo que esteve presente. Desse modo, para obter o consumo de energia é necessário saber a potência do equipamento e o tempo que este esteve consumindo energia (GUIMARÃES, 2011). Mas na prática, devido à obtenção de consumo, que é feita entre pequenos intervalos, é usado o tempo em relação ao somatório da integral da potência e amostras de tempo, como ilustra a Equação 4, em que para cada amostra *I* existe uma potência instantânea e um intervalo de tempo.

$$E = \sum_{i=1}^{N} P(i).\Delta t(i) \tag{4}$$

Na Figura 3 é mostrado uma possível variação de dados obtidos em relação ao tempo.

Figura 3 - Consumo pelo tempo.



Fonte: (ROSTIROLA, 2015).

A medição do consumo elétrico pode ser feita em diversos locais de uma aplicação, para após poder ser comparada à divergência dos valores obtidos. No caso, os principais locais onde a medição pode ser feita são entre a tomada elétrica e a fonte de alimentação, entre a fonte e a placa mãe, ou entre componentes diferentes da aplicação (KRITIKAKOS, 2011).

Outra forma de medir consumo é por sensores ativados por softwares instalados no sistema operacional, assim as medições podem ser feitas em componentes individuais de uma aplicação. Este método permite medir o consumo individual do processador e compará-lo com outra parte da aplicação (KRITIKAKOS, 2011).

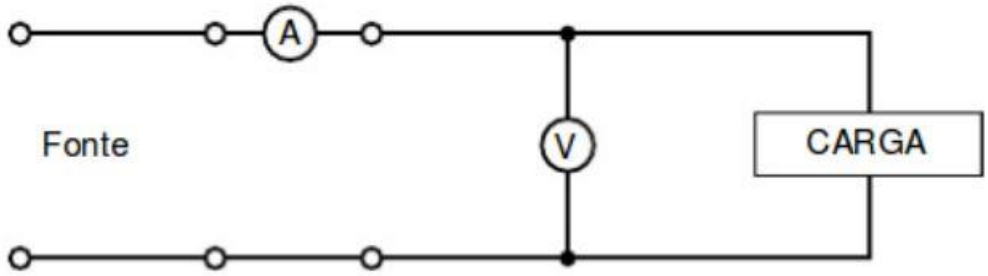
2.4 Sensores

Sensores são dispositivos instalados em determinados processos, que realizam a comunicação entre o sistema físico e o sistema de controle eletrônico. Ao detectar um evento nesse processo, retornam um sinal de resposta a este evento (MARTINS, 2008).

2.4.1 Sensores internos e externos

Existem algumas formas de realizar a medição de grandezas elétricas como corrente elétrica e tensão elétrica. Após retornar esse sinal de resposta, os tipos mais comuns são uma ligação em paralelo ou uma ligação em serie com a carga. Para fazer a medida da tensão elétrica, deve ser efetuada uma ligação em paralelo com a carga. Já para medir a outra grandeza informada anteriormente, a corrente elétrica, necessitamos abrir o circuito e realizar a ligação em serie com a carga (ALBIERO, 2010). Este modo de operação pode ser visto na Figura 4.

Figura 4 - Modo operação amperímetro e voltímetro



Fonte: (ALBIERO, 2010).

2.4.2 Contadores em hardware

Um recurso existente na maioria dos processadores (CPUs) atuais no mercado é a capacidade de contar determinados eventos ocorridos, denominados por contadores em hardware. Neste recurso existem registradores internos que trazem informações detalhadas sobre o estado do processador (CORRÊA, 2000).

Os eventos disponíveis para monitoramento dependem muito da plataforma computacional, entretanto há eventos comuns na maioria das CPUs, como: contagens de operações de ponto flutuante (FLOPs do Inglês Floating Point Operations) ou de inteiros, total de instruções executadas, além do registro das atividades ocorridas nas memórias *cache*. O monitoramento de novos eventos é adicionado conforme novas versões de CPUs são lançadas (CORRÊA, 2000).

A Tabela 1 mostra os eventos mais comuns disponíveis nas plataformas computacionais.

Tabela 1 - Eventos mais comuns disponíveis em algumas plataformas.

Evento	Pentium	IBM 604	SUN Ultra II	Cyrix 6x86MX	HP PA RISC	Alpha
Ciclos	X	X	X	X	X	X
Instruções	X	X	X	X	X	X
Loads	X	X	X	X	-	X
Stores	X	X	X	X	-	X
L1 misses	X	X	X	X	X	X
L2 misses	X	X	X	-	-	X
FLOPs	X	X	-	X	-	X
Condicionais	X	X	-	X	-	X
TLB misses	X	X	-	X	X	X
Code cache misses	X	X	X	X	-	X

Fonte: (CORRÊA, 2000).

Os Registradores Específicos de Modelo, ou MSRs (Model-Specific Registers), são registradores que foram projetados para facilitar a depuração de aplicações, permitindo a configuração da CPU e o acompanhamento de seu desempenho. Estes foram adicionados inicialmente pela Intel na

microarquitetura Pentium.

Com a sua aceitação pelos consumidores e a possível preocupação do consumo energético, novos contadores em hardware foram projetados, em particular um grupo de MSR adicionado a partir da arquitetura Sandy Bridge para auxiliar no monitoramento do consumo energético. Esse grupo de registradores está referenciado no manual do desenvolvedor por RALF (Running Average Power Limit), em que sua interface é constituída por contadores de consumo de energia, dissipação de potência e de tempo (CARVALHO ALEXANDRO BALDASSIN, 2013).

Sensores dentro do processador fornecem a medida de consumo do processador, contudo não mostrando o consumo de outros componentes. Vale ressaltar que, desta forma, não representa o consumo real de energia em um todo (CARVALHO ALEXANDRO BALDASSIN, 2013).

2.4.3 Sensores conectados ao hardware

Em diversos equipamentos são utilizados sensores integrados às placas para monitorar suas funcionalidades, sendo que a grande maioria das placas-mãe presentes no mercado já apresentam sensores integrados. Em geral, possibilitam a medição da temperatura em diversas partes do hardware, podendo ser definindo quando o sistema de resfriamento deve ser ligado ou desligado para economia de energia (JR, 2012).

Os *smartphones*, através dos mais variados tipos de sensores integrados em seu hardware, são capazes de gerar informações com grande precisão, promovendo uma melhor experiência para seus utilizadores. A maioria dos *smartphones* têm pelo menos três sensores, permitindo saber sua localização ou também identificar se o aparelho está em queda (DUARTE, 2013).

2.4.4 Sensores na alimentação e na fonte

Na alimentação do sistema, antes da fonte, como também após a fonte, normalmente é necessário utilizar sensores externos. Então, para esse tipo de medição pode ser utilizado um sensor de corrente não invasivo, como mostra a Figura 5. Esse sensor será detalhado e explicado na seção 2.4.6.

Figura 5 - Sensor de corrente não invasivo

Fonte: (NERY, 2013).

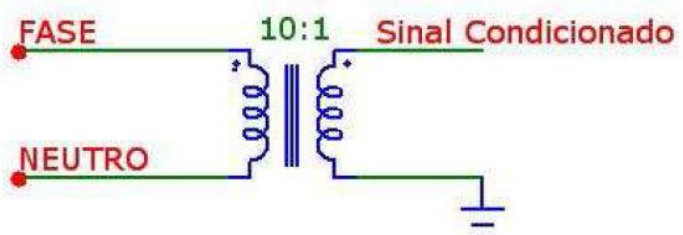
Diferente dos sensores internos, que em alguns casos já são integrados em placas de controle de sistemas, as fontes de energia necessitam de sensores externos para se obter as medições de corrente e tensão após a fonte. Para realizar essa mediação entre a fonte e a aplicação, pode ser utilizado multímetros ou *power meters* (KRITIKAKOS, 2011).

2.4.5 Sensor de tensão

Na grande maioria dos circuitos não é possível simplesmente colocar um sinal de tensão diretamente, devido à incompatibilidade dos níveis de tensão do sinal e do sistema que irá tratar os seus dados. Levando essa incompatibilidade em questão é necessário um sensor capaz de padronizar os valores obtidos em valores que serão aceitos pelo sistema de processamento (WHITAKER, 2008). A técnica usada para fazer esta padronização será o Transformador de potencial.

O transformador de potencial mostrado na Figura 6, é utilizado para redução de grandezas elétricas como corrente ou tensão. É um transformador em que o primário é ligado em derivação com o circuito a ser medido, em que o secundário é ligado no circuito de medição, tendo sua padronização feita ajustando a relação de espiras do primário com a do secundário. Por conta da tensão em seu circuito secundário ser normalmente menor que a tensão no primário, ele é considerado um redutor de tensão (TEIXEIRA, 2009).

Figura 6 - Transformador de potencial



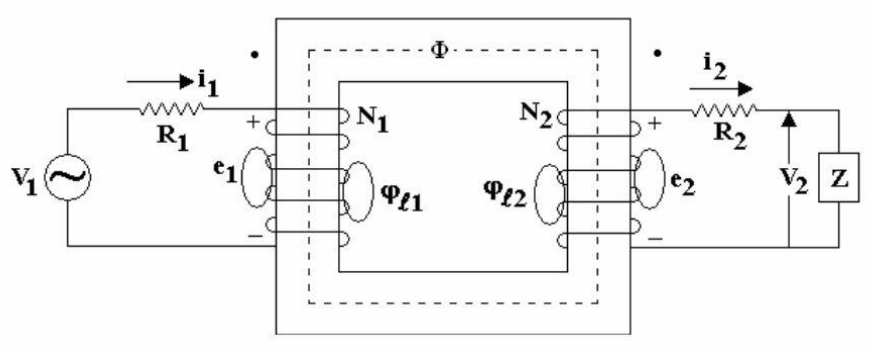
Fonte: (TEIXEIRA, 2009).

2.4.6 Sensor de Corrente

Como os sensores de tensão, os sensores de corrente necessitam que seja adequado o valor de corrente para um padrão compatível de ser interpretado pelo sistema de tratamento de dados. Existem muitas técnicas utilizadas para montar este tipo de sensor, descritas a seguir (TEIXEIRA, 2009).

- Transformador de corrente: Figura 7 é um dispositivo que transforma a corrente de um enrolamento primário, usando como princípio de funcionamento a indução eletromagnética, gerando uma corrente no secundário. Se o primário for conectado a uma fonte de tensão alternada, será produzida uma corrente que dependera da tensão do primário e do número de esfirras. A corrente que é induzida para o secundário dependerá do número de esfirras dele. O transformador não é capaz de converter energia, mas sim, transferir de um circuito para outro (CHAVES, 2008). O transformador da Figura 7 é um núcleo magnético com dois enrolamentos individuais, ligados ao fluxo magnético.

Figura 7 - Diagrama esquemático de um transformador de corrente



Fonte: (CHAVES, 2008).

As variáveis mostradas na Figura 7 têm as seguintes descrições:

V1 - Tensão no primário;

i1 - Corrente no primário;

R1 - Resistência do enrolamento primário;

e1 - Tensão induzida no enrolamento primário;

V2 - Tensão no secundário;

i2 - Corrente no secundário;

R2 - Resistência do enrolamento secundário;

e2 - Tensão induzida no enrolamento secundário;

N1 - Número de espiras do primário;

N2 - Número de espiras do secundário;

φ - Fluxo responsável pela transferência de potência do primário para o secundário;

φ_1 - Fluxo de dispersão do enrolamento primário;

φ_2 - Fluxo de dispersão do enrolamento secundário.

- Sensor de corrente não invasivo: Figura 5, eles são sensores utilizados para medir corrente alternada. Este sensor possui suporte a uma corrente de no máximo 100 Amperes e trabalha em temperaturas entre -25ºC e +70ºC. O sensor pode ser fixado ao redor de uma linha de alimentação para dizer quanta corrente está passando através dela. Ele atua como um indutor e responde ao campo magnético. Pela leitura da quantidade de

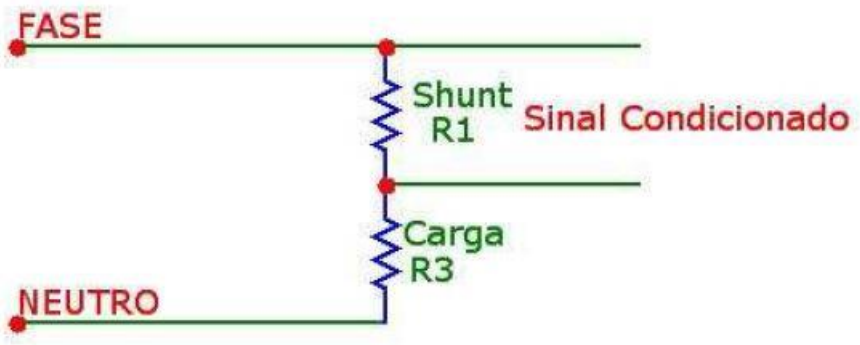
corrente produzida pela bobina você pode calcular a corrente que está passando pelo condutor (NERY, 2013).

- O Resistor Shunt: Figura 8, o processo mais utilizado para medição de corrente elétrica, é introduzindo uma resistência de valor conhecido e baixo em série com o circuito. A intensidade de corrente é obtida usando a lei de Ohm (Equação 5) tendo de base a queda de tensão medida na resistência. O problema desse método é o aquecimento do resistor no caso de grandes valores de corrente e falta de uma proteção para o sistema que realiza a leitura da tensão do resistor (TEIXEIRA, 2009).

$$V(t) = R \times I(t) \qquad (5)$$

Onde, *V(t)* é a tensão medida, *I(t)* é a corrente elétrica e o valor da resistência *R* é dados em Ohms.

Figura 8 - Resistor shunt



Fonte: (TEIXEIRA, 2009).

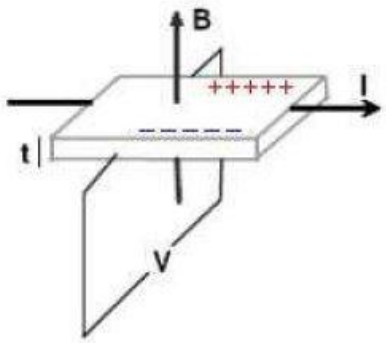
- Sensor de efeito Hall: Figura 9. O elemento mais básico do sensor de efeito Hall é pastilha de um material condutor que, ao se aplicar uma fonte de tensão uma corrente passará por ele e estando posicionado em uma região do campo magnético, consegue medir a variação do fluxo. Com a existência de uma densidade de fluxo magnético *B* formando um angulo reto de 90º à superfície de uma pastilha de espessura *t*, uma diferença de potencial *V* é induzida pela força de Lorentz. A expressão que determina o

valor de tensão é mostrada na Equação 6.

$$V = \frac{RH \times I \times B}{T} \quad (6)$$

RH é o coeficiente de Hall, e é constante para um material e uma temperatura fixa, *I* corrente que atravessa a pastilha, *T* temperatura da operação, *V* tensão produzida nas laterais da pastilha e *B* é a densidade do fluxo magnético.

Figura 9 - Sensor de efeito hall



Fonte: (TEIXEIRA, 2009).

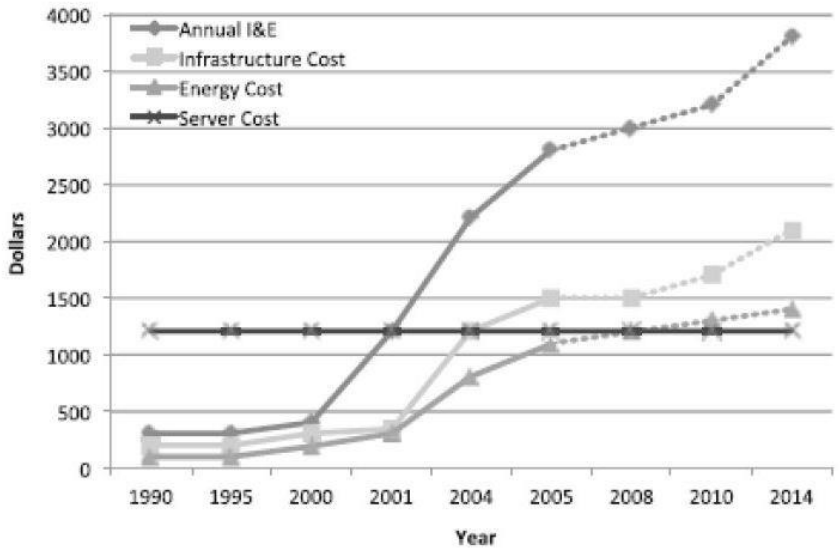
2.5 Trabalhos relacionados

Com o atual cenário energético, muitos estudos estão sendo feitos para melhorar a utilização da energia e diminuir o consumo energético. A seguir serão detalhados alguns estudos com referência a consumo energético.

Com o objetivo de fabricantes desenvolverem computadores com grande capacidade de processamento, pensando apenas no desempenho, chegou a um momento em que servidores consumiam grandes quantidades de energia, tendo um custo elevado com a refrigeração. Em 2001 o custo anual com infraestrutura e energia de um servidor era inferior que o valor deste servidor. A partir de 2004 o custo da infraestrutura por si só, igualou o custo do servidor, e até 2008 o custo apenas da energia anual para manter o servidor funcionando, igualou ao custo de sua aquisição Figura 10. Em 2007 Wun-chun Feng, incentivando uma competição por servidores mais eficientes e mostrando o

problema de autoconsumo energético, propôs uma nova métrica para a classificação de servidores, o Green500, que levava em conta o número de operações de ponto flutuantes por Watt (flops/watt) (FENG, 2012).

Figura 10 - Custo de infraestrutura, energia e do servidor em relação ao tempo



Fonte: (FENG, 2012).

O Green500 apresenta informações relacionadas à arquitetura das maquinas, consumo, desempenho e sua localização. A lista publicada em junho de 2013 é liderada por dois sistemas que utilizam processadores Xeon E5-2687W, baseados em aceleradores NVIDIA Kepler K20, com desempenho de 3.208,83 e 3.179,88 Mflops/Watt para primeira e segunda colocada respectivamente (ROSTIROLLA, 2014). A Tabela 2 - Top 10 Supercomputadores do Green500 apresenta uma lista com os dez primeiros colocados no Green500.

Tabela 2 - Top 10 Supercomputadores do Green500

Nome	Ranking	Eficiência		
	Green500	Rmax	Rpico	Mflops/Watt
Eurora	1	98511	175667	3208,827362
Aurora Tigon	2	98640	167782	3179,883946

Nome	Ranking	Eficiência		
	Green500	Rmax	Rpico	Mflops/Watt
Beacon	3	110500	157547,52	2449,567723
SANAM	4	421200	1098000	2351,102428
BlueGene/Q, Power BQC	5	188967	209715	2299,148315
Cetus	6	188967	209715	2299,148315
CADMOS BG/Q	7	188967	209715	2299,148315
BlueGene/Q, Power BQC	8	188967	209715	2299,148315
Vesta	9	188967	209715	2299,148315
BlueGene/Q, Power BQC	10	188967	209715	2299,148315

Fonte: (ROSTIROLLA, 2014).

A corrida dos supercomputadores para obtenção de maior desempenho desconsidera o alto consumo energético destas máquinas. Em relação a isso, é relevante citar um trabalho que analisa a possibilidade de supercomputadores serem desenvolvidos com computadores de placa única, tendo baixo consumo e alto processamento.

O trabalho em questão é descrito em (ROSTIROLLA, 2014). Seu principal objetivo foi realizar uma análise do desempenho e consumo energético de placas de desenvolvimento BeagleBone Black. Tais placas utilizam processadores ARM Cortex-A8, tendo seu maior foco no baixo consumo energético, o que é uma das apostas da computação de alto desempenho utilizando processadores ARM.

Figura 11 - Cluster beaglebone black

Fonte: (ROSTIROLLA, 2014)

Os testes realizados, tanto de desempenho como de medição do consumo, foram feitos em cima de um cluster homogêneo Single-board Computers (computadores de placa única), que foi montado utilizando 10 placas BeagleBone Black Figura 11, rodando o sistema operacional Linux, foram realizados testes sob diversas cargas de trabalho, através do uso do benchmark HPL (High Performance LINPACK), e do Ondes3D, fazendo sua comunicação através de troca de mensagens com MPI e utilizando uma placa de medição de consumo da plataforma LabVIEW (ROSTIROLLA, 2014).

Outro trabalho relevante é de um grupo de pesquisa da Universidade Federal de Sergipe (UFS) (SANTOS, 2011), que analisou quatro técnicas para medir o consumo de energia executando em plataformas computacionais. As quatro técnicas foram:

- Medição do nível de descarga da bateria para um notebook usando comandos do próprio Linux;
- Estimativa teórica com base nas características do processador, em um PC tipo *desktop*;
- Simulação de uma arquitetura usando a ferramenta Sim-Panalyzer;
- Medição do consumo real usando um osciloscópio.

Nos testes foram usados os algoritmos SHA e Blowfish do *benchmark* Mibench. A técnica que ofereceu maior precisão foi a experimental, pois consegue medir a real corrente utilizada pelo sistema enquanto executa uma aplicação.

Foi possível observar que apenas a movimentação do mouse, chega a causar um aumento de até

90% do consumo em relação ao sistema ocioso. A execução de qualquer instrução demanda uma quantidade de energia específica para essa tarefa. Isso exemplifica que o consumo de energia é um fator importante nos projetos de sistemas embarcados, e que seja cada vez mais importante a medida para redução do consumo de energia.

A primeira técnica foi executada em um notebook desenvolvido com tecnologias de baixo consumo, que resultaram no dispositivo com melhor relação entre desempenho e consumo de energia.

A segunda técnica foi totalmente teórica e a mais falha, pois os dados de consumo do processador e a quantidade de ciclos necessários para executar o algoritmo, são calculados com base teórica.

A terceira técnica apresenta o resultado da simulação em um relatório do consumo com os parâmetros usados, foi utilizado o processador com configurações padrão do Sim-Panalyzer apesar da arquitetura de menor potência utilizada, ele apresentou o pior desempenho entre as técnicas simuladas, pois o tempo de execução foi muito grande.

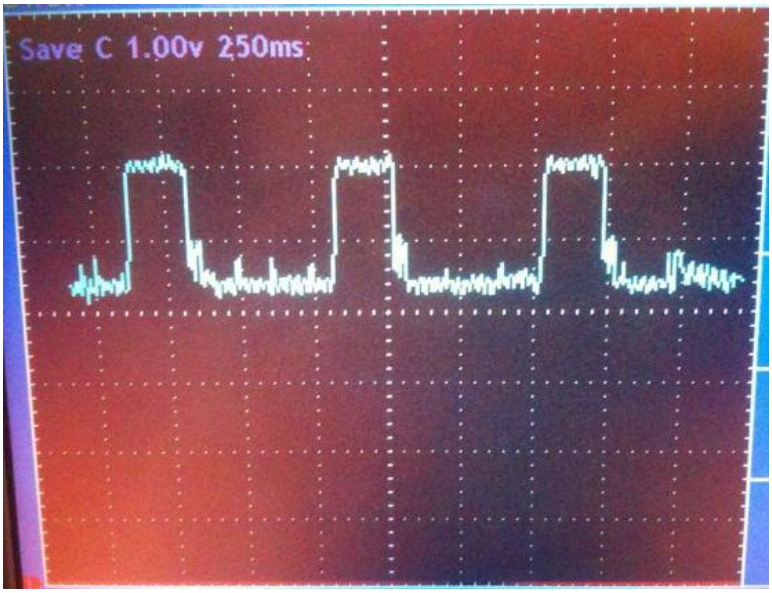
A quarta e última técnica empregada apresenta uma medição real do consumo de uma placa mãe contendo um processador de *desktop*. A medição foi realizada utilizando um osciloscópio e foram observados os picos de consumo de cada algoritmo e o tempo real de execução dos mesmos. Esta técnica foi a que teve os resultados próximos do consumo real do processador.

Foi observado que as arquiteturas desenvolvidas para baixo consumo apresentam resultados satisfatórios como pode ser observado com a medição da descarga da bateria.

Outra conclusão importante foi observada com o resultado da simulação por meio do simulador Sim-Panalyzer, onde o dispositivo simulado apresenta uma arquitetura simples e apresentando a menor potência entre os dispositivos testados, foi o que apresentou o maior consumo de energia, pois o tempo de execução foi o mais elevado.

Ao utilizar a medição do consumo com o osciloscópio é possível observar picos de consumo dentro do algoritmo, possibilitando analisar e encontrar os trechos de código responsáveis pelo alto consumo de energia (SANTOS, 2011), como pode ser visto na Figura 12.

Figura 12 – Picos de consumo



Fonte: (SANTOS, 2011).

3 DESENVOLVIMENTO

Esta seção tem o objetivo de descrever o funcionamento do módulo de medição de energia, suas configurações necessárias e quais equipamentos ou sensores foram utilizados no desenvolvimento deste trabalho. Na primeira parte é descrito o hardware utilizado e detalhadas as placas utilizadas, incluindo os métodos utilizados para fazer a medição fora da aplicação e dentro da aplicação. Após é feito um detalhamento da montagem física do projeto e como o hardware está conectado.

Também são descritos os softwares utilizados e o seu funcionamento. Em particular, é detalhada a obtenção do consumo e a utilização da biblioteca desenvolvida durante o projeto.

3.1 Visão geral do hardware

O hardware utilizado para medição é composto por uma placa Odroid-XU+E, uma placa EmonTx 2 Shield do projeto OpenEnergyMonitor (OEM), um Arduino Uno, um sensor de corrente não invasivo e um sensor de efeito Hall (ACS712).

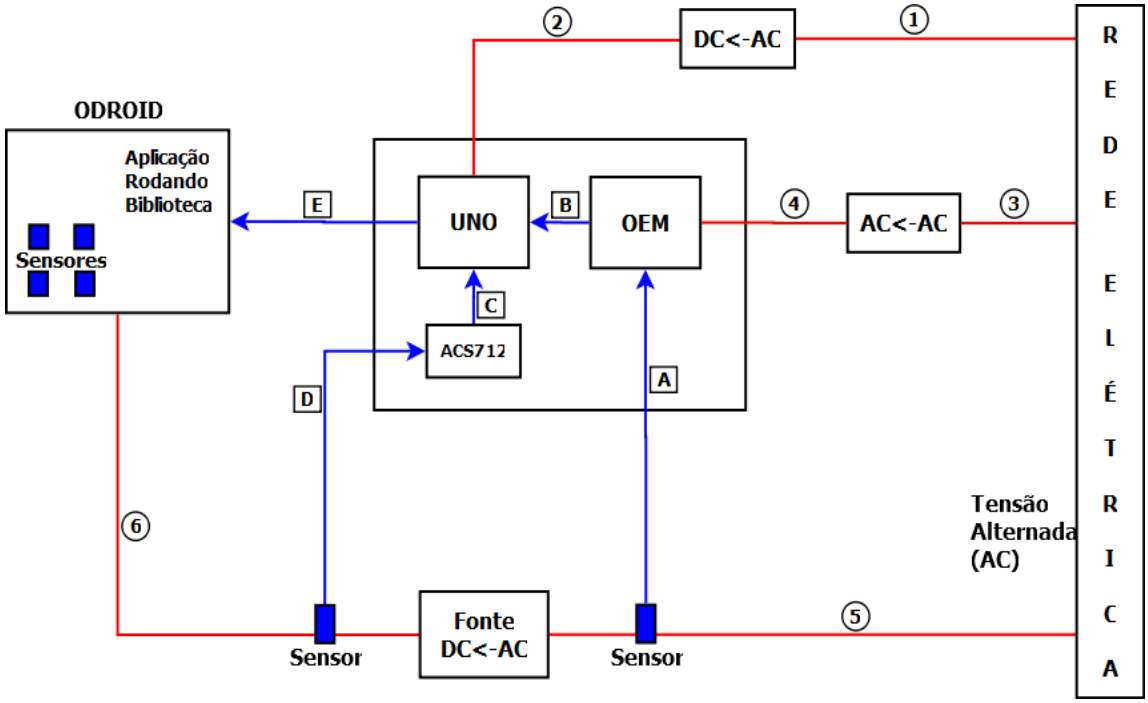
A Alimentação das placas como pode ser visto na Figura 13, está disposta da seguinte forma: o fio 1 está conectado à rede elétrica com tensão de 220V com corrente alternada (AC, do inglês Alternating Current) e em uma fonte conversora de corrente alternada para corrente contínua (DC, do inglês Direct Current). O conversor AC/DC está alimentando a placa Uno através do fio 2 utilizando uma tensão de 5V, as placas do OEM e do sensor ACS712 estão conectadas no Arduino Uno e sua alimentação é feito pelo fio 2 através da Placa Uno. É utilizado um transformador de tensão, 220V para 9V que é usado para medir tensão da rede elétrica em tempo real, este transformador AC/AC está ligado à rede pelo fio 3, e ele está conectado a placa do OEM pelo fio 4 que tem uma tensão de 9V. O fio 5 está conectado à rede elétrica em uma corrente alternada e a uma fonte transformadora de corrente alternada para corrente contínua, esta fonte realiza a alimentação do Odroid com corrente contínua e tensão de 5V através do fio 6.

A transferência dos dados das medições de consumo, como pode ser visto na Figura 13, se dá da

seguinte forma: o sensor conectado ao fio 5 passa os dados através da conexão para a placa do OEM, que após realizar a leitura dos dados repassa os dados para o Arduino Uno utilizando a conexão B. O sensor conectado ao fio 6 transfere os dados para a placa ACS712 pela conexão D, que repassa os dados através da conexão C para o Arduino. O Uno por sua vez faz o tratamento dos dados passados pelas duas placas anteriores e passa essas informações pela conexão E para o Odroid, o qual está rodando a biblioteca que foi desenvolvida e estará esperando essas informações.

Na Figura 13 pode ser visto um diagrama da montagem do hardware utilizado, sua alimentação e a transferência dos dados da medição.

Figura 13 – Diagrama hardware



Fonte: (Elaborado pelo autor).

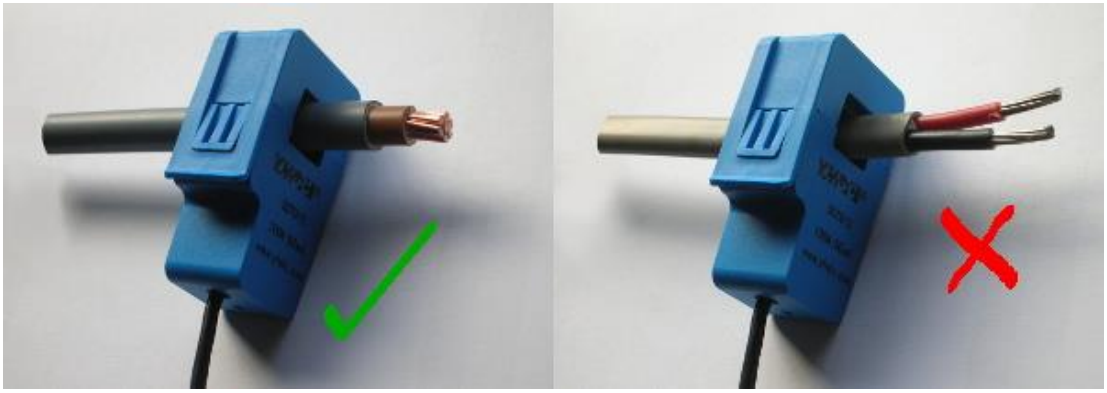
A medição do consumo é realizada em três locais diferentes, diretamente na placa Odroid, entre a rede elétrica (fonte de energia) e a fonte de alimentação da placa Odroid e entre a fonte de alimentação e o Odroid. Detalhando:

- Medição dentro da placa: No Odroid, são executadas as rotinas que obtém e trata os dados dos sensores, essa placa contém sensores de corrente e tensão conectados diretamente na placa por seu desenvolvedor, através desses sensores é obtido o consumo interno na placa, da CPU, Unidade de Processamento Gráfico (GPU do inglês Graphics

Processing Unit), e na Memória de Acesso Aleatório Dinâmica (DRAM do inglês Dynamic Random Access Memory), o primeiro tem 2 sensores um para cada grupo de 4 núcleos da CPU, sendo que a CPU possui 8 núcleos, os demais contem 1 sensor em cada, o detalhamento completo dessa placa está descrito na seção 3.2.

- Entre a placa e a fonte de alimentação: É utilizado o sensor de efeito *hall* de modelo ACS712.
- Entre a fonte de alimentação e a tomada: Para realizar essa medição está sendo utilizado o sensor de corrente não invasivo Figura 5, que está ligado a placa OEM, esse sensor é colocado ao redor de apenas um dos fios de transporte de corrente Figura . Caso o sensor seja ligado de forma errada, os valores obtidos serão somados se a corrente tiver o mesmo sentido. Se a corrente estiver em sentidos opostos os valores entre elas serão subtraídos, gerando assim uma saída zerada caso os valores sejam iguais, a medição da tensão da rede elétrica é feita através do transformador de tensão, 220V para 9V, ligado a placa do OEM.

Figura 14 – Utilização sensor de corrente não invasivo

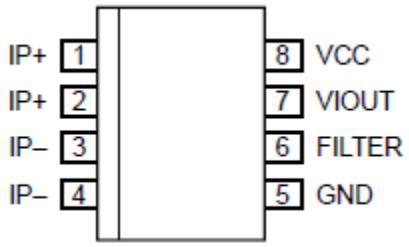


Fonte: (openenergymonitor.org).

No projeto será utilizado o sensor de efeito hall de modelo ACS712 desenvolvido pela Allegro MicroSystems. O ACS712 é um sensor linear de efeito hall, seu funcionamento consiste na aplicação de uma corrente entre os terminais IP+ (pinos 1 e 2) e IP- (pinos 3 e 4) representados na Figura 15Figura 14. A corrente, ao passar por estes terminais de cobre gera um campo eletromagnético, o qual é convertido para um valor de tensão proporcional. A resistência interna entre estes terminais é de 1,2 mΩ, proporcionando baixa perda de potência, além de possuir isolamento de 2,1kVRMS entre pinos 1 a 4 e os pinos 5 a 8, dispensando a utilização de componentes para isolação elétrica (ALLEGRO

MICROSYSTEMS ACS712, 2012).

Figura 15 - Sensor de efeito hall ACS712



Fonte: (Allegro MicroSystems, 2012).

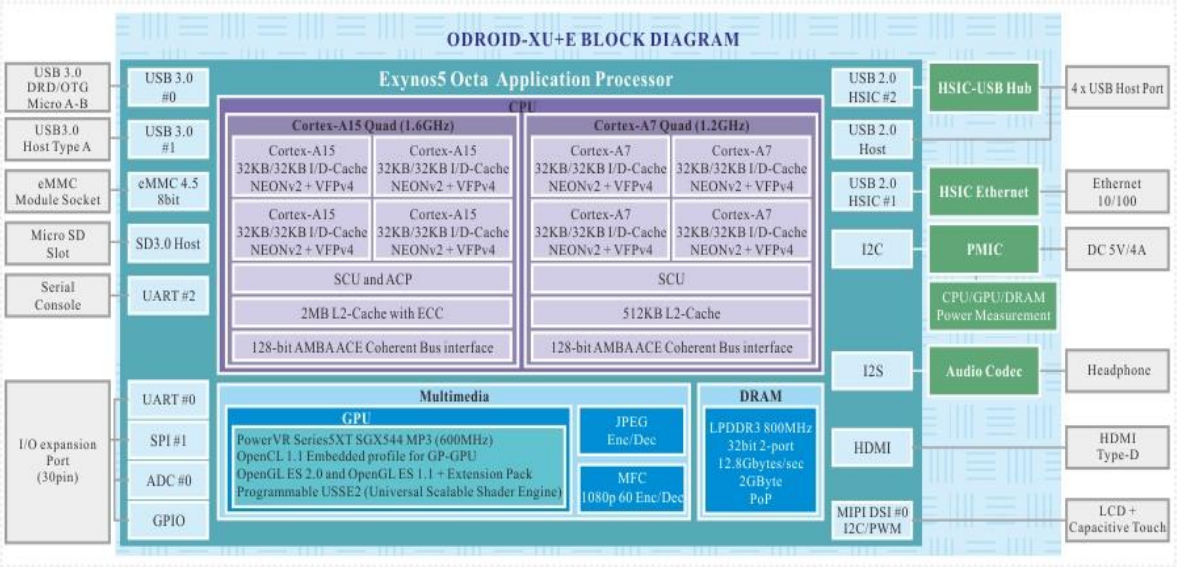
O circuito integrado (CI) utilizado foi o ACS712ELECTR-20A-T, que possibilita a medição de corrente contínua ou alternada, na faixa de -20 A a +20 A, gerando um valor de tensão no pino 7 (VIOUT) de 100 mV/A. Por funcionar também com corrente alternada, o valor inicial de tensão informado em sua saída é correspondente a metade da tensão de alimentação, ou seja, como o CI está sendo alimentado com uma tensão de 5 V, o valor inicial de saída é de 2,5 V.

3.2 Odroid

O ODROID XU+E é um computador de placa única com alto processamento e baixo consumo de energia, produzido empresa sul-coreana Hardkernel. Utiliza um processador projetado pela empresa ARM e fabricado pela Samsung, o Exynos5 Octa 5410. O Odroid foi o primeiro dispositivo de desenvolvimento lançado comercialmente que tem um processador de oito núcleos e o primeiro sistema-em-um-chip (SoC, do inglês System on Chip) do mercado implementando a estratégia big.LITTLE da ARM. Ele tem processadores Cortex™-A15 de 1.6 GHz mais 4 processadores Cortex™-A7 de 1.2 GHz. Todos os núcleos têm 32 KB de *cache* de dados e um *cache* de instruções do mesmo tamanho. O cluster A15 também tem 2 MB de *cache* L2 e trabalha numa frequência de 800 MHz a 1600 MHz. O A7 possui 512 KB de *cache* L2 e trabalha entre 500 MHz e 1200 MHz. Este processador foi implementado utilizando uma técnica onde apenas os núcleos A15 ou os núcleos A7 podem estar ativas ao mesmo tempo.

Essa placa possui ferramentas para análise de energia através de sensor integrados diretamente nela e conta ainda com 2GB de RAM e a GPU PowerVR SGX544MP3 (HARDKERNEL, 2013).

Figura 16 - Diagrama de bloco Odroid-XU + E



Fonte: (HARDKERNEL, 2013).

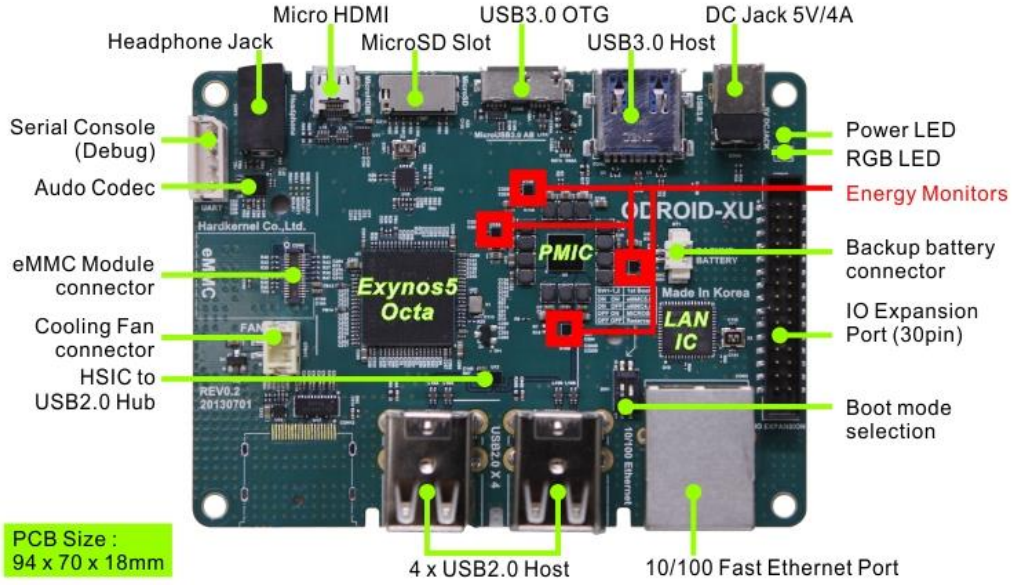
A estrutura da placa com suas especificações, será detalhada abaixo.

- Energy Monitors: Sensores de corrente e sensores de tensão.
- Processador: Samsung Exynos5 Octa ARM Cortex ™ -A15 1.6Ghz Quad e Cortex ™ -A7 1.2GHz Quad CPUs.
- Memória RAM: LPDDR3 2GByte PoP (800Mhz, 1600Mbps / pin, 2 x 32bit Bus).
- Accelerator 3D: PowerVR SGX544MP3 GPU (OpenGL ES 2.0, OpenGL ES 1.1 e OpenCL 1.1 EP).
- Vídeo: Suporta 1080p via cabo HDMI (formato contêiner MP4 H.264 + AAC).
- Video Out: conector HDMI micro.
- Audio: Codec de áudio on-board. 3,5 mm para auscultadores standard. HDMI digital.
- USB3.0 Host: Conector padrão SuperSpeed USB Um tipo x 1 porta.
- USB3.0 OTG: Conector do tipo Micro USB A-B SuperSpeed x 1 porta.
- USB2.0 host: Padrão de alta velocidade um conector do tipo x 4 portas
- Display: Monitor HDMI
- Armazenamento: Slot para cartão MicroSD (opcional). Soquete do módulo eMMC: eMMC 4,5, armazenamento flash (até a capacidade 64GByte e 160MByte velocidade de acesso / sec).
- Fast Ethernet: 10/100Mbps Ethernet com conector RJ-45 (suporte Auto-MDIX) Gigabit

Ethernet LAN (opção) USB3.0 para adaptador Gigabit. SuperSpeed USB (USB 3.0) para adaptador ATA3 Serial para 2.5 "/3.5" armazenamento HDD e SSD.

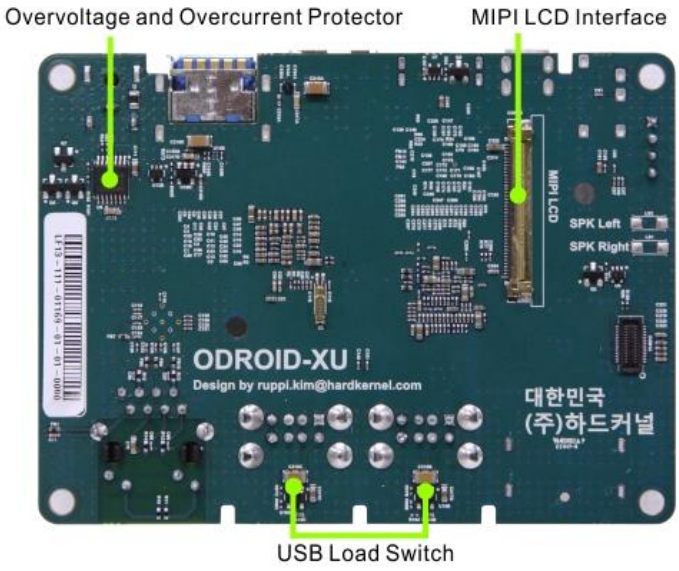
- Alimentação: 5V 4A potência.
- Refrigeração: Cooler (98 x 74 x 29 milímetros aprox.)

Figura 17 - Detalhes frontal da placa Odroid-XU + E



Fonte: (HARDKERNEL, 2013).

Figura 18 - Detalhes traseiros da placa Odroid-XU + E



Fonte: (HARDKERNEL, 2013).

3.2.1 Energy Monitors

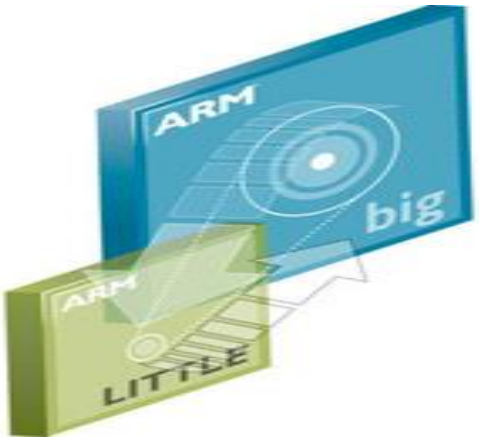
O Energy Monitors ou monitores de energia são sensores de corrente e sensores de tensão que foram implementadas no bloco de controle de processo (PCB, do inglês Proccess Control Block) da placa para facilitar a medição de energia. É medido o consumo de energia dos núcleos de processamento, GPU e SRAM individualmente.

O sensor implementado é o modelo INA231, desenvolvido pela Allegro MicroSystems e utiliza um resistor Shunt para realizar as medições. Este dispositivo fornece os valores atuais de tensão e corrente permitindo medidas com precisão (ALLEGRO MICROSYSTEMS INA231, 2013).

3.2.2 ARM big.LITTLE

O big.LITTLE não consiste no uso de oito núcleos da mesma arquitetura (Cortex A-15), mas sim no uso combinado de dois conjuntos de quatro núcleos de arquiteturas diferentes. O big é o conjunto de quatro núcleos Cortex™ A-15 reservado para tarefas de alto desempenho, consequentemente, seu consumo de energia é alto. Já o LITTLE, é o conjunto dos outros quatro núcleos Cortex™ A-7, que são extremamente eficientes no que diz respeito a consumo de energia, sendo destinado a tarefas de baixo consumo de recursos. A vantagem de conter uma parte do processador em um Cortex mais antigo, é o menor consumo de energia. Isso se assemelha ao que acontece em *notebooks* que são fabricados com uma placa gráfica integrada de baixo desempenho, ao lado de outra de alto desempenho, possibilitando que o sistema escolha a ser usada, levando em conta a sua necessidade.

Figura 19 - ARM big.LITTLE



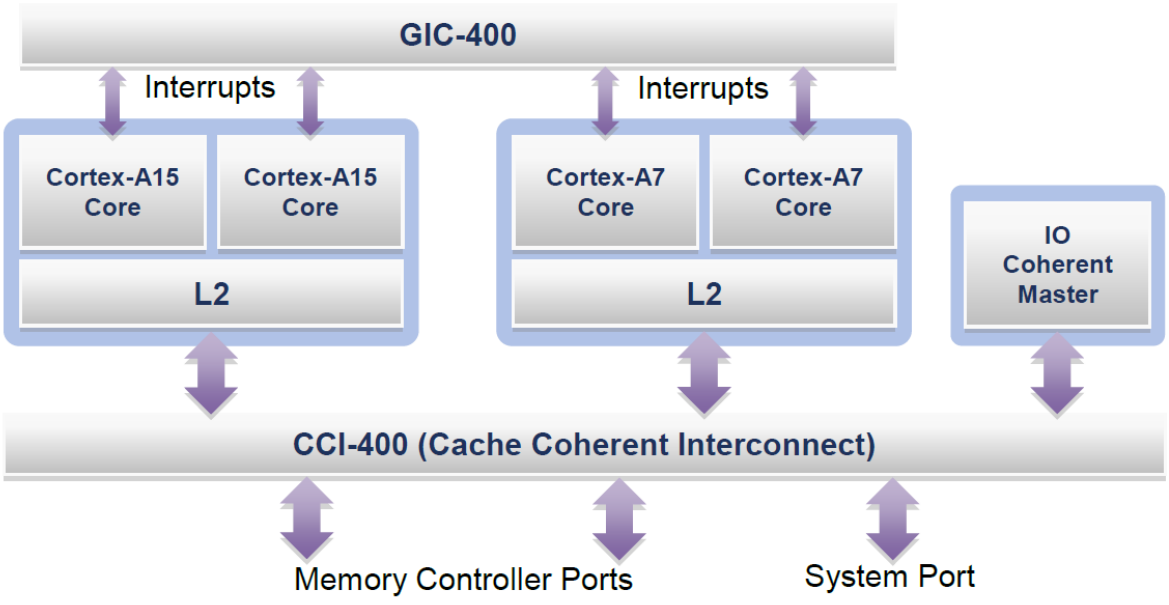
Fonte: (HARDKERNEL).

Ambos os conjuntos de processadores funcionam ao mesmo tempo, mas apenas o mais adequado será executado no recurso de processamento que é mais adequado. O outro fica em modo de espera, aguardando sua necessidade (HARDKERNEL, 2013).

Para criar uma solução big.LITTLE obrigando o sistema em torno dos processadores devem também precisam ser considerados. Uma parte fundamental é a interconexão de CCI-400, que facilita a coerência total entre Cortex-A15 e Cortex-A7, bem como outros componentes de entrada e saída, tais como uma GPU. Através de otimizações em torno das características da operação de Cortex-A15 e Cortex-A7, bem como por considerar os caminhos para a memória principal e o sistema, uma única solução é oferecida com o mais alto desempenho possível.

Outro elemento do sistema big.LITTLE é um genérico Controlador de interrupção compartilhada (GIC-400). Que é capaz de distribuir até 480 interrupções para Cortex-A15 e Cortex-A7, a natureza programável do GIC-400 permite que as interrupções sejam migradas entre quaisquer núcleos no Cortex-A15 ou clusters Cortex-A7. Não tem nenhum recurso que necessite de configuração específica. No entanto, para reduzir a complexidade do software no modelo de uso da arquitetura big.LITTLE é recomendado que o mesmo número de núcleos de sejam implementados no cluster Cortex-A15 e no Cortex-A7 (GREENHALGH, 2011).

Figura 20 – Sistema do cortex-A15 e cortex-A7



Fonte: (GREENHALGH, 2011).

3.2.3 System on Chip

Um System on Chip (SoC) é definido como um dispositivo com mais de cem mil portas lógicas que possua pelo um ou mais núcleos programáveis e uma memória no chip. O SoC junta componentes que estariam separados em placas convencionais, como processadores, codificadores, filtros ativos, protocolos e amplificadores operacionais. As principais vantagens desta abordagem são:

- Aumento da velocidade de operação do sistema, devido ao fato do fluxo de dados entre o processador e os outros componentes ocorrer no mesmo chip;
- Redução da potência consumida, devido ao alto grau de integração e uso de tensões menores;
- Redução de tamanho, atribuído a redução do uso de componentes adicionais;
- Redução no uso de trilhas nas placas de circuito impresso, aumentando a confiabilidade do sistema.

Esta arquitetura de chip está presente em dispositivos como *smartphones*, *tablets* e outros que necessitam de economia de energia. O crescimento do mercado de dispositivos móveis, aliada com os fatores apresentados anteriormente, faz com que os SoC estejam atingindo desempenho e eficiência energética cada vez maiores. (RAJOVICA ALEJANDRO RICOA, 2013).

3.3 Arduino

Arduino é uma plataforma de prototipagem com base em uma placa com pinos analógicos e digitais, com um microcontrolador integrado, ele é o que chamamos de plataforma de computação física ou embarcada, ou seja, um sistema que pode interagir com seu ambiente por meio de hardware e software.

Uma das vantagens é que os pinos podem ser configurados como entradas ou saídas, conforme necessário, ele interage com uma variedade de sensores para registar os *inputs* produzidos e processar os dados conforme os seus valores para periféricos externos, além de ser facilmente programado e apresenta um bom custo benefício. Há muitos tipos de placa, de acordo com o que o usuário deseja usar (ARDUINO, 2013). No caso desse projeto foi utilizado o Arduino Uno, mas sendo possível utilizar qualquer outra placa de prototipagem compatível.

Figura 21 – Placa Arduino Uno

Fonte: (Arduino)

A placa Arduino Uno possui um micro controlador ATmega328, que possui 32 KB de memória, sendo 0.5 KB utilizado para *bootloader*. Tem ainda 2 KB de SRAM e 1 KB de EEPROM que pode ser utilizada para leitura e escrita. A placa trabalha com uma tensão de 5 volts, possui 14 entradas digitais, 6 analógicas e um botão reset para reiniciar o micro controlador. Pode ser alimentado pela conexão USB ou uma por fonte externa (AC - CC) com tensão entre 7 e 12 volts (Recomendado).

3.4 Projeto OpenEnergyMonitor

Um projeto para o monitoramento de energia que é conduzida por uma comunidade de desenvolvimento aberto (fonte aberto e hardware aberto) sob o nome OpenEnergyMonitor (OEM). A comunidade procura "Desenvolver ferramentas de código aberto para ajudar no uso da energia e do desafio de energia sustentável". Todos os elementos constituintes do projeto estão disponíveis online. O software pode ser descarregado diretamente do *website* e o hardware pode ser adquirido em várias lojas credenciadas.

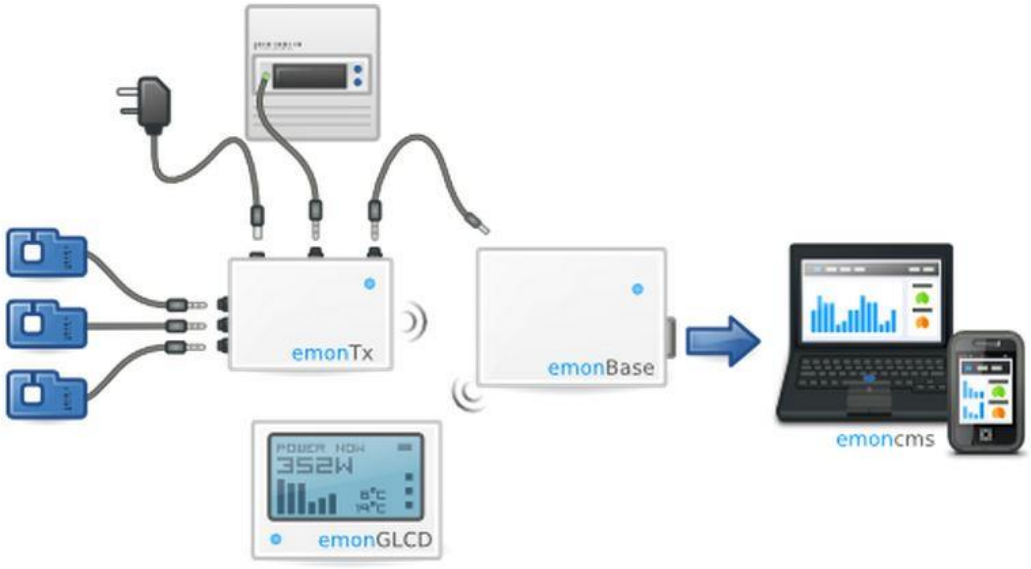
O sistema completo Figura 22, inclui um módulo de medição que pode ser ligado a três sensores de corrente adicionais, incluindo a transmissão de dados sem fios, um monitor sem fio para a exposição, uma estação base recebe dados de controle remoto da medição e os envia para um servidor *web*, e finalmente, uma aplicação *web* recebe o processamento, armazena ele e exibe seu consumo. As diferentes partes podem ser montadas e configuradas para trabalhar numa variada gama de aplicações. Estas vão desde monitores de energia, temperatura, pressão, até controladores de energia solar. Esta versatilidade do sistema permite utilizar os conhecimentos adquiridos na montagem do projeto para

outras aplicações.

Apesar da presença de todos estes sistemas no mercado atual, é de salientar a existência desse projeto extremamente interessante. O OpenEnergyMonitor (OEM), onde uma série de especialistas na área têm vindo a desenvolver vários sistemas de monitorização energética.

Para realizar a leitura de valores com sensores é necessário um circuito amplificado.

Figura 22 - Projeto OpenEnergyMonitor



Fonte: (OPENENERGYMONITOR, 2013).

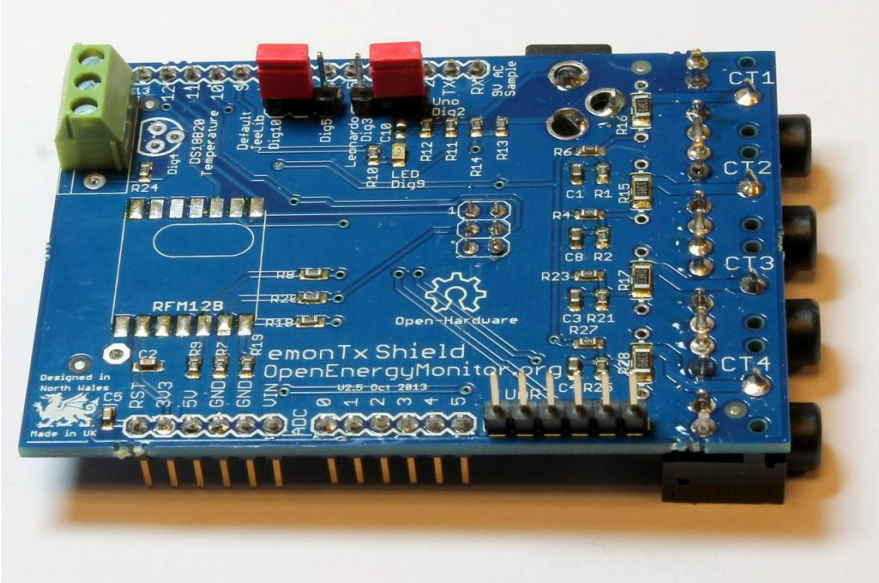
No caso particular do OEM, é utilizado uma placa de desenvolvimento Arduino, que comunica os dados medidos e calculados para uma interface construída na internet. Neste momento o sistema está direcionado para instalações monofásicas, pois apenas utiliza um sensor de corrente capaz de medir até 100A. O sistema desenvolvido pelo OEM está preparado a receber valores instantâneos de tensão fornecidos por um sensor (OPENENERGYMONITOR, 2013).

Há empresas que estão promovendo o conceito de "Smart Home" (casa inteligente) habitação controlada para a eficiência energética. Tendril, Onzo, Google PowerMeter, Silver Spring Networks, The Energy Detective (TED), Green Energy Options e Energy Aware, são algumas das empresas que estão oferecendo seus produtos e serviços para este fim.

A placa utilizada do OEM é a EmonTx Shield V2 Figura 23 por ela ser compatível com as placas Arduino. Um *shield* é uma placa de expansão de hardware que encaixa na placa Arduino

principal. Através dessas placas o Arduino ganha novas funcionalidades. O EmonTx possui 4 entradas para sensor de corrente 1 entrada para adaptador de 9 volts AC que possibilita a medição de tensão AC.

Figura 23 - Plataformas de prototipagem

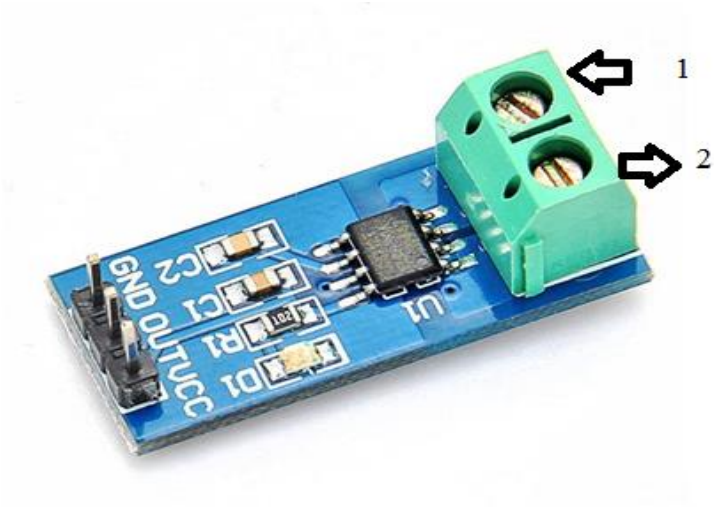


Fonte: (OPENENERGYMONITOR, 2013).

3.5 Montagem física

O hardware está configurado da seguinte forma, o Odroid sendo o nodo principal, que utiliza a biblioteca desenvolvida (Energy), estará esperando os dados dos sensores, nele está conectado um Arduino por uma conexão USB (Universal Serial Bus), no sistema operacional do Odroid a conexão USB é reconhecida como conexão serial, pois os drivers do Arduino fazem a virtualização de uma conexão serial encima da conexão USB. O Arduino está recebendo, tratando e repassando os dados de duas direções, a primeira do sensor ACS712 Figura 24, nele é ligado uma corrente positiva em 1, a partir da fonte de alimentação do Odroid, essa corrente continua e sai por 2 em direção ao Odroid, esse será a corrente medida pelo sensor. A alimentação do sensor é feita pela saída de 5 V do Arduino, conectada ao pino VCC do sensor, também é ligado o pino negativo do sensor, mais conhecido como terra (GND, do inglês *ground*) no pino GND do Arduino, os dados de medição são passados a partir da saída OUT do sensor, conectado no conversor AD A5 do Arduino.

Figura 24 – Sensor de corrente



Fonte: (OPENENERGYMONITOR, 2013).

A segunda direção é através da placa EmonTx Shield, ela é conectada no Arduino, adicionando as suas funções ao Arduino, a leitura das entradas da placa EmonTx são realizadas pelo Arduino nas conexões de A0 até A4, sendo A0 a leitura do sensor de transformador de tensão e as outras quatros dos sensores de corrente.

A placa EmonTx tem conectado dois sensores, um de tensão e o outro de corrente. Na entrada do adaptador AC-AC 9V um transformador de tensão 220V ou 110V para 9V, este transformador está ligado diretamente à tomada, com tensão de 220V ou 110V. Na entrada CT1 está ligado um sensor de corrente não invasivo através de um conector TRS (do inglês Tip-Ring-Sleeve, lit. ponta-anel-capa) P2, o qual saem dois fios que estão ligados ao sensor, fornecendo um sinal entre 0 e 1V para o microcontrolador. Para realizar a medição de corrente, o sensor deve ser envolvido apenas em um dos fios conforme Figura 14, caso seja ligado os dois fios, os valores se anulam e será mostrado um valor zerado

3.6 Software

Nesta seção está descrito os softwares utilizados para realizar a análise do consumo com a biblioteca desenvolvida. Primeiramente será descrito o sistema operacional utilizado e no decorrer é falado os compiladores e ambientes utilizados.

3.6.1 Sistema Operacional

A placa Odroid-XU+E é disponibilizada por padrão com o Android 4.2.2 JellyBean, um sistema operacional de código aberto mantido pela Google e baseado no *kernel* do Linux 3.4.39, mais utilizado por dispositivos móveis em geral, como telefones celulares, tabletes e sistemas embarcados. Devido ao pouco suporte oferecido pela comunidade, optou-se a instalação do sistema operacional Ubuntu, desenvolvido pela comunidade Linux, e também utiliza o núcleo do Linux.

O Ubuntu tem uma tem uma quantidade interessante de material disponível além de inúmeras versões compatíveis. A versão escolhida foi a última disponível até o atual momento, o Ubuntu Xubuntu 14.04 LTS (http://odroid.in/) que utiliza o *kernel* 3.4.98 do Linux e que acompanha uma grande quantidade de bibliotecas.

Para realizar a compilação da biblioteca foi utilizado o compilador GCC (Gnu Compiler Collection), no Ubuntu com a versão 4.8.2-19ubuntu1, ele constitui uma suíte de compiladores para diferentes linguagens de programação, entre elas C, C++ e Fortran. Com a utilização do GCC, a possibilidade de compilação de programas utilizando um software de código aberto robusto e portável tornou-se realidade. Atualmente, o GCC é um dos principais compiladores utilizados na comunidade de software livre, sendo constantemente atualizado com novas tecnologias.

Com o passar dos anos, o compilador GCC evoluiu e hoje gera programas para diversas arquiteturas, de sistemas embarcados a supercomputadores. Este também possui recursos que garantem a otimização do seu código binário, tanto em termos de tamanho de código quanto rapidez na execução, através do uso de *flags* de otimização.

A linguagem C que é uma linguagem de programação compilada, estruturada, procedural, padronizada pela ISO. É uma das mais populares linguagens e para a maioria das arquiteturas, existem compiladores para C.

3.6.2 Arduino

O Ambiente de Desenvolvimento Integrado (IDE, do inglês Integrated Development Environment) é um programa desenvolvido pela empresa Arduino Figura 25, ela é uma aplicação multiplataforma escrita em Java e derivada dos projetos Processing e Wiring, e que permite escrever esboços para a placa Arduino em uma linguagem simples.

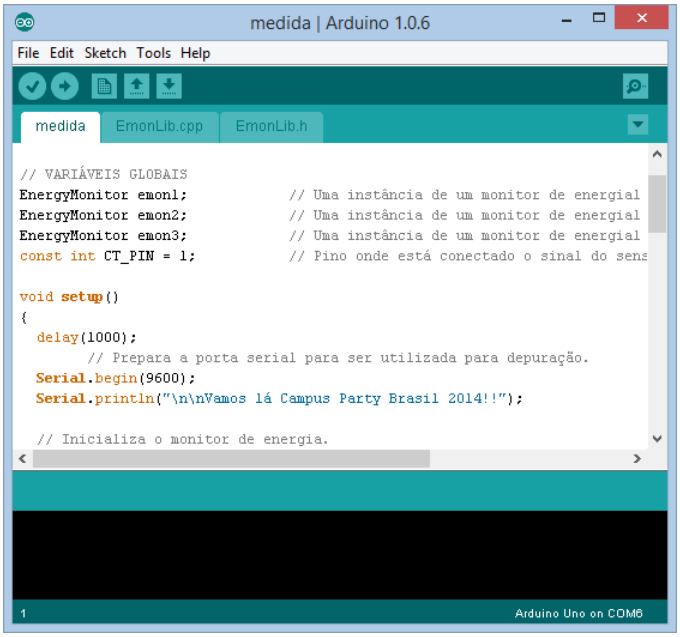
Ao solicitado pelo usuário a IDE carrega o código para a placa, o que você escreveu é traduzido para a linguagem C, e é passado para o compilador que faz a tradução final para a língua entendida pelo

micro controlador. Esta última etapa é muito importante, porque é onde a IDE torna a sua vida simples, tanto quanto possível escondendo as complexidades da programação de micro controladores. O ciclo de programação em Arduino é basicamente a seguinte:

- Ligue sua placa em uma porta USB no seu computador.
- Desenvolva um código que será carregado para placa.
- Envie este código para a placa através da conexão USB e aguarde alguns segundos para a placa reiniciar.
- A placa executa o código que você escreveu (BANZI, 2011).

O projeto OpenEnergyMonitor é compatível com o IDE do Arduino, o que facilita a sua utilização e percepção por parte do público em geral, através da utilização da biblioteca emonLib, criada por eles para trabalhar em conjunto com as placas emonTx.

Figura 25 – IDE Arduino



Fonte: (Elaborado pelo autor).

3.7 Biblioteca desenvolvida

Nesta seção será detalhada a biblioteca que foi desenvolvida, todas as chamas de funções e configurações disponível nela. Ao final da monografia a biblioteca será compartilhada no GitHub juntamente com todos os códigos fontes e o trabalho escrito, ela também estará no anexo do atual

trabalho.

3.7.1 Facilitar a Obtenção de Dados do Consumo de Energia

A biblioteca Energy foi desenvolvida na linguagem de programação C que é uma linguagem de programação compilada, estruturada, procedural, padronizada pela ISO. É uma das mais populares linguagens. As funcionalidades desenvolvidas permitem obter dados do consumo da aplicação após uma configuração inicia para definir alguns itens necessários, permitindo o monitoramento instantâneo através dos comandos implementados, retornando os dados do consumo multinível dessa aplicação.

Ela utiliza *threads* para ler os dados de corrente e tensão dos sensores, o que possibilita ao mesmo tempo que este processo está em execução, utilizar outras chamadas de métodos para monitorar o consumo e dentro do necessário aplicar medidas necessárias. Uma *thread* é um fluxo de execução individual, quando a execução de um processo é quebrada em duas ou mais partes que rodam em paralelo.

Por padrão a biblioteca irá realizar a leitura dos sensores a cada 10 segundos, mas com a intenção de criar uma biblioteca configurável, foi criado uma configuração para este tempo ser alterado.

3.7.2 Chamadas de Funções

As principais funções que estão implementadas na biblioteca Energy são **initEnergy()**, **setSensor()**, **setTime()**, **startEnergy()**, **stopEnergy()**, **getEnergy()**, **getCurrent()**, **getTension()** e **printEnergy()**. Em todas as funções detalhadas é necessário passar um ponteiro instanciado a partir da biblioteca Energy, e este é o primeiro parâmetro de todas as funções, antes de utilizar qualquer método da biblioteca deve ser alocado a memória para a variável que foi instanciada, elas serão detalhadas a seguir.

- Função **initEnergy()**: Essa é a primeira função que deve ser chamada, ela coloca o padrão das demais configurações, na chamada dela é passada a variável Energy e o tipo de sensor utilizado, 0 se for sensor integrado na placa ou 1 se for sensor lido por uma conexão externa e o seu retorno é 0 caso o processo foi executado com sucesso e 1 sem sucesso.
- Função **setSensor()**: Essa é a função utilizada para passar o caminho onde é realizada a leitura dos sensores. Na chamada dela é passada a variável Energy, o nome do sensor, e

em sequência mais três parâmetros que variam conforme o tipo de sensor iniciado, caso seja sensor interno da placa os parâmetros são, o caminho do sensor de tensão, o caminho do sensor de corrente e por último o local onde o sensor será iniciado, caso o sensor seja externo os parâmetros são, o id para comunicação externa, o segundo parâmetro não é necessário e deve ser passados como vazio, e por último o endereço para a comunicação externa, a potência será calculada através da tensão e da corrente. A função retornara 0 caso o processo foi executado com sucesso e 1 sem sucesso.

- Função **setTime()**: A chamada dessa função é utilizada para alterar o tempo entre cada leitura dos sensores, sendo necessário passar a variável Energy e o novo tempo de intervalo, a função retornara 0 caso o processo foi executado com sucesso e 1 sem sucesso. O tempo passado nesse método deve ser pensado com calma pois um tempo muito alto provavelmente irá causar erro na precisão do calculado realizado pela biblioteca.
- Função **startEnergy()**: Esta é o método chamado que iniciara a *thread* a qual estará realizando a leitura dos sensores e os cálculos de potência. Na sua chamada é passada a variável Energy e o seu retorno é 0 caso o processo foi executado com sucesso e 1 sem sucesso.
- Função **stopEnergy()**: Esta função é utilizada para finalizar a *thread* iniciada na função **startEnergy()**, e após sua execução é finalizada a leitura dos sensores. Para sua chamada é passada a variável Energy e o seu retorno é 0 caso o processo foi executado com sucesso e 1 sem sucesso.
- Função **getEnergy()**: ela pode ser usada de duas formas, a primeira para obter o consumo registrado desde o momento que essa variável começou a obter os dados dos sensores, ou da segunda forma, passando um tempo desejado para realizar a medição do consumo dentro desse intervalo de tempo. Na sua chamada é necessário passar a variável Energy e caso seja usado da primeira forma, deve ser passado 0, já para utilizar da segunda forma deve ser passado um valor maior que 0. O seu retorno será o valor do consumo levando em conta os dados passados na chamada da função.
- Função **getCurrent()** e **getTension()**: Estas duas funções retornam em sequência a corrente e a tensão da última leitura dos sensores. A chamada das duas funções é necessária passar a variável Energy e o seu retorno será o valor lido do sensor.

- Função **printEnergy()**: Essa função ira imprimir na tela o valor atual de tensão, corrente e potência consumida em W e potência consumida em KW/h, na sua chamada é necessário passar a variável Energy essa função não tem retorno.

Dentro da biblioteca existem outras funções que são utilizadas pelas funções detalhadas acima.

3.8 Medição dentro da aplicação

A medição dentro da aplicação é realizada por sensores internos conectados na placa e em seus respectivos componentes, sendo realizada uma medição mais detalhada, facilitando a identificação dos componentes de maior consumo dentro da aplicação.

4 RESULTADOS

Este capitulo tem os seguintes objetivos. O primeiro é a descrição dos ensaios realizados e como os dados obtidos pela biblioteca desenvolvida serão comparados para sua validação. O segundo é apresentar uma análise e discussão sobre os dados obtidos nos ensaios descritos anteriormente. E por fim é feito um relato dos das principais dificuldades para a realização e implementação do atual trabalho.

4.1 Ensaios

Para realização dos ensaios será utilizado a ferramenta Stress, ela é um gerador de carga de trabalho que pode ser configurada para realizar testes de estresse de CPU, I/O, memória e disco em sistemas operacionais. Sua licença é GPL. Esta ferramenta foi desenvolvida para vários sistemas operacionais. Assim, existem compilações específicas para determinados sistemas e, também, existe o código fonte disponível para compilação local.

Entre os ensaios realizados foram definidas 3 diferentes configurações da ferramenta Stress, as quais estão nomeadas de Ensaio 1, Ensaio 2 e Ensaio 3. Para definir tais configurações foi definido uma estratégia de simples experimentação, que apresenta apenas poucas alterações dos parâmetros configurados na ferramenta Stress, ainda será realizado o Ensaio 4 o qual ira consistir na medição do consumo durante a execução de um *benchmark.*

Todos ensaios realizados serão repetidos 5 vezes com a mesma configuração para aumentar a confiabilidade dos dados obtidos. O tempo entre cada leitura realizada pela biblioteca foi definida em 5 segundos evitando que a biblioteca interfira na utilização de energia. Os ensaios realizados serão detalhados a seguir, bem como as configurações utilizadas e os seus objetivos. Os resultados obtidos em cada um dos ensaios serão detalhados na seção 4.4.

Em todos os ensaios serão realizadas medições nos 3 diferentes níveis descritos anteriormente tendo as informações separadas com AC sensor A da Figura 13, DC sensor B também da Figura 13,

A15 sensor no processador Cortex-A15, A7 sensor no processador Cortex-A7, MEM sensor na memória RAM e GPU que é o sensor na GPU, os últimos 4 sensores, são os integrados na placa Odroid.

Ensaio 1: O primeiro ensaio é medido o consumo energético com o processamento da placa Odroid totalmente ocioso (*idle*) durante o tempo de 5 minutos, ou seja, é medido a quantidade de energia gasta para apenas manter a placa com o sistema operacional executando e mantendo ele ligado aguardando qualquer interação do usuário, este é o modo que em que o ambiente testado deve ter o menor consumo.

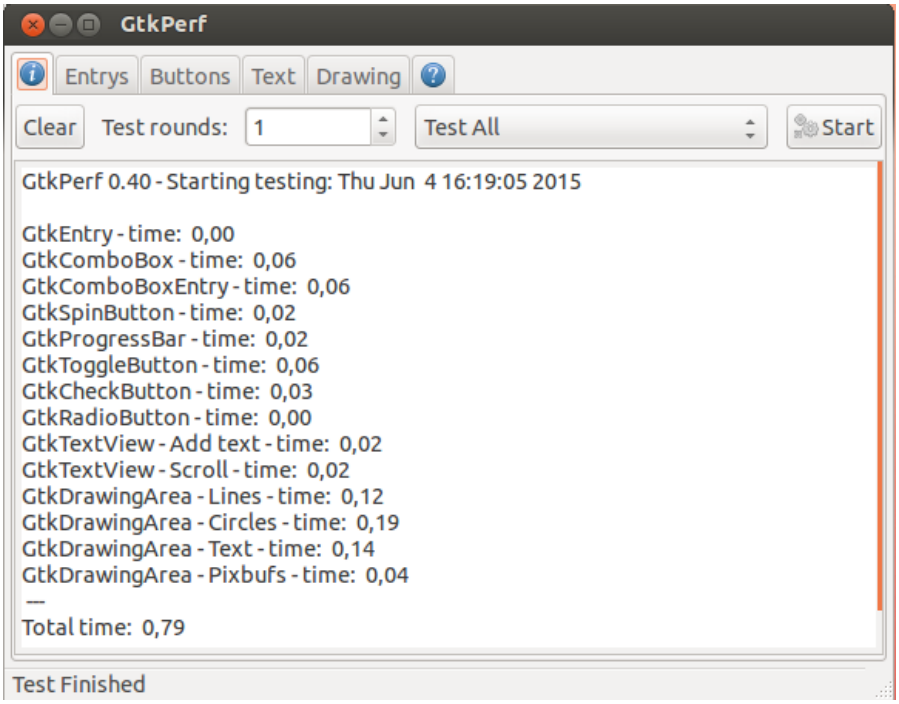
Ensaio 2: O segundo ensaio será realizado utilizando a ferramenta Stress, a configuração dos parâmetros foi feita pensando na utilização de 50% do seu processamento que será através do comando **stress --cpu 1 --vm 1 --vm-bytes 512M --timeout 5m**, que irá iniciar dois processos que irão utilizar 2 núcleos dos 4 que são utilizados em conjunto, um processo estará alocando e liberando 512MB de memória: um durante um tempo de 5 minutos, o outro estará realizando outro processo, utilizando 100% de um dos núcleos. A intenção seria utilizar 50% do processamento da placa, mas sabendo que a CPU também estará realizando processos para o gerenciamento do sistema operacional. Portanto, na prática o processamento será um pouco superior a 50%.

Ensaio 3: O terceiro ensaio também será utilizando a ferramenta Stress, a configuração dos parâmetros foi feita para ter a utilização de 100% do processamento possível, que será através do comando **stress --cpu 4 --vm 2 --vm-bytes 512M --timeout 5m**, que irá iniciar 6 processos dos quais 2 irão estar alocando e liberando 512MB de memória cada um, os outros 4 irão ficar executando processos para nenhum dos 4 núcleos ficarem ociosos, sendo que todos os 6 processos continuarão em execução durante 5 minutos.

Ensaio 4: No quarto ensaio será realizado um *benchmark* com a ferramenta GtkPerf. Ele foi criado para realizar testes de desempenho de um computador. Essa ferramenta pode ser baixada diretamente pela central de softwares de boa parte das distribuições Linux. O GtkPerf executa uma sequência de testes comuns, como abrir caixas de diálogo, simular cliques do mouse, navegar por textos e desenhar uma série de imagens de diversas cores e formatos na tela, ele executa um total de 14 testes como pode ser visto na Figura 26.

Para executar o *benchmark* é passado um parâmetro de quantas vezes cada teste deve ser executado, no exemplo da Figura 26 foi passado para executar apenas 1 vez cada teste, ao final é mostrado o tempo total para executar todos os testes. Nesse ensaio foi configurado o parâmetro para repetidor 900 vezes cada teste, de forma que o tempo total ficasse perto dos 5 minutos.

Figura 26 - Ferramenta GtkPerf



Fonte: (Elaborado pelo autor).

4.2 Validação

Para realizar a validação dos dados obtidos pelos sensores foi utilizado um multímetro. Os testes comparativos foram feitos nos sensores entre a rede elétrica e a fonte de energia e entre a fonte de energia e a plataforma computacional onde estará rodando a biblioteca desenvolvida. A validação é parte fundamental do trabalho, pois através dela é realizada a calibração dos sensores, a partir da validação e da calibração os dados obtidos nas medições realizadas serão confiáveis.

Levando em conta que o desenvolvedor do Odroid deve ter feito validações nos sensores internos da placa, foram feitas as leituras com o intuito de verificar se esses dados estão corretos. A única verificação que foi feita é que o consumo interno do Odroid não seja maior que a medição do consumo entre a fonte e a placa.

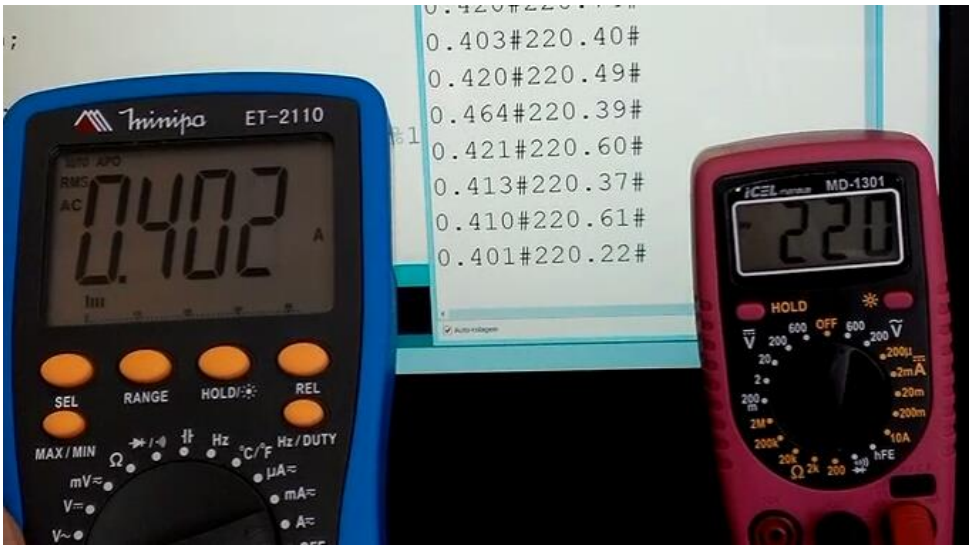
Ao tentar realizar a validação da corrente entre o Odroid e a fonte, houve dificuldades que serão detalhadas na seção 4.4. Devido a tais dificuldades foi necessário utilizar uma outra plataforma computacional para poder realizar todos as validações propostas anteriormente. A plataforma adicionada foi um *notebook* Asus, que foi substituído pelo Odroid apenas nas validações, também foi substituído a fonte do Odroid pela fonte do *notebook*. Após a validação das medições utilizando o

notebook, foi retornado para o *setup* inicial para realizar as validações possíveis também utilizando o Odroid e sua fonte.

As validações realizadas para comparação com a tensão e corrente obtida através dos sensores foram:

- Medição da tensão e corrente entre rede elétrica e a fonte de energia utilizando o *notebook* e sua fonte, e utilizando dois multímetros, conforme Figura 27

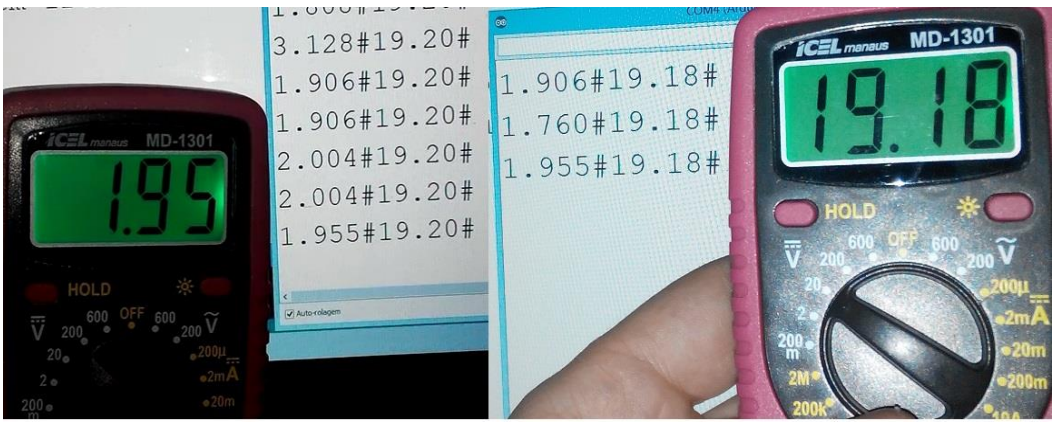
Figura 27 - Corrente e tensão AC



Fonte: Elaborado pelo autor.

- Medição da tensão e corrente entre a fonte de energia e a plataforma computacional utilizando o *notebook* e sua fonte, e utilizando dois multímetros, conforme Figura 28.

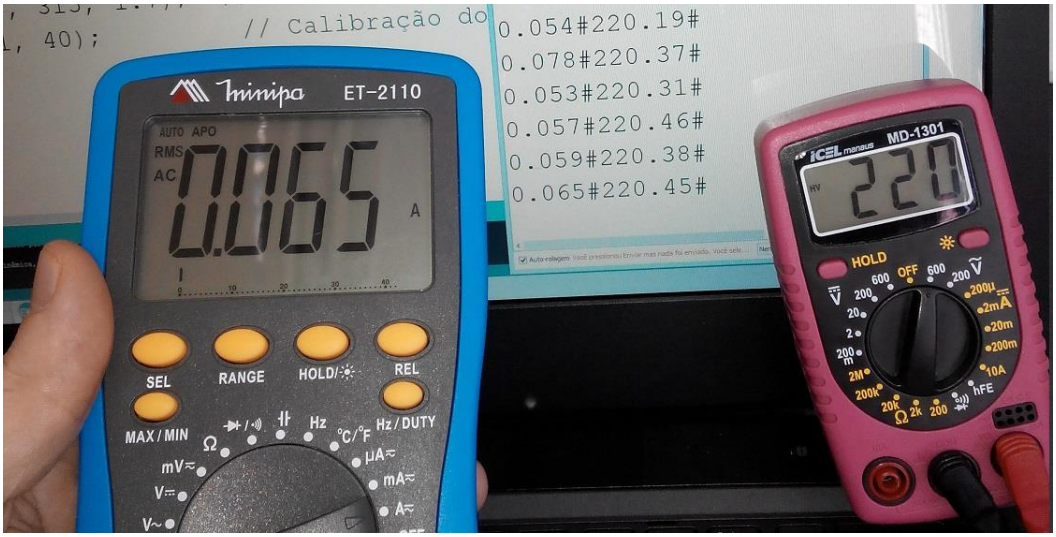
Figura 28 - Corrente e tensão DC



Fonte: Elaborado pelo autor.

- Medição da tensão e corrente entre rede elétrica e a fonte de energia utilizando o Odroid e sua fonte, e utilizando dois multímetros, conforme Figura 29.

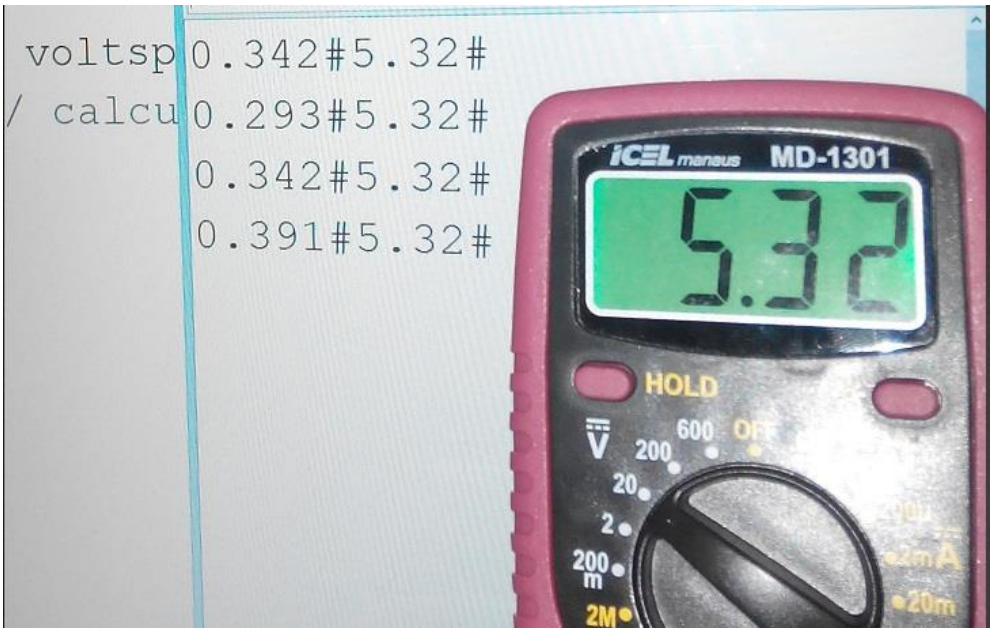
Figura 29 - Corrente e tensão AC



Fonte: Elaborado pelo autor.

- Medição da tensão entre a fonte de energia e a plataforma computacional utilizando o Odroid e sua fonte, e utilizando um multímetro, conforme Figura 30.

Figura 30 - Tensão DC



Fonte: Elaborado pelo autor.

4.3 Resultados e discussões

Nesta seção serão detalhados os testes realizados nos ensaios de 1 a 4. A nomenclatura utilizada será: A15 sensor no processador Cortex-A15, A7 sensor no processador Cortex-A7, MEM sensor na memória RAM e GPU que é o sensor na GPU, ODROID o valor da soma dos 4 sensores anteriores, DC sensor B da Figura 13, AC sensor A também da Figura 13, todos os ensaios foram realizados medições nos 3 diferentes níveis.

Ensaio 1: No primeiro teste realizado analisando os dados da Tabela 3, já foi possível ver a arquitetura big.LITTLE em funcionamento, durante o teste de execução do Odroid com o processamento ocioso, o processador A15 consumiu menos energia que o processador A7, a partir disso é possível deduzir que processador A7 estava gerenciando o sistema.

Nesse teste também foi possível ver que o consumo da fonte, pegando o consumo no sensor AC e descontando o consumo no sensor DC, foi maior que 70% do consumo total.

Tabela 3 - Consumo em Idle

Sensor	Tempo	Consumo W	Consumo KWh
A15	5m	1,75873	0,000029
A7	5m	3,081404	0,000051
MEM	5m	16,586161	0,000276
GPU	5m	0,999728	0,000017
ODROID	5m	22,426023	0,000373
DC	5m	815,635559	0,013594
AC	5m	3725,089844	0,062085

Fonte: Elaborado pelo autor.

Ensaio 2: No segundo teste realizado analisando os dados da Tabela 4 ficou ainda mais evidente o funcionamento da arquitetura big.LITTLE, mesmo a placa estando em processamento em torno de 50% do total, o consumo do processador A7 foi baixo levando em consideração o consumo do processador A15, esse visivelmente esteve realizando o processamento pesado e claro elevando em muito o seu consumo de energia, sendo quase que 50% da energia consumida após o sensor DC, durante este ensaio foi acompanhado o processamento da placa e foi verificado que ele esteve dentro do esperado, estando 2 núcleos com 100% de processamento, e os outros estiveram em torno de 5%.

O consumo da fonte neste teste ficou um pouco superior a 60% do consumo total, mesmo tendo um grande aumento no consumo, em percentagem esse aumento foi menos que o aumento no sensor DC. No sensor AC o aumento foi de 54%, já no sensor DC o aumento foi de 150%.

Tabela 4 - Consumo em 50% de processamento

Sensor	Tempo	Consumo W	Consumo KWh
A15	5m	959,831726	0,015997
A7	5m	6,08097	0,000101
MEM	5m	34,558296	0,000576
GPU	5m	1,113282	0,000019
ODROID	5m	1001,584274	0,016693
DC	5m	2043,705566	0,034062
AC	5m	5737,561035	0,095626

Fonte: Elaborado pelo autor.

Ensaio 3: Nos testes realizados no terceiro ensaio fazendo uma análise dos dados da Tabela 5, pode ser visto em comparação ao ensaio 2 que mesmo aumentando o processamento total em quase 50% o aumento do consumo total foi de apenas 15% no sensor AC, no sensor DC o aumento foi de 20% e por sua vez o aumento do consumo no processador A15 foi de 31%. Os outros sensores, todos tiveram uma pequena variação no consumo. Em comparação ao primeiro ensaio o consumo total teve aumento de 78% no sensor AC.

No ensaio 3 o consumo da fonte ficou em torno de 63% do consumo total de energia.

Tabela 5 - Consumo em 100% de processamento

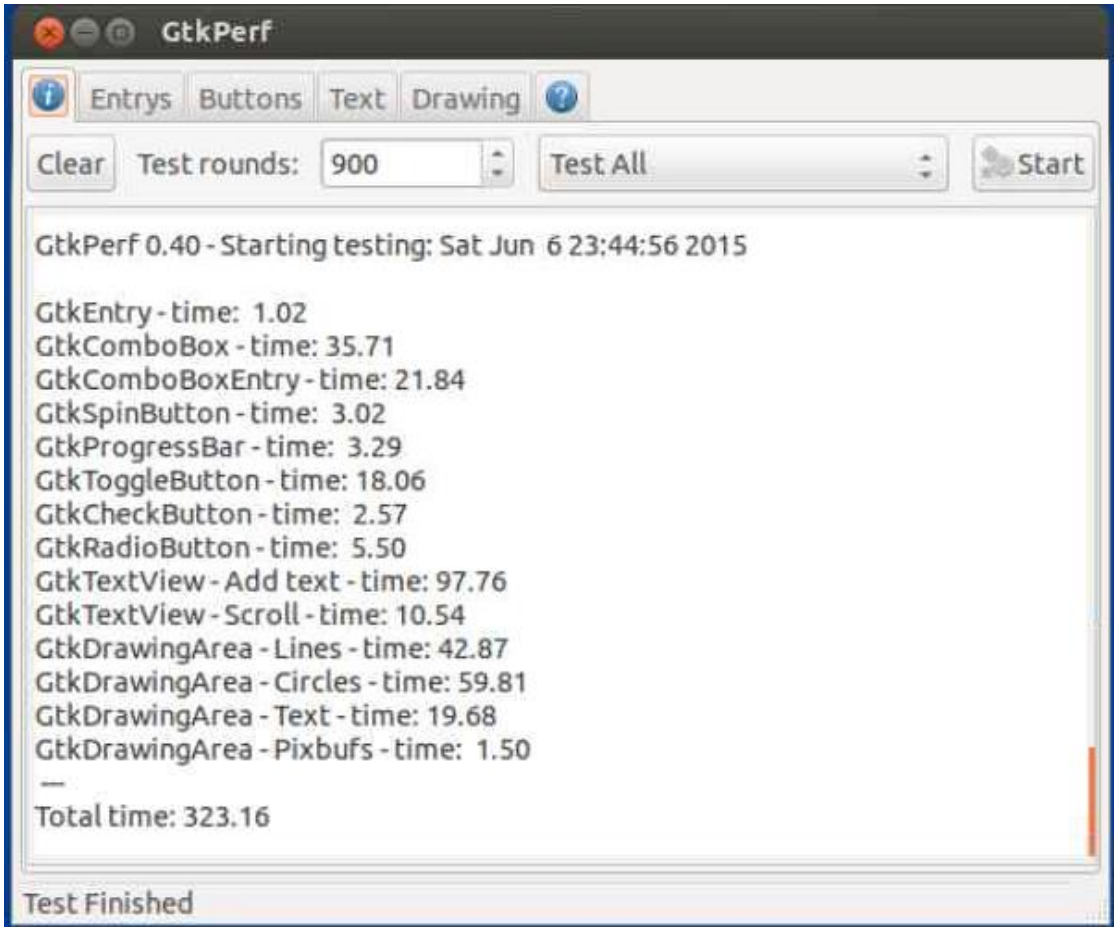
Sensor	Tempo	Consumo W	Consumo KWh
A15	5m	1262,639526	0,021044
A7	5m	6,618042	0,000110
MEM	5m	35,105316	0,000585
GPU	5m	1,494112	0,000025
ODROID	5m	1305,856996	0,021764

Sensor	Tempo	Consumo W	Consumo KWh
DC	5m	2467,78833	0,041130
AC	5m	6632,644043	0,110544

Fonte: Elaborado pelo autor.

Ensaio 4: No quarto e último ensaio realizado analisando os dados da Tabela 6 e fazendo o acompanhamento do ensaio, foi possível ver que o processamento dos núcleos foram bem variados em alguns momentos estando em 100% e outros momento estavam entre 30 e 80%, dependendo do tipo de teste realizado durante o *benchmark*, o tempo total de sua realização foi de 5 minutos 23 segundos e 160 milésimos para a realização dos 14 testes com repetição de 900 vezes conforme pode ser visto na Figura 31, com esse ensaio pode ser feito um comparação de processamento com consumo de energia em outras plataformas.

Figura 31 – Resultado benchmark



Fonte: Elaborado pelo autor.

O consumo obtido em todos os sensores foi menor que o consumo obtido nos sensores no ensaio 2, sabendo que esse *benchmark* realiza testes comuns, esse ensaio pode ficar mais perto do consumo durante a utilização de um usuário, que não realiza uma grande demanda de processamento na utilização de uma plataforma computacional.

Tabela 6 - Consumo durante a execução de Benchmark

Sensor	Tempo	Consumo W	Consumo KWh
A15	5,23m	647,183838	0,010786
A7	5,23m	5,899576	0,000098
MEM	5,23m	25,048946	0,000417
GPU	5,23m	1,070956	0,000018
ODROID	5,23m	679,203316	0,011319
DC	5,23m	1687,716797	0,028129
AC	5,23m	4670,169434	0,077836

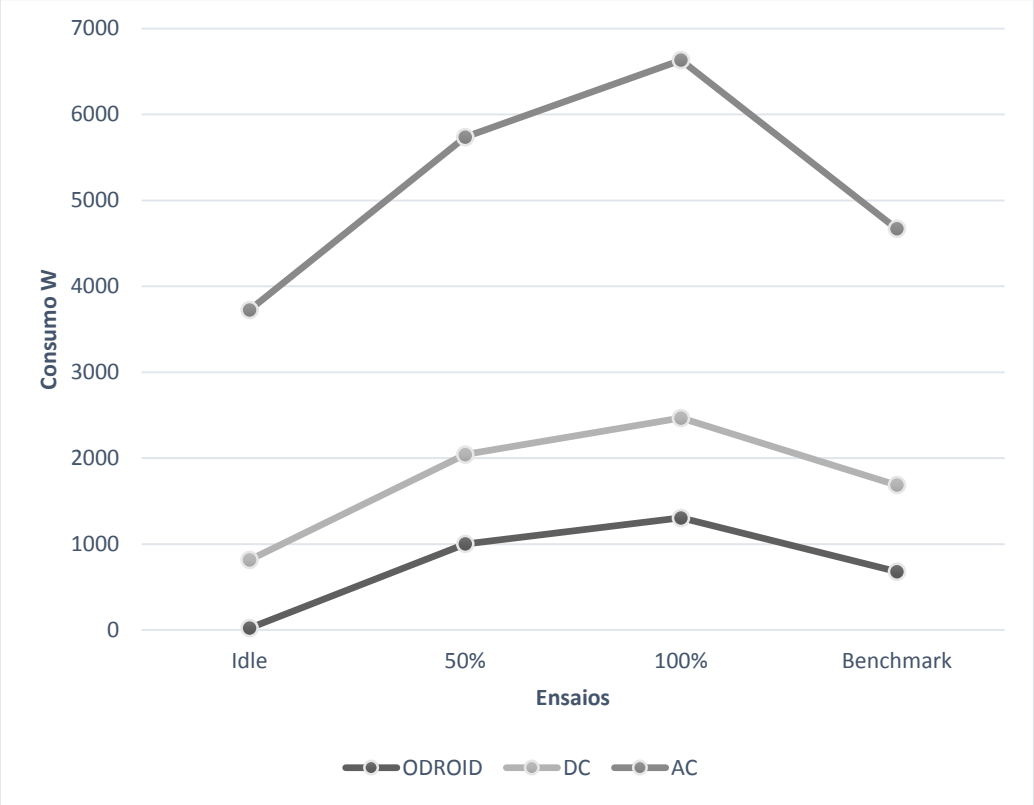
Fonte: Elaborado pelo autor.

Em todos os ensaios realizados fazendo uma análise geral pode ser visto que a fonte foi onde se concentrou o maior consumo, em todos os ensaios esse consumo foi maior que 60% do consumo total e quando a placa Odroid estava ociosa esse valor foi maior ainda passando dos 70%.

Na Figura 32 pode ser visto um gráfico do consumo por nível medido em todos os 4 ensaios, nesse gráfico fica extremamente evidente que o consumo da fonte da placa é o local de maior consumo elétrico.

Foi possível verificar em todos os ensaios realizados, que o aumento do consumo foi relativamente proporcional ao aumento do processamento, nos sensores integrados na placa Odroid e no sensor DC.

Figura 32 – Gráfico consumo por nível

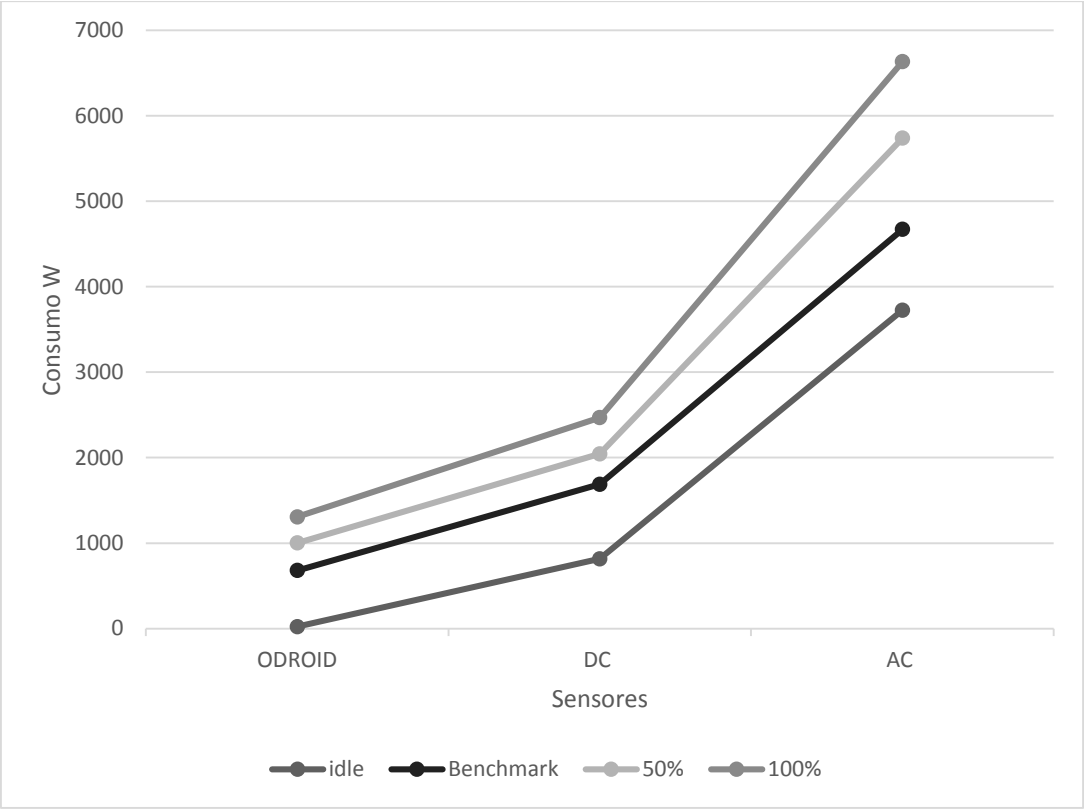


Fonte: Elaborado pelo autor**.**

A Figura 33 mostra um gráfico do consumo em cada nível como proposto nos objetivos desse trabalho, tendo em cada linha do gráfico o consumo do ensaio nos 3 níveis, sendo o primeiro os 4 sensores integrados na placa nomeado como ODROID, o segundo a medição entre a placa Odroid e a fonte nomeado como DC e o terceiro a medida entre a fonte e a rede elétrica nomeado como AC.

Pode ser levado em conta que a placa Odroid foi desenvolvida para ter um baixo consumo de energia, a placa atende totalmente a esses requisitos e quando necessita de maior processamento o seu consumo energético aumenta em proporção relativamente parecida.

Figura 33 – Gráfico consumo por ensaio



Fonte: Elaborado pelo autor**.**

4.4 Relatos de dificuldades

A principal dificuldade encontrada foi para realizar a validação dos dados obtidos de corrente entre o Odroid e a fonte, para realizar essa validação foi aberto o circuito e adicionado o multímetro para realizar a medição da corrente, mas ao adicionar o multímetro para realizar a validação a placa Odroid realizava a identificação de alimentação ligando o LED vermelho e ao iniciar suas validações internas ligava o LED verde, mas não realizava a inicialização do sistema operacional ficando apenas com um LED vermelho acesso e apagando o LED verde.

A inicialização normal do Odroid é identificar a alimentação ligando o LED vermelho, antes de iniciar o *boot* da placa ligava o LED verde e quando inicia o sistema operacional a placa desliga o LED verde e liga o LED azul, após iniciado o sistema operacional o LED azul permanece piscando até o sistema operacional ser desligado.

Inicialmente foi tentado trocar o multímetro imaginando que o utilizado estava com problemas, mas tendo o mesmo resultado, o segundo teste feito foi colocar um multímetro de melhor qualidade para realizar a medição continuando o mesmo resultado e o sistema operacional não iniciado. O último

teste com multímetro foi utilizando o Agilent 34461A, mas também houve o mesmo problema. Em todos os testes o multímetro media a corrente que era utilizada pela placa durante o pouco tempo que ela se ligava para realizar seus testes de *boot*. Sempre que retirado o multímetro a placa iniciava normalmente, mesmo utilizando o sensor de efeito hall ASC712.

Outra dificuldade encontrada foi para instalação de uma distribuição Linux na placa Odroid, pois o *site* do desenvolvedor diretamente nele só disponibiliza a instalação do sistema operacional Android. O sistema operacional encontrado em um outro *site* também mantido pelo desenvolvedor da placa, que foi encontrada apenas pela pesquisa em fóruns relacionados ao Odroid-XU+E, no *site* principal do desenvolvedor, esse outro *site* não está identificado.

5 CONCLUSÃO

O presente trabalho apresentou uma análise do consumo energético em diferentes níveis de uma plataforma computacional, utilizando a biblioteca desenvolvida para obtenção dos dados de consumo. Esta proposta vai ao encontro do aumento do consumo de energia elétrica e do atual cenário brasileiro de aumento nos valores cobrados pela energia consumida, possibilitando acompanhar o consumo energético atual e dentro da necessidade realizar medidas no atual momento de redução de gastos.

O referencial teórico apresentou os fundamentos para a instrumentação e utilização de sensores, as formas de medir a potência e como obter o consumo energético total. No desenvolvimento foi montado um *setup* do hardware para realizar a medições detalhando como as placas e sensores foram interligados, descrito o hardware e software utilizado. Também foi detalhado o funcionamento da biblioteca desenvolvida, descrevendo todas suas funções e configurações necessárias para sua utilização.

Com base na análise dos ensaios realizados e no *setup* montado foi possível concluir que a maior parte do consumo energético de uma plataforma computacional está na fonte que alimenta essa plataforma, sendo sempre superior a 60% do consumo em relação ao consumo total nos ensaios realizados. Sendo assim, apenas diminuir o processamento ou dividir entre núcleos existentes é uma das soluções possíveis, mas não é o suficiente para uma melhor utilização energética. De fato, caso a plataforma esteja ociosa o ideal seria efetuar o seu total desligamento.

Outra contribuição importante é a comprovação que 50% do consumo energético da placa esteve relacionado diretamente com o processador. Fazendo-se a análise dos ensaios realizados ficou evidente que o aumento do consumo da placa foi relativamente proporcional ao aumento do processamento realizado.

É importante citar que foi possível realizar uma análise com maior precisão devido à os sensores integrados na placa Odroid, entre eles os sensores no processador. Este tipo de recurso diretamente integrado nas placas possibilita realizar o controle do consumo energético pelo tipo de

processamento realizado, também podendo ser implementado no sistema operacional.

Também analisando apenas o consumo energético do processador foi possível ver o funcionamento da arquitetura big.LITTLE da ARM, ficando evidente a utilização dos núcleos de baixo consumo durante processos que necessitam de pouco processamento. Ao ser solicitado um maior processamento, o consumo passou quase que totalmente para os núcleos que possibilitavam atender à demanda desse processamento.

A principal contribuição deste trabalho foi a implementação da biblioteca para obtenção do consumo energético, possibilitando realizar a medição e análise de quanta energia uma aplicação consome durante a sua execução, podendo serem realizadas ações referentes à essa aplicação durante a sua execução, ou mesmo limitando o consumo diário de uma aplicação com base no processamento utilizado por essa aplicação. Essa biblioteca estará disponível publicamente no ambiente GitHub.

Como oportunidade de continuação desse trabalho de pesquisa, cita-se a possibilidade de realizar a medição do consumo em sistemas distribuídos, analisando a melhor forma de distribuir o processamento entre os variados nos do sistema, ou se seria mais vantajoso realizar o total desligamento d um nodo juntamente com sua fonte de alimentação, para uma melhor utilização do consumo energético.

Esse tipo de análise também poderia ser realizado em servidores, ou qualquer tipo de sistema que realize grande quantidade de processamento ou de consumo energético.

Outra possibilidade de continuação para o atual trabalho seria realizar uma análise do consumo de energia em alguns algoritmos específicos que utilizam uma maior carga de trabalho da CPU, como por exemplo, algoritmos combinatórios, de busca, de pesquisa em strings, de ordenação, de compressão de dados, de computação gráfica ou ainda algoritmos criptográficos.

REFERÊNCIAS

ADELLE MARC PALLEMAERTS, J. C. C. Report of SIEPS. Climate Change and Energy Security in Europe Policy Integration and its Limits. Stockholm: Swedish Institute for European Policy Studies, 2009.

ALBIERO, F. W. Monografia de graduação em ciência da computação. Monitoramento e avaliação do consumo energético em função do poder computacional. Santa Maria: UFSM, 2010.

ALMEIDA, A. L. B. Monografia de graduação em engenharia de controle e automação. Desenvolvimento de um sistema sem fios e microcontrolado de medição inteligente de energia elétrica para cargas residenciais. Ouro Preto: CECAU, 2011.

ARDUINO. Arduino Uno. [s.n.], 2013. Online; acessada em 10 de abril de 2015. Disponível em: <http://www.arduino.cc>.

BANZI MASSIMO. Getting Started with Arduino. [S. l.]: Publisher: Maker Media, 2011.

BEGA, E.A. Nível. In: BEGA/DELMÉE/COHN/BULGARELLI/KOCH/FINKEL. **Instrumentação Industrial.** 3º Edição. Rio de Janeiro: Editora Interciência, 1-50, 2011.

BRAGA, N. C. Livro. Os segredos no uso do multímetro. São Paulo: Instituto newton c braga, 2013.

CALDEIRÃO, L. C. Pós-Graduação Em Engenharia Elétrica. Avaliação Experimental de Medidores Watt-Hora Operando em Condições Não-Senoidais. Ilha Solteira: Faculdade De Engenharia De Ilha Solteira, 2005.

CARVALHO ALEXANDRO BALDASSIN, R. A. João P. L. de. XIV Simpósio em Sistemas Computacionais. Reavaliando a Eficiência Energética de Memória Transacional em Processadores Convencionais. Rio Claro, Campinas: XIV Simpósio em Sistemas Computacionais, 2013.

CHAVES, C. F. Pós-Graduação em Engenharia Elétrica. Transformador de Corrente Eletrônico Utilizando Bobina de Rogowski e Interface Óptica com POF para Aplicação em Sistemas de Potência. Rio de Janeiro: Universidade Federal do Rio de Janeiro, 2008.

CORRÊA, R. V. Dissertação (Mestrado em Computação Aplicada). Otimização de desempenho utilizando contadores de hardware. São José dos campos: instituto nacional de pesquisas espaciais, 2000. Online; acessada em 13 de Outubro de 2014. Disponível em: <http://mtc-m05.sid.inpe.br/col/sid.inpe.br/deise/2001/08.03.12.26/doc/Pdfs/dissertacao.pdf>.

DUARTE, F. J. A. Dissertação de Mestrado em Engenharia de Redes de Comunicação e Multimídia. Classificação de atividades físicas através do uso do acelerómetro do Smartphone. Lisboa: instituto superior de engenharia de Lisboa, 2013.

FENG, T. S. B. S. W. chun. The Green500 list: escapades to exascale. Virginia Tech: Department of Computer Science, 2012. Online; acessada em 01 de novembro de 2014. Disponível em: <http://tom.scogland.com/pubs/pdf/scogland-green500-isc12.pdf>.

FILHO, A. P. S. J. A. B. Artigo cientifico. Processadores Low-Power/Energy Efficient. Campinas: Centro de Pesquisas Wernher Von Braun, 2011.

GREENHALGH PETER. big.LITTLE Processing with big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7, Improving Energy Efficiency in High-Performance Mobile Platforms, 2011, ARM p. Disponível em <http://www.arm.com/files/downloads/big_LITTLE_Final_Final.pdf >. Acesso em: 29 mar. 2015.

GUIMARÃES, V. V. M. Graduação, Desenvolvimento de um módulo de monitoramento de consumo de energia para a construção de Smart Appliances utilizando o Sun SPOT. Porto Alegre: [s.n.], 2011.

HARDKERNEL, O. Copyright 2013 Hardkernel co., Ltd. ODROID-XU+E. Gwanyang: [s.n.], 2013. Online; acessada em 15 de Novembo de 2014. Disponível em: <https://www.hardkernel.com >.

HIGASHI, E. M. Dissertação de Mestrado, Programa de Pós-graduação em Engenharia. Modelagem Da Bobina De Rogowski Para Medidas De Pulsos De Corrente Elétrica. Curitiba: Universidade Federal do Paraná, 2006.

HULLER, I. **Sistema de atuação e supervisão para segurança de ambientes remotos**. 2014. 109 p. Dissertação – Curso de Pós-Graduação em Desenvolvimento de Produtos Eletrônicos, Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, Florianópolis, 2014.

INATOMI, T. A. H. Dissertação de Mestrado. Análise dos impactos ambientais na produção de energia dentro do planejamento integrado de recursos. São Paulo: USP, 2000.

IRWIN, D. J. Livro. Análise Básica de Circuitos para Engenharia - 10a Ed. [S.l.]: LTC Editora, 2013.

JR, R. K. R. XIV Simpósio em Sistemas Computacionais. Introdução a Sistemas de Informação, 3a Edição. [S.l.]: Elsevier Brasil, 2012. ISBN 9788535249132.

KRITIKAKOS, P. Dissertação (Mestrado em Computação de Alto Desempenho). Low-Power High Performance Computing. Escócia: Universidade de Edimburgo, 2011. Online; acessada em 25 de setembro de 2014. Disponível em: <https://www.epcc.ed.ac.uk/sites/default/files/Dissertations/2010-2011/PanagiotisKritikakos.pdf>.

LEMOS, M. Medidor de Consumo de energia elétrica conectado a novem com Arduino. São Paulo: Arduino Day, 2014.

MARTINS, G. M. Sensores. Santa Maria: Universidade Federal de Santa Maria, 2008.

MATOS, L. S. N. E. D. M. L. B. de. Artigo Teórico. Técnicas de medição do consumo de energia em plataformas computacionais. Salvador: Revista de Sistemas e Computação - UFS, 2011.

MME, M. de Minas e E. Plano Decenal de Expansão de Energia – PDE 2022. Plano Decenal de Expansão de Energia 2022. Brasília: Empresa de Pesquisa Energética. 2013.

NERY, M. V. Dissertação (Graduação). Solução Para Gestão Do Consumo De Energia Residencial. Brasília: Centro Universitário De Brasília - Uniceub, 2013.

OPENENERGYMONITOR. OpenEnergyMonitor. [s.n.], 2013. Online; acessada em 15 de novembro de 2014. Disponível em: <http://openenergymonitor.org>.

PRIETO, E.; AUGUSTO, P.; MIGUEL, C. **Adoção da estratégia modular por empresas do setor automotivo e as implicações relativas à transferência de atividades no desenvolvimento de produto: Um estudo de casos múltiplos**. Gestão da Produção - São Carlos, v. 18, n. 2, p. 425–440, 2011.

RAJOVICA ALEJANDRO RICOA, N. P. C. A.-J. A. R. N. Tibidabo: Making the Case for an ARM-Based HPC System. [s.n.], 2013. Online; acessada em 15 de novembro de 2014. Disponível em: <http://www.montblanc-project.eu/sites/default/files/publications/armhpc-sc.pdf>.

RÉGIS, G, R. **Plataforma modular para automação, controle e supervisão de sistemas**. 2014. 71 p. Dissertação – Curso de Pós-Graduação em Desenvolvimento de Produtos Eletrônicos, Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, Florianópolis, 2014.

ROSTIROLA, G.; et. al. **GreenHPC: A Novel Framework to Measure Energy Consumption on HPC Applications**. **Applied Computing Graduate Program**, São Leopoldo, 2015.

ROSTIROLLA, G. Monografia de graduação em engenharia da Computação. Análise de desempenho e consumo energético de um cluster baseado em computadores de placa única. Lajeado, Centro universitário univates centro de ciências exatas e Tecnológicas, 2014.

SANTOS, L N.; et. al. **Técnicas De Medição Do Consumo De Energia Em Plataformas Computacionais**, Sergipe, 2011.

TEIXEIRA, L. L. Monografia de graduação em engenharia da elétrica. Medidor de Energia Eletrônico. Porto Alegre: Universidade Federal do Rio Grande do Sul, 2009.

THOMAZINI, Daniel; ALBUQUERQUE, Pedro U. B. **Sensores industriais: Fundamentos e aplicações**, 4 ed rev. São Paulo: Érica, 2007.

WHITAKER, D. T. B. Bi-Directional Inverter and Energy Storage System. University of Arkansas: University of Arkansas, 2008.

.

ANEXO A - BIBLIOTECA DESENVOLVIDA, ENERGY

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <termios.h>
#include <fcntl.h>
#include <pthread.h>

typedef struct Energy Energy;

struct Energy{
	char* name;
	char* tensionSensor;
	char* currentSensor;
	char* initiate;
	float tension;
	float current;
	float instantPower;
	float power;
	int sleep;
	int time;
	int loop;
	int type;
	int port;
	pthread_t thread;
};
```

```
int initEnergy(Energy *energy, int type);

int setSensor(Energy *energy, char *name, char *tension, char *current,
char* initiate);

void setTimer(Energy *energy, int sleep);

int startEnergy(Energy *energy);

int stopEnergy(Energy *energy);

void printEnergy(Energy *energy);

float getEnergy(Energy *energy, int interval);

float getCurrent(Energy *energy);

float getTension(Energy *energy);


int confPort(char *url);

void readSensorSerial(Energy *energy);

void startSensor(char *url);

float readSensor(char *url);

void *start(void *tmp);


int initEnergy(Energy *energy, int type){

      //energy = calloc(sizeof(Energy), 1);

      energy->type = type;

      energy->loop = 1;

      energy->sleep = 5;

      return 0;

}


int setSensor(Energy *energy, char *name, char *tension, char *current,
char* initiate){

      energy->name = name;

      energy->tensionSensor = tension;

      energy->currentSensor = current;

      energy->initiate = initiate;

      switch (energy->type){

      case 0://sensor interno

            startSensor(energy->initiate);
```

```
	break;
	case 1://porta serial
		energy->port = confPort(energy->initiate);
	break;
	}
	return 0;
}

void setTimer(Energy *energy, int sleep){
	energy->sleep = sleep;
}

int startEnergy(Energy *energy){
	int ret = pthread_create(&(energy->thread), NULL, start, energy);
	if (ret != 0) {
	ret = 1;
	}
	return ret;
}

int stopEnergy(Energy *energy){
	energy->loop = 0;
	if (energy->type == 1){
	close(energy->port);
	}
	return 0;
}

float getEnergy(Energy *energy, int interval){
	if (interval > 0){
	energy->power = 0;
	pthread_create(&(energy->thread), NULL, start, energy);
	sleep(interval);
	stopEnergy(energy);
```

```
	}
	return energy->power;
}

void printEnergy(Energy *energy){
	printf("%s=%f V; ", energy->name, energy->tension);
	printf("%s=%f A; ", energy->name, energy->current);
	printf("%s=%f W; ", energy->name, energy->instantPower);
	printf("%s Consumo=%f W ", energy->name, energy->power);
	printf("%f KWh\n", (energy->power /(1000*60)) );
}

float getCurrent(Energy *energy){
	return energy->current;
}

float getTension(Energy *energy){
	return energy->tension;
}

void *start(void *tmp){
	Energy *energy = (Energy *) tmp;
	char *tempo;
	while (energy->loop){
	switch (energy->type){
		case 0://sensor interno
			if (energy->tensionSensor != ""){
				energy->tension = readSensor(energy-
>tensionSensor);
			}
			if (energy->currentSensor != ""){
				energy->current = readSensor(energy-
>currentSensor);
			}
			if ((energy->tension > 0) && (energy->current > 0)){
```

```
                        energy->instantPower = energy->tension * energy-
>current;
                  }
            break;
            case 1://portaserial
                  readSensorSerial(energy);
                  energy->instantPower = energy->tension * energy-
>current;
            break;
      }
      sleep(energy->sleep);
      energy->power += energy->instantPower * energy->sleep;
      energy->time += energy->sleep;
      }
      pthread_exit(NULL);
}

void startSensor(char *url){
      FILE *arq;

      arq = fopen(url, "w");

      if(arq != NULL){
      fwrite("1", sizeof(char), 1, arq);
      fclose(arq);
      }
}

float readSensor(char *url){
      FILE *arq;
      float back;
      arq = fopen(url, "r+");

      if(arq != NULL){
      fscanf(arq,"%f\n", &back);
```

```
	fclose(arq);
	}
	return back;
}

void readSensorSerial(Energy *energy){
	char myStr[20];
	float f;

	if (energy->port == -1){
	printf("ErroReadserial \n");//system("error");
	} else {
	write(energy->port, energy->tensionSensor, 1);
	read(energy->port, myStr, 20);
	f = atof((char*) strtok(myStr, "#"));
	energy->current = f;

	f = atof((char*) strtok(NULL, "#"));
	energy->tension = f;
	}//close(port);
}

int confPort(char *url){
	int fd = 0; //arquivo da porta
	struct termios iniConfig, myConfig;// porta serial

	fd = open(url, O_RDWR | O_NOCTTY | O_NDELAY);
	if (fd == -1){
	printf("Erro ao abrir porta");
	close(fd);
	return fd;
	}
	else
	//ioctl(fd, F_SETFL, FNDELAY);//para nao esperar na porta seria
```

```
	fcntl(fd, F_SETFL, 0);

	//configura porta
	tcgetattr(fd, &iniConfig);
	tcgetattr(fd, &myConfig);

	//seta velocidade porta
	cfsetispeed(&myConfig, B9600);
	cfsetospeed(&myConfig, B9600);

	//checar paridade
	myConfig.c_cflag &= ~PARENB;
	myConfig.c_cflag &= ~CSTOPB;
	myConfig.c_cflag &= ~CSIZE;
	myConfig.c_cflag |= CS8;//8 bits dados

	myConfig.c_cflag &= ~HUPCL;
	myConfig.c_cflag &= ~CRTSCTS;

	myConfig.c_lflag = ICANON;

	tcsetattr(fd, TCSANOW, &myConfig);
	sleep(2);
	return (fd);
}
```

ANEXO B – UTILIZAÇÃO DA BIBLIOTECA ENERGY

```
#include "Energy.h"


int main(){
      int a = 0;


      Energy *energy0 = calloc(sizeof(Energy), 1);
      Energy *energy1 = calloc(sizeof(Energy), 1);
      Energy *energy2 = calloc(sizeof(Energy), 1);
      Energy *energy3 = calloc(sizeof(Energy), 1);
      Energy *energy4 = calloc(sizeof(Energy), 1);
      Energy *energy5 = calloc(sizeof(Energy), 1);


      initEnergy(energy0, 0);
      initEnergy(energy1, 0);
      initEnergy(energy2, 0);
      initEnergy(energy3, 0);
      initEnergy(energy4, 1);
      initEnergy(energy5, 1);


      setSensor(energy0, "A15", "/sys/bus/i2c/drivers/INA231/4-
0040/sensor_V", "/sys/bus/i2c/drivers/INA231/4-0040/sensor_A",
"/sys/bus/i2c/drivers/INA231/4-0045/enable");
      setSensor(energy1, "A7 ", "/sys/bus/i2c/drivers/INA231/4-
0045/sensor_V", "/sys/bus/i2c/drivers/INA231/4-0045/sensor_A",
"/sys/bus/i2c/drivers/INA231/4-0040/enable");
      setSensor(energy2, "MEM", "/sys/bus/i2c/drivers/INA231/4-
0041/sensor_V", "/sys/bus/i2c/drivers/INA231/4-0041/sensor_A",
"/sys/bus/i2c/drivers/INA231/4-0041/enable");
```

```
      setSensor(energy3, "GPU", "/sys/bus/i2c/drivers/INA231/4-
0044/sensor_V", "/sys/bus/i2c/drivers/INA231/4-0044/sensor_A",
"/sys/bus/i2c/drivers/INA231/4-0044/enable");
      setSensor(energy4, "DC ", "D", "", "/dev/ttyACM0");
      setSensor(energy5, "AC ", "A", "", "/dev/ttyACM0");


      startEnergy(energy0);
      startEnergy(energy1);
      startEnergy(energy2);
      startEnergy(energy3);
      startEnergy(energy4);
      startEnergy(energy5);
      while(a < 60){
      system("clear");
      printEnergy(energy0);
      printEnergy(energy1);
      printEnergy(energy2);
      printEnergy(energy3);
      printEnergy(energy4);
      printEnergy(energy5);
      a ++;
      sleep(5);
      }
      system("clear");
      printf("stop \n");
      stopEnergy(energy0);
      stopEnergy(energy1);
      stopEnergy(energy2);
      stopEnergy(energy3);
      stopEnergy(energy4);
      stopEnergy(energy5);
      sleep(5);


    printEnergy(energy0);
      printEnergy(energy1);
```

```
        printEnergy(energy2);
        printEnergy(energy3);
        printEnergy(energy4);
        printEnergy(energy5);
    }
```

ANEXO C - CLASSE DESENVOLVIDA NA IDE DO ARDUINO

```
#include "EmonLib.h"          // Emon Library: cálculos para os
sensores.


// VARIÁVEIS GLOBAIS

EnergyMonitor emon1;    // uma instância de um monitor de energial da
Emon Library.

const int sensorPin = A5;

//const int pin = A5;

int sensorValue_aux = 0;

float sensorValue = 0;

float currentValue = 0;

const float voltsporUnidade = 0.004887586;// 5%1023


void setup(){

  while (!Serial) {

    ; // wait for serial port to connect.

  }

  Serial.begin(9600);


  pinMode(sensorPin, INPUT);


  // Inicializa o monitor de energia.

  emon1.voltage(0, 315, 1.7);  // Voltage: input pin, calibration,
phase_shift

  emon1.current(1, 40);             // Calibração do sensor

}
```

```
void loop() {
  int leitura = Serial.read();
  if (leitura == 65){
    medicaoAC();
  }

  if (leitura == 68){
    medicaoDC();
  }
  delay(10);
}

void medicaoAC(){
   // Mede a corrente usando a biblioteca EmonLib e calcula potência.
  // Imprime dados na serial para depuração.
  emon1.calcVI(20, 2000);
  double Irms = emon1.calcIrms(1480);  // Mede a corrente RMS.

  double Potencia = Irms * emon1.Vrms;          // Calcula a potência
aparente (supondo que a rede elétrica esteja em 127 V).
  Serial.print(Irms, 3);            // Imprime a corrente na serial.
  Serial.print("#");          // separador
  Serial.print(emon1.Vrms);         // Imprime a corrente na serial.
  Serial.print("#");          // final
  Serial.println();
}

void medicaoDC(){
  //for(int i=100; i>0; i--){
  sensorValue_aux = (analogRead(sensorPin) - 511); // lê o sensor na
pino analógico A5
  sensorValue += pow(sensorValue_aux, 2); //somam os quadrados das
leituras.
  //}
```

```
  sensorValue = (sqrt(sensorValue / 1)) * voltsporUnidade; // finaliza o
cálculo da média quadrática e ajusta o valor lido para volts

  currentValue = (sensorValue / 0.100); // calcula a corrente
considerando a sensibilidade do sernsor (100 mV por amper)

  float voltage = readVcc()/1000;


  // mostra o resultado no terminal

  Serial.print(currentValue, 3);

  Serial.print("#");

  Serial.print(voltage);

  Serial.print("#");

  Serial.println();

  sensorValue = 0;

}


long readVcc() {

  #if defined(__AVR_ATmega32U4__) || defined(__AVR_ATmega1280__) ||
defined(__AVR_ATmega2560__)

    ADMUX = _BV(REFS0) | _BV(MUX4) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);

  #elif defined (__AVR_ATtiny24__) || defined(__AVR_ATtiny44__) ||
defined(__AVR_ATtiny84__)

    ADMUX = _BV(MUX5) | _BV(MUX0);

  #elif defined (__AVR_ATtiny25__) || defined(__AVR_ATtiny45__) ||
defined(__AVR_ATtiny85__)

    ADMUX = _BV(MUX3) | _BV(MUX2);

  #else

    ADMUX = _BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);

  #endif


  delay(2); // Wait for Vref to settle

  ADCSRA |= _BV(ADSC); // Start conversion

  while (bit_is_set(ADCSRA,ADSC)); // measuring


  uint8_t low  = ADCL;

  uint8_t high = ADCH;
```

```
  long result = (high<<8) | low;


  result = 1125300L / result; // Calcula Vcc (in mV); 1125300 =
1.1*1023*1000

  return result; // Vcc in millivolts

}
```