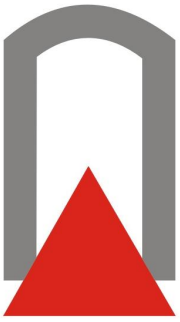
INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
MINAS GERAIS
Campus Avançado Conselheiro Lafaiete



UFSJ
UNIVERSIDADE FEDERAL
DE SÃO JOÃO DEL-REI

Internet das Vacas: Montagem de Placa de Protótipo de Dispositivo IoT para Localização Inteligente do Gado

Autores: Fernando A. Teixeira
João Victor Carvalho Tereza
Jonas Henrique Nascimento

2 de maio de 2019

Resumo

A fuga de animais de suas propriedades para estradas ou propriedades vizinhas é um dos problemas enfrentados por diversos fazendeiros. Uma possível solução para tal problema é o monitoramento da localização desses animais, para que uma medida a fim de proteger o gado seja tomada. Entretanto, as formas de monitoramento ofertadas pelo mercado, atualmente, são de alto consumo e custo, pois se baseiam em tecnologias não voltadas para esta aplicação em específico, tais como: rastreamento por satélite, GPS e por radiofrequência. Com o intuito de aprimorar as opções de mercado e apresentar uma solução ao problema, foram desenvolvidas duas propostas de monitoramento utilizando de uma tecnologia mais viável, com baixo consumo energético. Para tal, foram utilizados chips inteligentes que se comunicam entre si por meio de sinais de rádio Wi-Fi ou Bluetooth. A partir deste ponto, foram estabelecidos dois métodos de solução para o problema: a primeira visa informar ao fazendeiro se o animal está dentro ou fora de sua propriedade, enquanto a segunda visa mostrar a localização aproximada do animal no interior da fazenda, utilizando o conceito de localização por detecção de posição. Com os testes foram constatados precisão acima dos 90% para ambos os métodos. Para que a forma de monitoramento possua baixo consumo, foi preciso estudar as diferentes tecnologias e suas formas de emprego, para tal, foram realizados inúmeros testes controlados aferindo as diferentes formas de operação dos protótipos. Ademais, foram analisados numerosos modelos de bateria, com a finalidade de encontrar as que correspondiam melhor para seus protótipos. A partir dos dados coletados, foram estimados valores de autonomia de todo o projeto para cada modelo de monitoramento, demonstrando as vantagens e desvantagens de cada modo de operação. Sendo o melhor caso de autonomia próximo a seis meses, mas podendo ser acrescido, caso o produtor agrícola opte por uma bateria de maior porte e consequentemente, maior tamanho. Conclui-se, então, que o projeto pode ser uma opção viável aos fazendeiros como solução ao problema, pois apresenta boa precisão na localização do gado e apresenta alta portabilidade.

Palavras-chave: Internet - GPS - Monitoramento - Gado - Microcontroladores - Baterias

Sumário

1 Introdução

O agronegócio no Brasil possui caráter de grande envergadura para toda a economia do país. Somente em maio de 2017, as exportações atingiram US$ 9,68 bilhões, valor que corresponde a aproximados 13% de aumento em referência ao mesmo período do ano anterior. Somente o valor desse superávit comercial causou um aumento de 790 milhões de dólares, demonstrando que esse setor possui alta taxa de crescimento. Dentre parte das exportações, está contida o setor de carnes, com arrecadação em 2017, de 1,22 bilhão de dólares. (SANTANDER, 2017)

Todavia, mesmo com notório crescimento, muitos fazendeiros passam por inúmeras dificuldades para acompanhar seu gado, devido a sua ausência por problemas do cotidiano que simplesmente impedem a presença diária do fazendeiro para o acompanhamento. Devido a isso, surgem ocasiões que geram transtornos e podem gerar prejuízo, tais como perder vacas por terem fugido da propriedade, por ficarem atoladas, ou mesmo perder muito tempo procurando o gado em um determinado local sendo que o mesmo pode estar no outro extremo da região. Levando em consideração tais problemas, propõem-se formas de monitorar o gado à distância, para um melhor gerenciamento por parte dos fazendeiros.

A solução proposta por todo o projeto[1] visa realizar o monitoramento e gerenciamento dos animais à distância, usando de tecnologias especificas, tais como o IoT - Internet of Things, cuja tradução direta é "Internet das Coisas", microchips inteligentes como o ESP8266, ESP32, ATtiny13 dentre outros e protocolos de comunicações e de gerenciamento de sinais, como o Wi-Fi, Bluetooth Low Energy, RSSI e MQTT.

Os principais preceitos do IOT se baseiam na ligação entre alguma "coisa" física ao meio das comunicações de rede dinâmica e global, portando, dessa maneira, a capacidade de configurar de forma inteligente ou interagir com o objeto físico em questão. Para tal interfaceamento, utilizam-se sistemas eletrônicos pré-programados conectados em alguma rede, bem como na rede global. Esta comunicação pode se dar pelo uso do Wi-Fi ou do Bluetooth.

Os chips inteligentes utilizados detêm a função de controlarem a comunicação entre si, utilizando protocolos de comunicações específicos, e gerenciar os dados colhidos. Para isso, serão pré-programados afim de cumprirem com suas respectivas funções.

Para o amplo emprego dessas tecnologias, é preciso viabilizar algumas características fundamentais no sistema, sendo estes o tamanho do projeto final, o custo e a autonomia, que é definida como o período máximo que o circuito poderá ser mantido em constante funcionamento sem apresentar falhas. Para se alcançar um bom valor, foram empregadas as mais recentes formas de tecnologia de baixo consumo disponíveis no mercado, que consistiu no emprego de Microcontroladores específicos e de modos de comunicação aplicados ao baixo consumo, além do aprimoramento de técnicas justapostas que relacionam dois ou mais modos de operação para uma combinação satisfatória de baixo consumo.

Um desafio no que diz respeito ao emprego desse sistema está justamente em cumprir com uma boa viabilidade o tamanho e o custo final do sistema eletrônico, sem que se perca a autonomia necessária para o funcionamento do código fonte[2], que será desenvolvido paralelamente pela UFSJ – Universidade Federal de São João Del-Rei. Será considerado um tamanho que caiba em uma etiqueta utilizada pelos fazendeiros para a identificação de cada animal, como mostrado na figura 1, a qual é presa em suas respectivas orelhas. Seguidamente, será analisado o custo do protótipo, a qual recorre

[1] O projeto completo constitui-se de duas partes separadas, que serão desenvolvidas em paralelo. A primeira parte é a que constitui o código fonte de todo o sistema de monitoramento e gerenciamento. A segunda parte constitui-se do Hardware envolvido nos nós da rede, nos estudos referentes aos modelos de bateria e nos diferentes modos de operação envolvidos no estudo do consumo do circuito.

[2] O código fonte resume-se nas operações pré-programadas que o circuito eletrônico fará atuando no meio físico.

da compra de basicamente dois principais componentes: o processador utilizado e a bateria escolhida para alimentar o circuito.

Figura 1: Modelo de etiqueta.

Levando-se esses problemas em consideração, os Microcontroladores escolhidos para a realização dos estudos foram os referentes à família ESP e a família ATtiny, sendo seus principais polos o ESP8266, o ESP32 e o ATtiny13A-PU. Todos foram escolhidos por apresentarem alto desempenho em eficiência energética. Ademais, esses Microcontroladores apresentam a portabilidade de modelos de comunicação distintos, de modo que o ESP8266 possa utilizar o Wi-Fi, o ESP32 possa utilizar o BLE - Bluetooth Low Energy, e, por fim, o ATtiny13 possui compatibilidade com módulos de rádio frequência, como o NRF24l01.

Com tais Microcontroladores, pode-se contornar o problema de custo e autonomia, visto que se apresentam de fácil acesso e apresentam tecnologias já inclusas de baixo consumo. Além disso, apresentam boa compatibilidade com o tamanho total do projeto, facilitando a instalação na etiqueta.

Após a escolha dos referidos Microcontroladores, foram realizadas inúmeras análises de diversos modelos de baterias, com a finalidade de se obter o modelo que melhor atende às necessidades de tamanho, custo e capacidade energética, para acréscimo da autonomia.

Mais especificamente, nesse projeto será montado um protótipo operacional de um dos nós da rede de integração de monitoramento do gado, utilizando um dos dois chips ESP, com o intuito de chegar a possíveis soluções para o projeto. Nas Figuras 2 e 3, apresentam-se as versões que servirão como base para os futuros protótipos, levando em consideração os modelos de bateria da mesma proporção, além dos pequenos componentes externos necessários para o funcionamento do circuito.

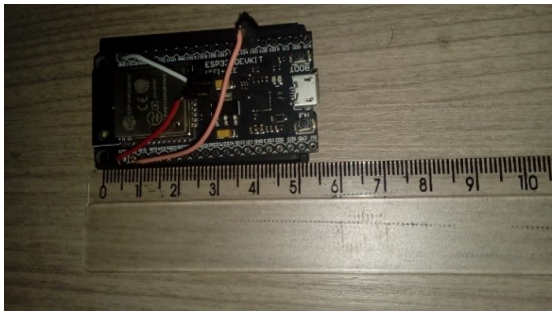


Figura 2: Modelo de protótipo utilizando o ESP32

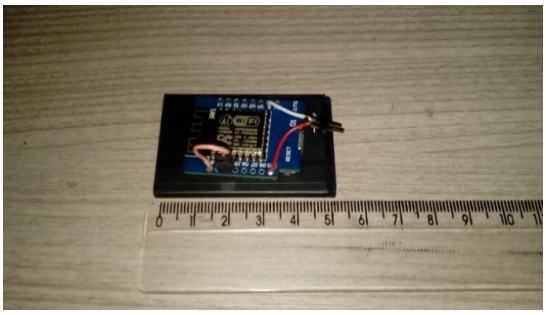


Figura 3: Modelo de protótipo utilizando o ESP8266

2 Objetivos

Para apresentar uma nova forma de solução dos problemas enfrentados pelos fazendeiros, o objetivo é integrar múltiplas tecnologias, se apropriando do uso do IoT em conjunto com outros sistemas, tais como RSSI, Wi-Fi, Bluetooth Low Energy, criando um conjunto que possibilite ao fazendeiro gerenciar e localizar seu gado.

Esse sistema seria composto por antenas fixas espalhadas na área a ser monitorada e pela placa de transmissão acoplada na etiqueta de cada animal. No momento em que a placa acoplada transmite sinal para as antenas, o sinal seria processado e então serial calculada a sua localização aproximada por meio da trilateração, e esses dados ficariam dispostos ao agricultor.

A figura 4 esboça de forma simplificada o sistema de transmissão de dados entre uma vaca com a placa desenvolvida acoplada em sua etiqueta de identificação entre duas antenas fixas. A partir dos cálculos realizados o fazendeiro poderá saber a localização aproximada da vaca em sua fazenda.

Figura 4: Modelo simplificado projeto

3 Material e Métodos

Inicialmente, foi realizada uma pesquisa dos diversos modelos e tipos de bateria para análise, com a finalidade de se optar por aquela que tenha maior eficiência nos critérios já mencionados (tamanho, custo e carga energética). Após a aferição de mais de 200 modelos diferentes, foram realizados vários filtros para a seleção destes, resultando na Tabela 1 que possui vinte e oito modelos.

Tabela 1: Tabela modelos de baterias

nº	Marca	Modelo	Química	Dimensões(mm)			Tamanho	Preço	Tensão	mAh	Wh	Custo/wh
				C	L	A						
01	Rontek	RT300AAAB4	Ni-cd	11	44	11	Aaa	R$4,98	1,20	300,00	360,00	0,009722
02	Energy Power	AA Ni-mh	Ni-mh	14,5	50,5	14,5	Aa	R$8,90	1,20	800,00	960,00	0,009271
03	Energy Power	AA Ni-cd	Ni-cd	14,5	50,5	14,5	Aa	R$9,50	1,20	1000,00	1.200,00	0,007917
04	Rontek	AA Ni-mh	Ni-mh	14,5	50,5	14,5	Aa	R$7,50	1,20	2100,00	2.520,00	0,002976
05	Mox	Aaa	Ni-mh	14,5	50,5	14,5	Aa	R$3,80	1,20	2700,00	3.240,00	0,001172
06	Knup	KP-BT9V	Ni-mh	47	20	15	Bat P	R$12,00	9,00	450,00	4.050,00	0,002963
07	FLEX	FX-45B1	Ni-mh	47	20	15	Bat P	R$28,00	9,00	450,00	4.050,00	0,008642
08	FullyMax	-	LIPO	9,5	26	45	Lipo M	R$15,20	3,70	650,00	2.405,00	0,006320
09	Mox	MO-086B	Ni-cd	31,5	44	10,5	Aaa	R$19,00	3,60	700,00	2.520,00	0,001428
10	Rontek	6RT1800SC-CX	Ni-cd	131	51	23	Bat. G	R$54,04	7,20	1800,00	12.960,00	0,004169
11	Rontek	6RT3000SC-CX	Ni-mh	131	51	23	Bat. G	R$100,14	7,20	3000,00	21.600,00	XXXX
12	Rontek	6LR61	Ni-mh	48	26	16	Bat. P	R$18,50	8,40	350,00	2.940,00	XXXX
13	Rontek	-	Ni-mh	2	16	16	P. Botão	R$5,15	3,60	80,00	288,00	XXXX
14	Rontek	-	Ni-mh	42	14	47	4 * Aaa	R$9,86	3,60	1300,00	4.680,00	XXXX
15	Rontek	-	Ni-cd	17	51	57	3 * aa	R$36,85	7,20	600	4.320,00	XXXX
16	FullyMax	-	LIPO	7	20	36	Lipo P	R$14,40	3,70	350,00	1.295,00	0,011119
17	minamoto	LFP803048	LiFePO4	8	30	50	Lipo M	Orçamento	3,20	800	2.560,00	XXXX
18	minamoto	LFP603450	LiFePO4	6	34	50	Lipo M	Orçamento	3,20	700	2.240,00	XXXX
19	minamoto	LFP101945HP	LiFePO4	10	19	45	Lipo M	Orçamento	3,20	440	1.408,00	XXXX
20	minamoto	LFP803048HP	LiFePO4	8	30	48	Lipo M	Orçamento	3,20	800	2.560,00	XXXX
21	minamoto	LFR26650E	LiFePO4	26	65	26	D+	Orçamento	3,20	3300	10.560,00	XXXX
22	minamoto	LFR18650E	LiFePO4	18,2	64,5	18,2	D+	Orçamento	3,20	1500	4.800,00	XXXX
23	minamoto	LFR18490E	LiFePO4	18,2	48,5	18,2	Aa	Orçamento	3,20	1000	3.200,00	XXXX
24	minamoto	LFR14500E	LiFePO4	14,1	48,5	14,1	Aa	Orçamento	3,20	500	1.600,00	XXXX
25	minamoto	LFR18650P	LiFePO4	18,2	64,5	18,2	D+	Orçamento	3,20	1100	3.520,00	XXXX
26	minamoto	LFR26650P	LiFePO4	26	65	26	D+	Orçamento	3,20	2300	7.360,00	XXXX
27	minamoto	LP104884	LIPO	10	48	84	-	Orçamento	3,7	5000	18.500,00	XXXX
28	FullyMax	-	LIPO	10	26	45	Lipo M	27,20	3,7	800	2.960,00	0,00125

Logo após a separação desses vinte e oito modelos, foram realizadas novas pesquisas para delimitar possíveis características que filtrassem novamente os mesmos. Foram consultados os vários Datasheets[3] referentes aos modelos de ESP e dos modos de funcionamento, chegando a especificamente cinco modelos que se apresentam como possíveis escolhas finais. Essas baterias serão adquiridas para a realização de experimentos práticos, para assim se chegar a uma conclusão definitiva.

De modo concomitante, para se chegar a um valor confiável do melhor modelo de bateria, foram realizados diversos testes experimentais, todavia, para efetuar tais testes foi feita uma placa shield, figura 5, para a realização desses experimentos. Foi projetado um circuito, cujo esquemático feito no software Proteus (ver figura 6). Este, possui 8 leds que estão ligados em current source com 8 pinos digitais da placa WeMos, que serão controlados por um código fonte anteriormente programado. A placa foi projetada para drenar uma corrente fixa em cada digital I/O do ESP, para assim, aferir o consumo de corrente do chip.

[3] Datasheet é um arquivo digital a qual contém diversas informações técnicas do fabricante sobre um dado componente eletrônico. Sempre as informações mais confiáveis são fornecidas por esses arquivos.

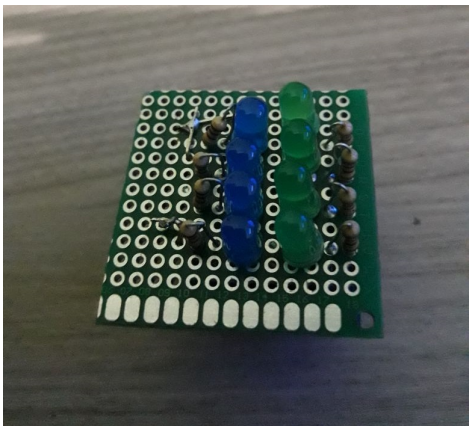


Figura 5: Placa desenvolvida para aferição de consumo do ESP8266

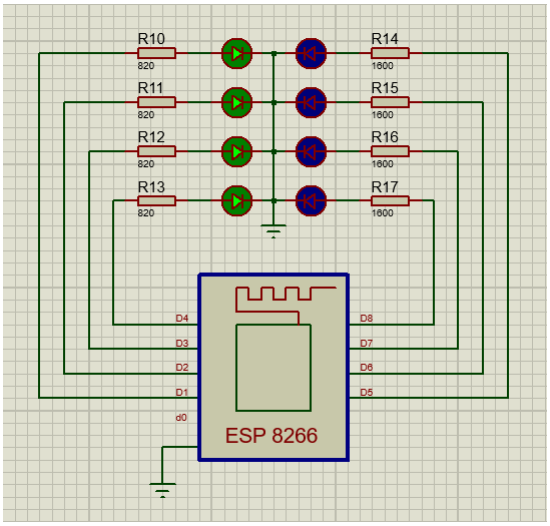


Figura 6: Esquemático desenvolvido no proteus - Shield 1

Após os primeiros testes de consumo do ESP, foram desenvolvidos diferentes códigos para que se possa aferir o consumo de energia por cada chip em cada modo de operação, modo de transmissão de dados e em cada modo de 'Sleep[4]'. Cada um desses códigos foi desenvolvido utilizando o Arduino, software IDE, sendo que cada um foi programado utilizando a linguagem C++. Após esta etapa, os códigos foram armazenados, junto aos demais arquivos do projeto, na plataforma Git Hub, com o nome "W8jonas".

Ao final da etapa de programação e de aferição do consumo de corrente pela shield 1. Foi feita uma segunda shield, (ver figura 7), capaz d alterar entre os diferentes modos de funcionamento por meio de um botão. O códigoe fonte inserido na placa ESP permite a troca de funções exercidas pelo ESP, desse modo, a cada pressionar do botão, uma nova forma de funcionamento é estabelecida.

O código tem como objetivo estabelecer 6 funções diferentes para o Arduino executar cada uma delas de modo separado. Esse programa apresenta dois principais núcleos de funcionamento, a leitura do pressionar do botão, cuja a lógica é compreendida entre as chaves do comando void setup() , e a parte da alteração das funções, em cada função declarada demonstra um modo de funcionamento

[4]Sleep é o nome técnico dado ao período em que o processador não realiza grandes funções, como contas aritméticas ou transmissão de dados. Estes períodos de Sleep são utilizados basicamente para se poupar energia, visto que quando o processador entra neste modo, ele não realiza nenhuma operação que demande grande consumo de energia.

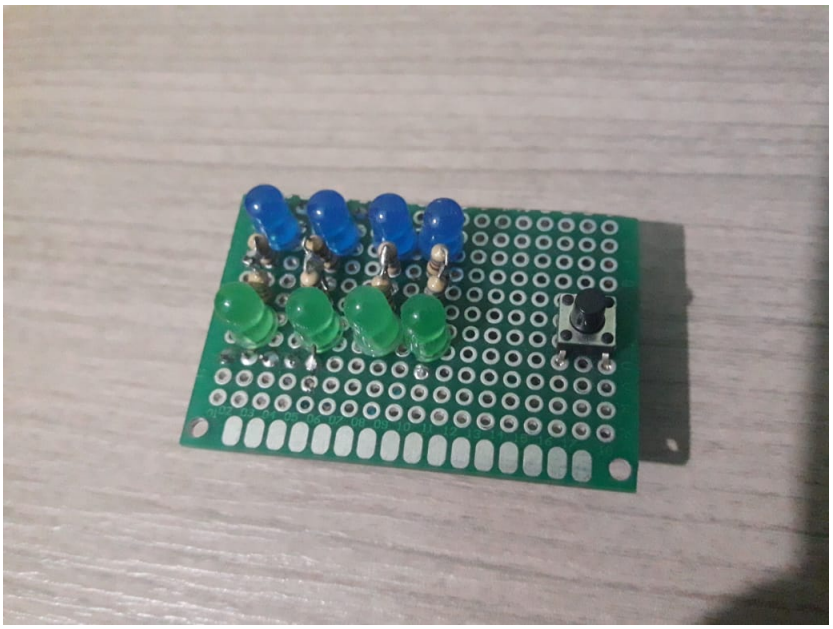


Figura 7: Placa desenvolvida para controlar as funções do ESP

diferente.
O código fonte em específico encontra-se na íntegra abaixo:

```
1
2  //  Internet das Vacas código 3
3  //  Autor: Jonas Henriquee Nascimento
4  //  PIBIC-Junior
5  //
6  //  Data de início: 04/06/2018
7  //  Data da ultima atualização: 19/06/2018
8  //  Data de término: 08/06/2018
9  //
10 //  O código tem como objetivo estabelecer 6 funções diferentes para o
       arduino
11 //  executar cada uma delas de moto paralelo. Essas funções são
       escolhidas através
12 //  de um botão que toda que apertado faz com que o código execute a
       função seguinte
13 //  A primeira função, que é executada junto ao ESP quando é ligado deixa
        todos os
14 //  LEDs desligados e o ESP em modo de standby. A segunda função, liga
       somente um
15 //  dos LEDs. A terceira, por sua vez, liga todos os 8 LEDs. A próxima
       função
16 //  executa uma série de operações aritméticas, com todos os LEDs
       desligados. Já a
17 //  função 5 executa as mesmas operações aritméticas, mas com 1 dos LEDs
       ligados.
18 //  De tal forma é feito na 6 função, no qual são executadas as operações
        matemáticas,
19 //  mas com todos os LEDs ligados.
```

```
20
21 //  Este código está disponível sempre no endereço abaixo, para livre
       aperfeiçoamento.
22 //  Todavia, pede-se por educação, que ao compartilharem o código,
       mantenham os autores
23 //  originais, tão bem quanto o nome da instituição.
24 //  https://github.com/W8jonas/Internet-das-Vacas/blob/master/programacao
       /codigo_modos_de_operacao/codigo_modos_de_operacao.ino
25
26
27
28
29 #define Output_1 D8
30 #define Output_2 D7
31 #define Output_3 D6
32 #define Output_4 D5
33 #define Output_5 D4
34 #define Output_6 D3
35 #define Output_7 D2
36 #define Output_8 D1
37 #define entrada_botao D0
38
39 void funcao__();
40 void funcao_0();
41 void funcao_1();
42 void funcao_2();
43 void funcao_3();
44 void funcao_4();
45
46 int operacao = 0;
47 boolean leitura = true;
48
49 void setup() {
50    pinMode (Output_1, OUTPUT);
51    pinMode (Output_2, OUTPUT);
52    pinMode (Output_3, OUTPUT);
53    pinMode (Output_4, OUTPUT);
54    pinMode (Output_5, OUTPUT);
55    pinMode (Output_6, OUTPUT);
56    pinMode (Output_7, OUTPUT);
57    pinMode (Output_8, OUTPUT);
58    pinMode (entrada_botao, INPUT);
59    Serial.begin(115200);
60 }
61
62 void loop() {
63   leitura = digitalRead(entrada_botao);
64   Serial.println(leitura);
65   if (leitura == LOW ){
66      operacao++;
67      delay(500);
```

```
  }

  funcao_2();

}

void funcao__ (){
   Serial.println("funcao 00 ");
   digitalWrite(Output_1, LOW);
   digitalWrite(Output_2, LOW);
   digitalWrite(Output_3, LOW);
   digitalWrite(Output_4, LOW);
   digitalWrite(Output_5, LOW);
   digitalWrite(Output_6, LOW);
   digitalWrite(Output_7, LOW);
   digitalWrite(Output_8, LOW);
}

void funcao_0() {
   Serial.println("funcao 0 ");
   digitalWrite(Output_1, HIGH);
}

void funcao_1() {
   Serial.println("funcao 1 ");
   digitalWrite(Output_1, HIGH);
   digitalWrite(Output_2, HIGH);
   digitalWrite(Output_3, HIGH);
   digitalWrite(Output_4, HIGH);
   digitalWrite(Output_5, HIGH);
   digitalWrite(Output_6, HIGH);
   digitalWrite(Output_7, HIGH);
   digitalWrite(Output_8, HIGH);
}

void funcao_2() {
   Serial.println("funcao 2 ");
   digitalWrite(Output_1, LOW);
   digitalWrite(Output_2, LOW);
   digitalWrite(Output_3, LOW);
   digitalWrite(Output_4, LOW);
   digitalWrite(Output_5, LOW);
   digitalWrite(Output_6, LOW);
   digitalWrite(Output_7, LOW);
   digitalWrite(Output_8, LOW);
   int cont = 1;
   float resp = 1;
   float resp2 = 1;
   for(int AA = 0; AA < 500; AA++){
      resp = 3 + sin(resp)/cos(resp*resp/2) * sqrt(sqrt(resp*resp));
      resp = resp * 0.5;
```

```
        resp2 = sqrt(AA);
        yield();
    }
}

void funcao_3() {
    Serial.println("funcao 3 ");
    digitalWrite(Output_1, HIGH);
    int cont = 1;
    float resp = 1;
    float resp2 = 1;
    for(int AA = 0; AA < 500; AA++){
        resp = 3 + sin(resp)/cos(resp*resp/2) * sqrt(sqrt(resp*resp));
        resp = resp * 0.5;
        resp2 = sqrt(AA);
        yield();
    }
}

void funcao_4() {
    Serial.println("funcao 4 ");
    digitalWrite(Output_1, HIGH);
    digitalWrite(Output_2, HIGH);
    digitalWrite(Output_3, HIGH);
    digitalWrite(Output_4, HIGH);
    digitalWrite(Output_5, HIGH);
    digitalWrite(Output_6, HIGH);
    digitalWrite(Output_7, HIGH);
    digitalWrite(Output_8, HIGH);
    int cont = 1;
    float resp = 1;
    float resp2 = 1;
    for(int AA = 0; AA < 500; AA++){
        resp = 3 + sin(resp)/cos(resp*resp/2) * sqrt(sqrt(resp*resp));
        resp = resp * 0.5;
        resp2 = sqrt(AA);
        yield();
    }
}
```

Além deste código, foram desenvolvidos outros códigos para o funcionamento de outros modos tratamento de transmissão de sinais, incluindo, também, diferentes formas de economia de energia. Novamente, utilizou-se do pressionar do botão para a alteração entre as funções relativas a cada estado de funcionamento do ESP. Todos os demais códigos feitos encontram-se nos Apêndices deste documento.

Nesse momento, foram realizadas medidas de consumo de corrente, para isso foram utilizados três multímetros de marcas e modelos diferentes, sendo suas aferições relatadas em maior valor lido e menor valor lido. A tensão foi medida em um resistor shunt de 1 Ω em série com o protótipo de teste de consumo. Após a coleta das seis medidas, foi calculada a média aritmética para se chegar ao resultado final de consumo. A tabela 2 apresenta, sucintamente, os valores obtidos, todos os valores medidos, incluindo todas as características dos modos de funcionamento estão dispostos na tabela apresentada

no apêndice A.

Tabela 2: Tabela resumo de consumo ESP8266

Modo de operação	Configurações do modo	Consumo médio (mA)
Standby	VCC = 5v	75,333
1 Led ligado	VCC = 5v	75,583
Todos os Leds ligados	VCC = 5v	79,3
100% uso do CPU	VCC = 5v	76,9
1 Led ligado + 100% CPU	VCC = 5v	77,783
Todos os Leds + 100% CPU	VCC = 5v	80,333
ESP server e cliente ligado	VCC = 5v	75,366
ESP only client	VCC = 5v	71,983
Modem Sleep	VCC = 5v	71,933
Light Sleep – CPU ativa	VCC = 5v	16,4
Light Sleep – CPU desativada	VCC = 5v	2,23
Deep Sleep	VCC = 5v	0,133
	VCC = 3.3v	0,011
Transmit 802.11b	VCC = 5v e POUT = +20.5dBm	75,233
	VCC = 5v e POUT = +14dBm	74,866
Transmit 802.11g	VCC = 5v e POUT = +20.5dBm	74,050
	VCC = 5v e POUT = +14dBm	71,116
Transmit 802.11n	VCC = 5v e POUT = +20.5dBm	71,433
	VCC = 5v e POUT = +14dBm	71,250

Após os valores aferidos, foi feita uma análise de autonomia em relação ao modelo de bateria e aos modos de funcionamento do ESP8266. O padrão de comportamento do chip se apresenta de maneira análoga em todos os casos de análise, pois o funcionamento se baseia em um tempo com o processador ligado, um período com o transmissor de sinais ligado, e um período de Sleep. Esse padrão foi definido dessa forma porque apresenta melhor rendimento.

Inicialmente, após a coleta dos dados de consumo, foram criadas equações matemáticas que estimam, de modo aproximado, o gasto energético da placa em diferentes modos de funcionamento com a carga energética das baterias selecionadas, calculando, dessa forma, os valores de autonomia do protótipo. O Gráfico 8 mostra a relação dos modos de operação selecionados e sua autonomia, tomando como exemplo uma bateria de carga energética igual a 650 mAh.

Cada modo de operação do Chip possui vantagens e desvantagens, que serão aprimoradas posteriormente junto à codificação final do projeto. Vale salientar que a principal diferença entre os modos está na velocidade dos rastreamentos realizados, sendo que quanto menor o tempo de repetição, mais precisa será a localização. Caso o tempo fosse zero, o monitoramento seria considerado em tempo real, o que se torna possível, mas não viável, pelo alto custo de se manter este sistema funcionando por um longo período.

A partir das análises preliminares do gráfico, é possível destacar a diferença entre a melhor curva e a pior curva de consumo, destacando, dessa forma, como a pouca diferença entre algumas variáveis altera o resultado final. De certo modo, é impossível que se tenha uma precisão de 100% no valor de autonomia, isso se deve a dois principais motivos.

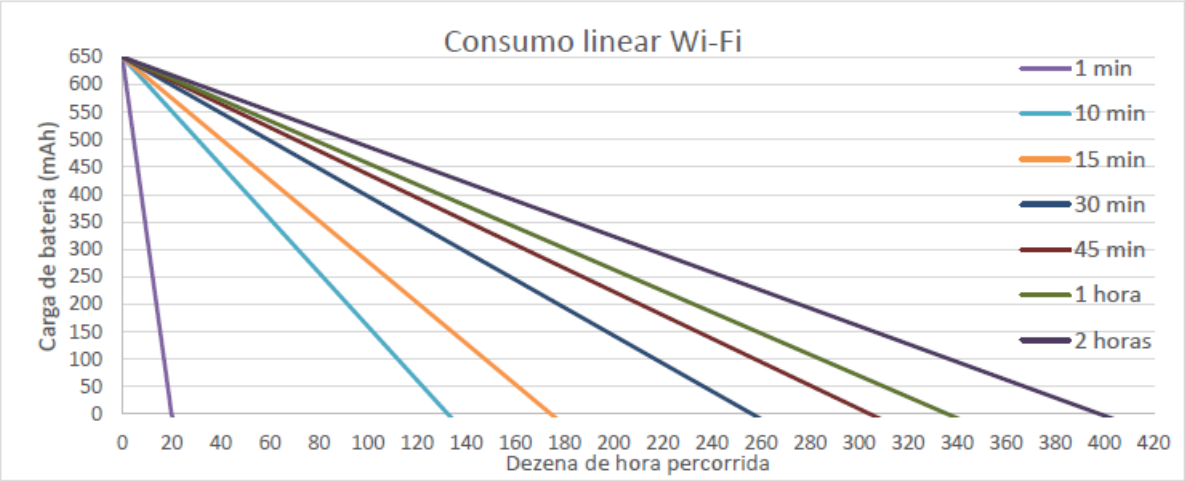


Figura 8: Gráfico consumo linear protótipo 1

O primeiro fator se deve a curva de descarga da bateria, a qual não é inteiramente linear, apresentando em seu início e fim curvas exponenciais, dificultando, dessa maneira, os cálculos precisos com relação a descarga da bateria. Essas curvas estão diretamente ligadas aos tipos químicos de cada bateria e os diferentes modelos produzidos por cada empresa da área.

O segundo fator direciona-se ao fato que os próprios chips possuem variantes internos, estes que por sua vez variam naturalmente, além de serem susceptíveis a variações externas como temperatura. Logo, com variações externas, soma-se as variações resultantes do código fonte executado pelo ESP8266, de suas contas e de suas variações com o decorrer do tempo. Ademais, soma-se as variações de corrente consumida pelo chip em consequência das variações de tensão fornecida pela bateria, visto que esta, ao passar do tempo, tende a diminuir.

Após os primeiros testes experimentais, foi projetado e montado um circuito capaz de analisar em tempo real os valores de tensão da bateria em decorrer do funcionamento dos chips ESPs em diferentes modelos de bateria. O experimento foi realizado para que se possa aferir divergências entre os valores obtidos teoricamente e os obtidos no funcionamento dos circuitos.

O funcionamento deste circuito, rotulado de “Circuito datalogger de tensão”, tem como função ler, a cada minuto, o valor de tensão de cada uma das baterias dispostas no circuito que alimentam determinado ESP e armazenar os valores em um arquivo do tipo ‘.svc’ para a análise em algum programa.

Inicialmente foram gravados os mesmos códigos com suas adaptações tanto para o ESP32 quando para o ESP8266, no qual eram responsáveis por operar o protótipo de modo a seguir o padrão de consumo/funcionamento referente a tabela 3.

Tabela 3: Tabela modo de funcionamento

Especificações:		Valor	Unidade
	bateria	750	mAh
	bateria	2700000	mAsec

Teste (nº)	Running	Operação	Consumo (mA)	Tempo de operação (s)	Consumo (mAsec)	Consumo por ciclo	Consumo por hora (mAh)	Número de ciclos possíveis	Duração total	
									Em horas	Em dias
1	Valor X Valor Y Valor W	Deep Sleep 100% uso do CPU Transmit 802.11n POUT = + 20.5dBm	0,133 76,900 71,433	100,00 20,00 5,00	13,30 1538,00 357,16	1908,46	15,26	1414,74	49,1	2,05
2	Valor X Valor Y Valor W	Deep Sleep 100% uso do CPU Transmit 802.11n POUT = + 20.5dBm	0,133 76,900 71,433	100,00 200,00 200,00	13,30 15380,00 14286,60	29679,90	59,36	90,97	12,6	0,53

O circuito teve como centro o Arduino Mega, a qual serviu de ponte entre a leitura de tensão de cada bateria ligada à um diferente ESP e o armazenamento dessas leituras em um cartão de memória. Para a leitura e o registro serem feitos com sucesso e uma coerente exatidão, foram utilizados dois módulos ao Arduino Mega, o módulo 'Cartão SD' e o módulo 'RTC – Real Time Clock'. Todo seu esquemático pode ser observado na figura 9. Todavia, seu esquemático se apresenta em um nível alto de abstração, visto que os componentes foram representados por meros blocos gráficos, porém, assim, não interferindo na análise do circuito, a partir de seu esquemático.

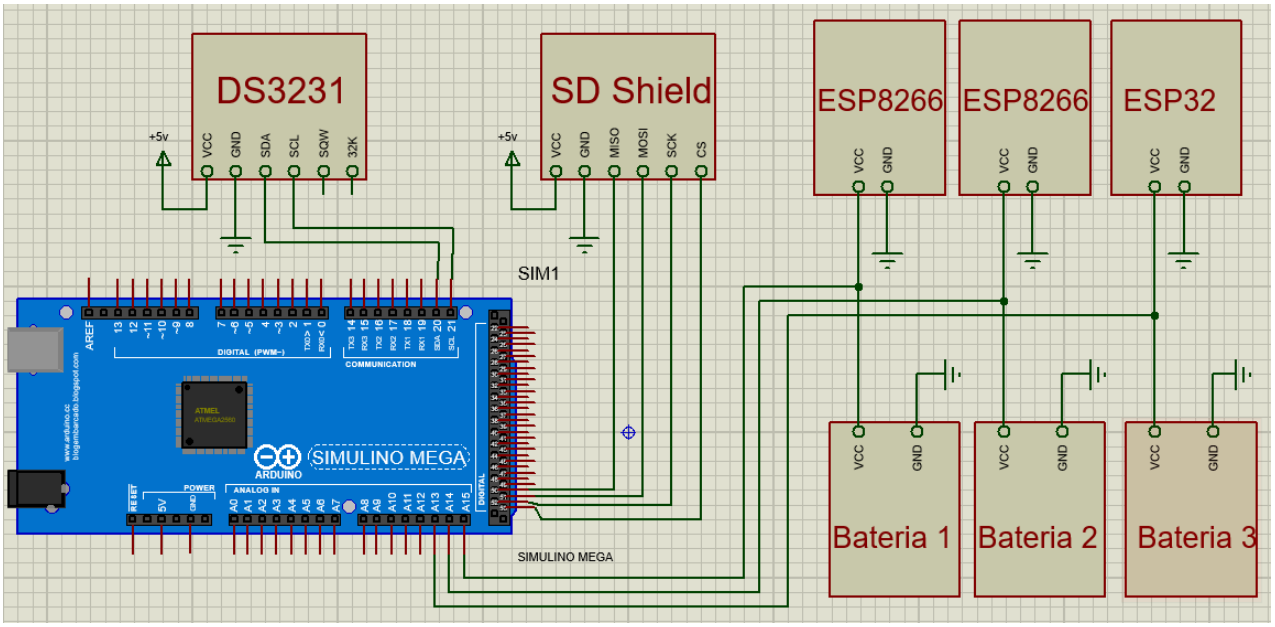


Figura 9: Esquemático do circuito Datalogger de tensão

Cada um dos módulos teve suas funções bem definidas. O 'Cartão SD' foi utilizado para armazenar todos os dados colhidos das leituras, contendo a data e hora da medida, que foram extraídos do RTC. As gravações foram feitas em um arquivo de nome "TENSOEX.SVC", onde 'X' representa o número do teste. Como por exemplo, o teste número 3 teve nome igual a: "TENSOE3.SVC".

Dentro desse arquivo, foram armazenados os dados separados por um caractere indicador, nos primeiros testes, foi escolhido o caractere ', ', porém, após algumas análises, o caractere escolhido foi substituído para '; '. Isso foi feito para que, após o termino das gravações, o arquivo possa ser aberto utilizando algum software para tabulações, o mais conhecido, e que foi utilizado é o Microsoft Excel. A partir dele, é possível abrir esse arquivo e colocar cada valor lido separado em uma célula para posteriormente trabalhar com alguma aplicação no Excel, como a construção de gráficos comparativos.

Os dados foram mantidos em uma hierarquização de ordem, em outras palavras, todos os dados foram sempre armazenados na mesma ordem, com a finalidade de se manter um padrão e facilitar a aplicabilidade em diversos programas de tabulação. A ordem escolhida para a escrita no cartão de memória foi: data, hora, valor do sensor 1, valor do sensor 2, erro 1, erro 2, ||; Dessa forma, os dados foram gravados como demonstrado na tabela 4:

Logo, transcrevendo as informações, temos os dados contidos na tabela 5:

A cada nova leitura de tensão nas baterias, é escrito uma nova linha, logo abaixo do anterior, com a exata mesma sintaxe. Todavia, com seus dados atualizados com os novos valores referentes a leitura.

Todo o código fonte pode ser observado abaixo:

Tabela 4: Valores gravados no cartão para análise

11.09.2018; 16:56:00; 3.51; 3.57; 3; 2; ||;
11.09.2018; 16:57:00; 3.51; 3.56; 3; 2; ||;
11.09.2018; 16:58:00; 3.50; 3.55; 4; 2; ||;
11.09.2018; 16:59:00; 3.50; 3.54; 4; 2; ||;
11.09.2018; 17:00:00; 3.50; 3.53; 4; 2; ||;

Tabela 5: Dados contidos na gravação do cartão para análise

Data: 11.09.2018
Horas: 16:56:00
Valor do sensor 1: 3.51
Valor do sensor 2: 3.57
Erro 1: 4
Erro 2: 2
|| : Caractere sinalizador de fim de linha.

```
1
2  /*
3   * Instituto Federal de Educação, Ciência e Tecnologia Minas Gerais
4   * IFMG - Campus Avançado Conselheiro Lafaiete
5   *
6   * Internet das Vacas código monitoramento de tensão
7   * Autor: Jonas Henrique Nascimento
8   * PIBIC-Junior
9   *
10  * Data de início.................: 06/09/2018
11  * Data da ultima atualização.....: 23/09/2018
12  * Data de atualização de versão..: 23/09/2018
13  * Data de término...............: 23/09/2018
14  *
15  *
16  *
17  *  Este código está disponível sempre no endereço abaixo, para livre
        aperfeiçoamento.
18  *  Todavia, pede-se por educação, que ao compartilharem o código,
        mantenham os autores
19  *  originais, tão bem quanto o nome da instituição.
20  *
21  *  https://github.com/W8jonas/Internet-das-Vacas/blob/master/programacao
        /Monitoramento_de_tensao_2.1/Monitoramento_de_tensao_2.1.ino
22  *
23  */
24
25
26 #include <DS3231.h>
27 #include <SPI.h>
28 #include <SD.h>
29 #define chip_select 4
```

```
30
31
32  int pino_sensor_de_tensao_ESP32 = A0;              // pino do arduino
       utilizado para medir a tensao da bateria ligada ao ESP32
33  float valor_do_sensor_de_tensao_ESP32 = 0;         // variavel de ponto
       flutuante que armazena o valor de tensao lido do ESP32
34                                                     //
35  int pino_sensor_de_tensao_ESP8266 = A8;            // pino do arduino
       utilizado para medir a tensao da bateria ligada ao ESP8266
36  float valor_do_sensor_de_tensao_ESP8266 = 0;       // variavel de ponto
       flutuante que armazena o valor de tensao lido do ESP8266
37
38
39  int pino_sensor_controle_ESP32 = A1;               // Valor lido de tensao
        para converter em estado de HIGH ou LOW
40  float valor_do_sensor_de_controle_ESP32 = 0;       // Valor armazenado de
       tensao
41                                                     //
42  int pino_sensor_controle_ESP8266 = A9;             // Valor lido de tensao
        para converter em estado de HIGH ou LOW
43  float valor_do_sensor_de_controle_ESP8266 = 0;     // Valor armazenado de
       tensao
44
45
46  unsigned int minuto_antigo = 0;
47  unsigned int minuto_atual = 0;
48  unsigned int marcador_tempo_1 = 0;
49  unsigned int marcador_tempo_2 = 0;
50
51  bool flag = false;
52  bool flag2 = false;
53  bool flag_controle_marcador_ON = false;
54  bool flag_controle_marcador_OFF = false;
55  bool flag_ligado = false;
56
57  String texto_marcador = "string marcador";
58  String dados = "string dados";
59  String condicao_ = "condicao";
60
61  unsigned char erro_marcador_1 = 0;
62  unsigned char erro_marcador_2 = 0;
63
64  void leitura();
65  void marcador(String marcador);
66  void gravar_dados_cartao();
67
68  DS3231  rtc(SDA, SCL);
69  Time  t;
70  File datalogger;
71
72  void setup() {
```

```
73
74      Serial.begin(115200);
75      rtc.begin();
76      while (!Serial) {;}
77
78      if (!SD.begin(chip_select)) {
79        Serial.println("Erro ao ler cartao de memoria");
80        return;
81     }
82
83  }
84
85
86  void loop() {
87      t = rtc.getTime();
88      minuto_atual = t.min;
89
90      if ( digitalRead(10) == HIGH) {
91         condicao_ = "LIGADO";
92         marcador(condicao_);
93      } else {
94         condicao_ = "DESLIGADO";
95         marcador(condicao_);
96      }
97
98      if ( minuto_antigo != minuto_atual ){
99         minuto_antigo = minuto_atual;
100        leitura();
101        dados = texto_marcador + rtc.getDateStr() + ";" + rtc.getTimeStr()
              + ";" + valor_do_sensor_de_tensao_ESP8266 + ";" +
              valor_do_sensor_de_tensao_ESP8266 + ";" + erro_marcador_1 + ";"
              + erro_marcador_2 + ";*";
102        gravar_dados_cartao();
103     }
104
105     valor_do_sensor_de_controle_ESP32 = analogRead(
           pino_sensor_controle_ESP32);
106     valor_do_sensor_de_controle_ESP32 =(valor_do_sensor_de_controle_ESP32
           * 3.75 ) / 843;
107     if(valor_do_sensor_de_controle_ESP32 > 2){
108        flag = true;
109     }
110     if( (valor_do_sensor_de_controle_ESP32 < 1) && (flag == true) ){
111        erro_marcador_1++;
112        Serial.println("Erro no marcador 2 (ESP32)");
113        Serial.println(String (erro_marcador_1));
114        Serial.println("  ");
115        flag = false;
116     }
117
118
```

```
119     valor_do_sensor_de_controle_ESP8266 = analogRead(
            pino_sensor_controle_ESP8266);
120     valor_do_sensor_de_controle_ESP8266 =(
            valor_do_sensor_de_controle_ESP8266 * 3.75 ) / 843;
121     if(valor_do_sensor_de_controle_ESP8266 > 2){
122        flag2 = true;
123     }
124     if( (flag2 == true) && (valor_do_sensor_de_controle_ESP8266 < 1) ) {
125        erro_marcador_2++;
126        Serial.println("Erro no marcador 1 (ESP8266)");
127        Serial.println(String (erro_marcador_2));
128        Serial.println("  ");
129        flag2 = false;
130     }
131
132 }
133
134
135 void leitura() {
136     int x = 0;
137     float valor_do_sensor_de_tensao_ESP32_soma = 0;
138     float valor_do_sensor_de_tensao_ESP8266_soma2 = 0;
139
140     while ( x < 10 ) {
141        valor_do_sensor_de_tensao_ESP32 = analogRead(
              pino_sensor_de_tensao_ESP32);
142        valor_do_sensor_de_tensao_ESP32 = (valor_do_sensor_de_tensao_ESP32
              * 3.75 ) / 843;
143        valor_do_sensor_de_tensao_ESP32_soma =
              valor_do_sensor_de_tensao_ESP32 +
              valor_do_sensor_de_tensao_ESP32_soma;
144
145        valor_do_sensor_de_tensao_ESP8266 = analogRead(
              pino_sensor_de_tensao_ESP8266);
146        valor_do_sensor_de_tensao_ESP8266 = (
              valor_do_sensor_de_tensao_ESP8266 * 3.75 ) / 843;
147        valor_do_sensor_de_tensao_ESP8266_soma2 =
              valor_do_sensor_de_tensao_ESP8266 +
              valor_do_sensor_de_tensao_ESP8266_soma2;
148
149        x++;
150
151     }
152     valor_do_sensor_de_tensao_ESP32 = valor_do_sensor_de_tensao_ESP32_soma
           / 10;
153     valor_do_sensor_de_tensao_ESP8266 =
           valor_do_sensor_de_tensao_ESP8266_soma2 / 10;
154
155 }
156
157
```

```
158 void marcador(String controle){
159    if ( (controle == "LIGADO") && (flag_controle_marcador_ON == false) &&
        (flag_ligado == false) ) {
160       flag_controle_marcador_ON = true;
161       texto_marcador = "\n \n \n \n \n \n \n \n \n Data;Hora;
          valor_do_sensor_de_tensao_ESP32;
          valor_do_sensor_de_tensao_ESP8266;erro marcador 1 (ESP8266);erro
           marcador 2 (ESP32); ||; \n";
162       flag_ligado = true;
163       Serial.println(texto_marcador);
164       delay(1000);
165    } else {
166       if ( (controle == "DESLIGADO") && (flag_controle_marcador_OFF ==
          false) && (flag_ligado == true) ) {
167          flag_controle_marcador_OFF = true;
168          texto_marcador = "\n \n \n \n \n \n \n \n \n \n  ; ; ; ; ; || \n
             ";
169          Serial.println(texto_marcador);
170          delay(1000);
171       }
172    }
173 }
174
175
176 void gravar_dados_cartao() {
177
178    datalogger = SD.open("2x6.svc", FILE_WRITE);
179       if ( datalogger ) {
180          Serial.println("Atualizando datalogger");
181          Serial.print(dados);
182          Serial.println("\n");
183
184          unsigned int tamanho_char = 100;
185          char vetor_dados [tamanho_char] = {""};
186          dados.toCharArray(vetor_dados, tamanho_char);
187
188          for ( int i = 0; i <= tamanho_char; i++){
189             if( vetor_dados[i] == '\n' ){
190                datalogger.println(" ");
191             }
192             if ( vetor_dados[i] == '*' ) {
193                break;
194             }
195             datalogger.write(vetor_dados[i]);
196          }
197          datalogger.println(" ||;");
198          datalogger.close();
199       } else {
200          Serial.println("Erro ao abrir datalogger");
201       }
202       Serial.println("Atualizado");
```

```
203         texto_marcador = "";
204         dados = "";
205 }
```

Após a leitura dos valores de tensão, foram montados gráficos afim de se analisar o comportamento prático dos primeiros protótipos.

4 Resultados

A partir da plotagem dos gráficos referentes aos valores de tensão armazenados no cartão SD, foram extraídas as conclusões iniciais. No gráfico presente na figura 10, percebe-se uma relativa aproximação com os valores teóricos, cuja a divergência foi o fator da descarga da bateria acontecer de forma exponencial, de maneira a impossibilitar os cálculos teóricos exatos.

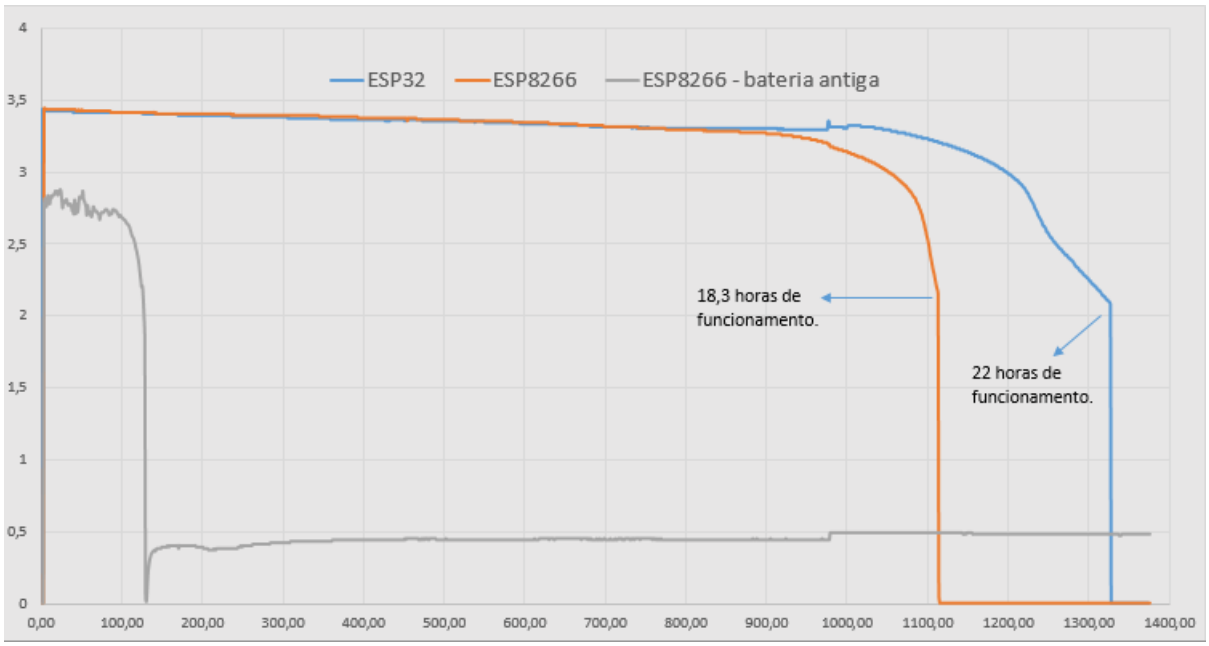


Figura 10: Gráfico 1 - Teste sem troca de funções

Com base na análise do gráfico 2 de consumo – figura 11, percebe-se a presença de várias variações bruscas de tensão, esse fato se deve ao modo de funcionamento em que as placas ESPs foram pré-programadas, pois com a troca do modo de operação, ocorre a troca de intensidade do consumo de corrente elétrica.

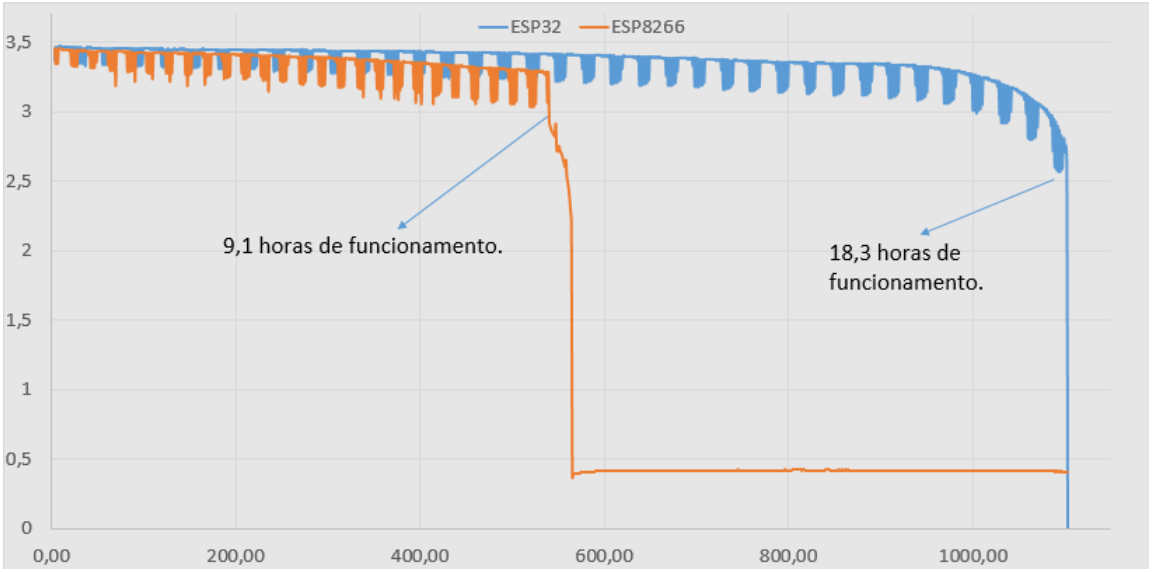


Figura 11: Gráfico 2 - Comparativo entre ESP8266 e ESP32

Isso é provado a partir da comparação entre o gráfico 1 de consumo – figura 10, no qual os ESPs não estão programados para realizar a troca de rotina de operações. Desse modo permanecem constantes sem alterar o modo de funcionamento. Por esse motivo não é perceptível nenhuma mudança abrupta no gráfico.

Devido a tal fato, a vida útil da bateria é fortemente danificada, pois não são preparadas para sofrerem esse tipo de variação de corrente elétrica, sendo percebível em todos os demais testes.

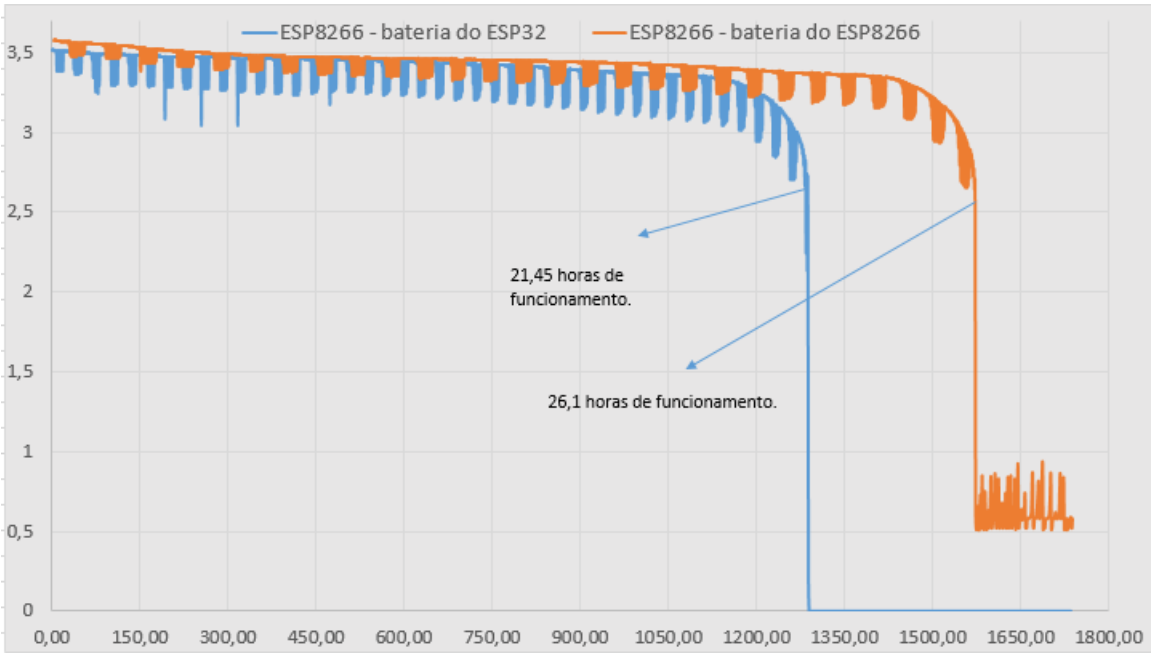


Figura 12: Gráfico 3 - Segundo comparativo, mas com as baterias trocadas

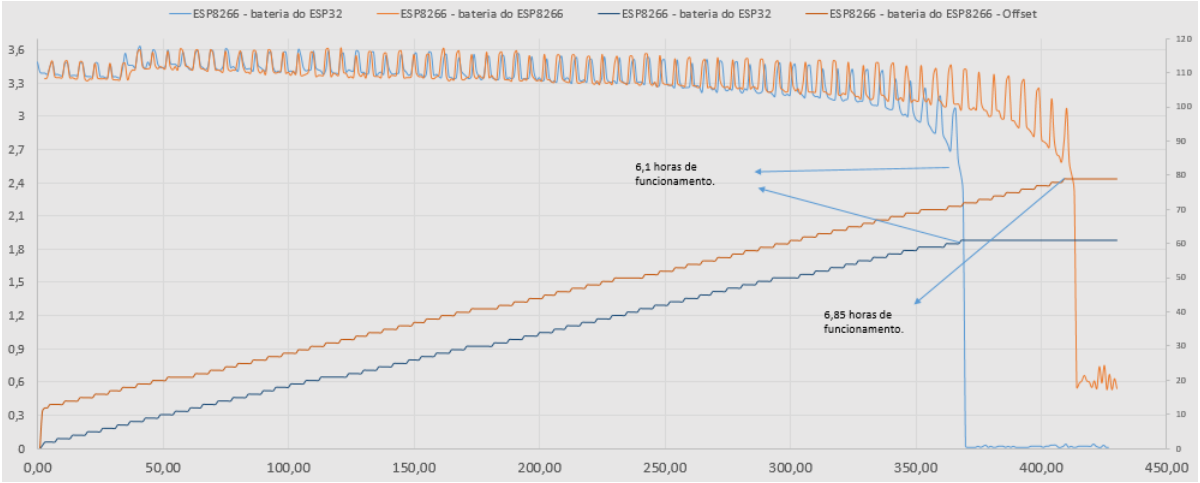


Figura 13: Gráfico 4 - Comparativo com contador de ciclos

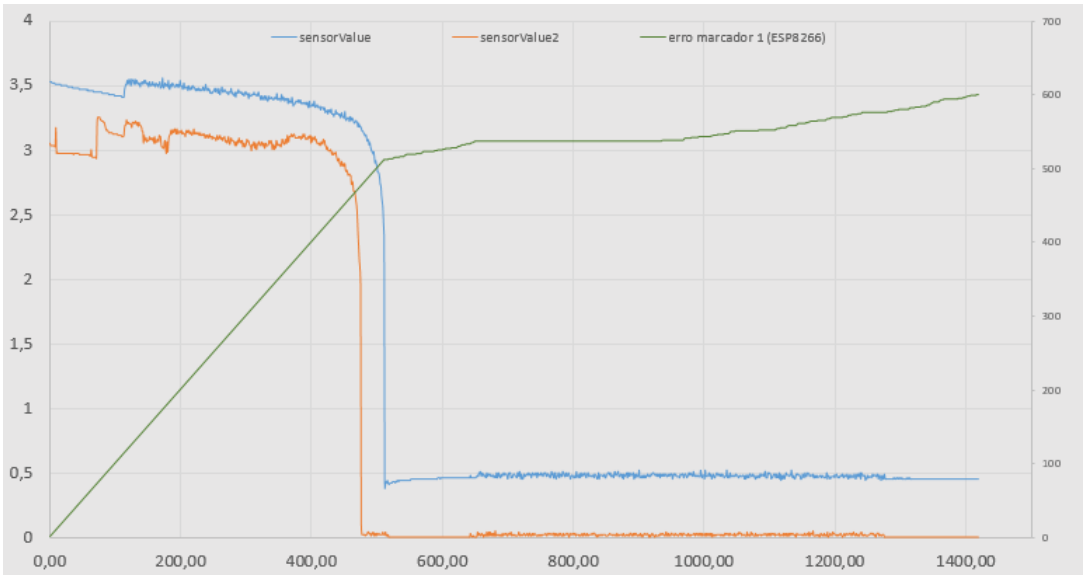


Figura 14: Gráfico 5 - Ciclo sem Deep-Sleep ativo.

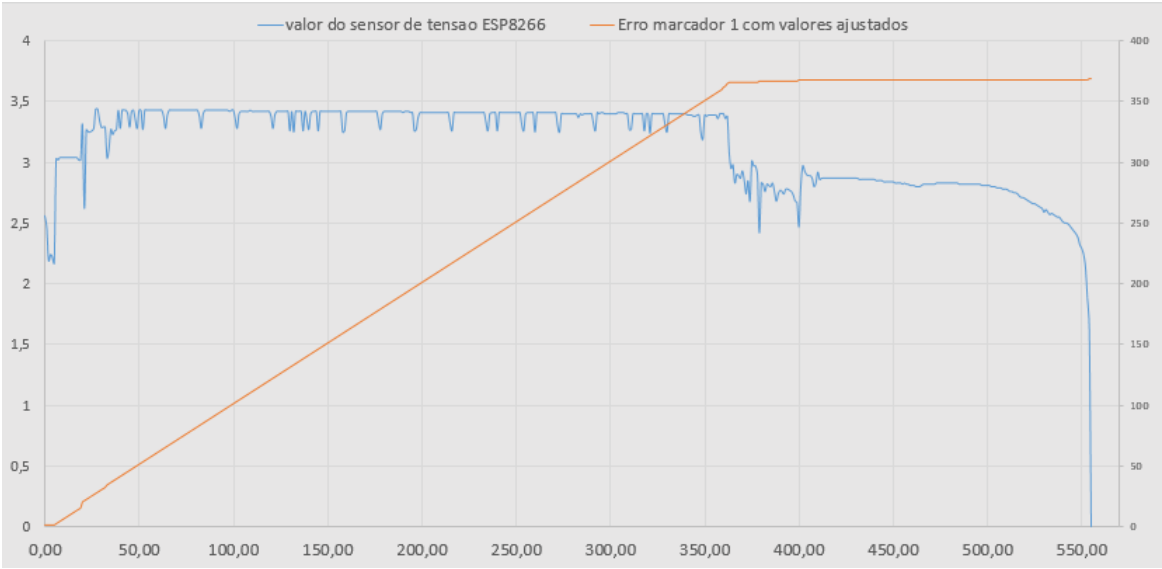


Figura 15: Gráfico 6 - Análise de limiar de tensão para funcionamento da placa ESP8266

5 Discussão

A partir dos resultados previamente encontrados, é visto que a característica autonomia e fortemente atingida pelas características de comportamento dos chips ESPs, dessa forma, faz-se necessário o estudo de soluções alternativas para eventualmente corrigir o problema e contornar a baixa vida útil da bateria.

Em primeiro momento, está sendo visado a utilização do Microcontrolador ATtiny13, como operador central dos comandos, em substituição do ESP, pelo fato de não consumir grandes valores de corrente elétrica. Como transmissor de sinais, está sendo utilizado o módulo NRF24L01, pelos mesmos motivos anunciados ao ATtiny.

Com essas novas tecnologias, é prevista uma forma de recuperar os altos valores de autonomia demonstrados nos resultados dos testes teóricos, visto que seu comportamento será mais semelhante aos teóricos do que propriamente os dos ESPs.

6 Perspectivas de continuidade do trabalho

Devido aos resultados obtidos, as pesquisas continuarão, todavia, deslocadas para novos microcontroladores, tais como o ATtiny, afim de se obter uma autonomia significativa.

Para futuras implementações, pode-se estudar a adição de um LDR – Light Dependent Resistor, cuja tradução direta se refere a um resistor dependente de luz. Este componente detecta o início da noite através da variação do seu valor de resistência, e desliga todo o chip até o amanhecer, pois espera-se que o monitoramento não seja de muita significância durante a noite. Podem-se utilizar de outros sensores ou métodos para a detecção do início da noite, mas deve-se atentar ao consumo energético, com a intenção de que este não exceda grandes valores.

Além disso, pode-se avaliar a possibilidade da utilização de um painel solar integrado ao circuito, com a finalidade de realimentar a bateria e garantir maior autonomia ao projeto. Deve-se salientar que, para que o método funcione, é preciso realizar adaptações nos chips, com a finalidade de torna-los compatíveis com recarga de energia.

7 Apêndice A - Tabela de valores de consumo completa

Os valores mostrados nessa tabela foram aferidos utilizando 3 multímetros de marcas e modelos diferentes, com a principal finalidade de se chegar a uma média precisa do valor de consumo em cada situação descrita na tabela.

Função em teste	VCC (V)	Mínimo (mA)	Máximo (mA)	Média (mA)
Standby[1]	5	74,4[2] 74,5[3] 75,2[4]	76,1 75,6 76,2	75,3333333
	4	74,8 74,3 74,8	76 77,6 76	75,5833333
	3,4	73,3 73,2 73,6	74,8 74,2 75,2	74,05
1 Led ligado[5]	5	75,5 75,3 75,8	75,2 75,3 76,4	75,5833333
	4	75,4 75,2 75,8	76,8 76,5 77,2	76,15
	3,4	73,8 73,3 73,9	74,8 74,6 75,8	74,3666667
Todos os Leds ligados	5	78,6 78,3 78,8	80 80,1 80	79,3
	4	78,6 78,2 78,8	79,5 79,3 80	79,0666667
	3,4	76,2 75,9 76,5	78,1 77,2 78,9	77,1333333
100% uso do CPU[6]	5	76 76 76,6	77,3 76,3 79,2	76,9
	4	75,5 75,3 75,9	76,6 76,6 77,5	76,2333333
	3,4	74,8 74,4 74,9	75,5 75,2 76,2	75,1666667
1 Led ligado + 100% uso CPU	5	76,8 76,4 78,2	78,1 77,7 79,5	77,7833333
	4	76,8 76,7 77,8	77,6 77,3 78,1	77,3833333
	3,4	74,9 74,4 75,4	76,3 75,2 77,1	75,55
Todos os Leds ligados + 100% uso	5	79,6 79,2 79,8	80,8 81,4 81,2	80,3333333
	4	79 79,1 79,9	80,5 80,1 80,5	79,85
	3,4	76,4 76,6 76,7	78,2 78,2 79,2	77,55
ESP em modo server e cliente ligado	5	75,2 75,1 75,5	75,5 75,1 75,8	75,3666667
	4	Instabilidade[7]		≈75
	3,4			
ESP only client	5	71,9 71,4 72,1	72,1 71,8 72,6	71,9833333
	4	Instabilidade		≈70
	3,4			
Modem Sleep	5	71,9 71,7 72	72 71,8 72,2	71,9333333
	4	70,8 71 71,4	71,4 71,1 72	71,2833333

		3,4	72,1 71,8 72,3	72,7 71,9 72,6	72,2333333
Light Sleep – CPU ativa		5	16,2 16,2 16,4	16,4 16,7 16,5	16,4
		4	16,3 16,1 16,3	16,5 16,2 16,4	16,3
		3,4	15,5 15,5 15,6	15,6 15,5 15,7	15,5666667
Light Sleep – CPU desativada		5	2,3 2,1 2,2	2,4 2,2 2,2	2,23333333
		4	2,3 2,1 2,3	2,3 2,1 2,2	2,21666667
		3,4	1,8 1,7 1,8	1,9 1,7 1,8	1,78333333
Deep Sleep		5	0,2 0,1 0,1	0,1 0,2 0,1	≈0,13333333
		4	0,02 0,01 0,01	0,02 0,02 0,02	≈0,01666667
		3,3	0,01 0,01 0,01	0,02 0,01 0,01	≈0,0116[8]
		3.0	0,01 0 0	0,01 0,01 0,01	≈0,0066
Transmit 802.11b CCK = 1Mbps Ou CCK = 11Mbps	POUT = +20.5dBm	5	74,6 74,7 74,8	75,8 75,2 76,3	75,2333333
		4	Muito instável[9]		-
		3,4			
	POUT = +18.5dBm	5	75,5 76,2 76,3	76 78,9 77,3	76,7
		4	Muito instável		-
		3,4			
	POUT = +16dBm	5	74,7 74,7 75,6	75,2 75 76,1	75,21666
		4	75,8 76,6 77,6	77,7 77 78,3	77,1666667
		3,4	Muito instável		-
	POUT = +14dBm	5	74,3 74,4 74,8	75,2 75 75,5	74,8666667
		4	75,5 75,4 76,8	77,6 76,8 78,4	76,75
		3,4	Muito instável		-
Transmit 802.11g OFDM 54Mbps	POUT = +20.5dBm	5	73,5 73,3 73,8	74,2 75 74,5	74,05
		4	Muito instável		-
		3,4			
	POUT = +18.5dBm	5	71,4 71,2 72	71,6 71,8 72,3	71,7166667
		4	71,3 71,2 71,6	71,5 71,3 72,2	71,5166667
		3,4	----	----	
	POUT = +16dBm	5	71 70,8 71,2	72 70,9 71,6	71,25
		4	70,9	71,2	71,2333333

			70,8 71,6	70,9 72	
		3,4	71,5 71,4 71,7	72 72,2 73,1	71,9833333
	POUT = +14dBm	5	70,8 70,5 71,4	71,2 71,2 71,6	71,1166667
		4	70,8 70,8 71,5	71,1 71 71,8	71,1666667
		3,4	Instabilidade		
Transmit 802.11n MCS 7	POUT = +20.5dBm	5	71,1 70,9 71,8	71,3 71,3 72,2	71,4333333
		4	71,1 70,8 71,8	71,2 71 72,1	71,3333333
		3,4	Muita instabilidade		
	POUT = +18.5dBm	5	70,1 70,5 71,6	71,3 71,1 72,5	71,1833333
		4	70,8 70,8 71,7	71,3 71,1 72	71,2833333
		3,4[10]	71,6 71,6 72,5	72 72 72,1	71,9666667
	POUT = +16dBm	5	70,9 70,9 71,5	71,1 71 71,7	71,1833333
		4	71 70,8 71,5	71,2 71 71,8	71,2166667
		3,4[*10]	71,7 71,4 72,7	72 72,1 73,3	72,2
	POUT = +14dBm	5	70,9 70,9 71,6	71,3 71,2 72	71,3166667
		4	71 70,6 71,8	71,1 71 72	71,25
		3,4[11]	Muita instabilidade		

Notas:

Existem três códigos distintos, para que seja feita uma análise mais bem elaborada. O primeiro código [1] tem como objetivo testar os modos operacionais do ESP, já o segundo[2], possui como única finalidade testar os diferentes modos de transmissão de dados e os modos de baixo consumo elétrico. O primeiro código conta com o WIFI ligado, todavia, sem a transmissão de energia.

(1) No modo *standby* o led onboard permanece ligado, sendo necessário caso queria desliga-lo, configurar via *software*.
(2) O primeiro valor de medição foi realizado com o multímetro DT830B.
(3) O segundo valor de medição foi realizado com o multímetro Imimipa ET-1002.
(4) O terceiro valor de medição foi realizado com o multímetro
(5) No modo '1 LED ligado' somente 1 do total de 8 LEDs é ligado, sendo a função seguinte ao '*standby*'
(6) O CPU executará uma série de operações aritméticas com a finalidade de por seu desempenho no máximo.
(7) Foi apresentada certa instabilidade nas medições, ou por grandes variações em curtos períodos de tempo ou pelo desligamento do ESP por problemas de alimentação.
(8) Como os valores medidos foram registrados por multímetros, estes não apresentam grande precisão para valores próximos a 0, sendo 0,01Volts o menor valor possível registrado antes do 0.
(9) Aferição das medidas impossível, por desligamento ininterrupto do ESP.
(10) Foram observados picos de tensão nas aferições que podem comprometer a autonomia. Além de observada certa instabilidade por parte do ESP.
(11) ESP apresentou muita instabilidade em certos momentos.

Tabela 4

Modo de operação	Configurações do modo	Consumo médio (mA)
Standby	VCC = 5v	75,333
1 Led ligado	VCC = 5v	75,583
Todos os Leds ligados	VCC = 5v	79,3
100% uso do CPU	VCC = 5v	76,9
1 Led ligado + 100% CPU	VCC = 5v	77,783
Todos os Leds + 100% CPU	VCC = 5v	80,333
ESP server e cliente ligado	VCC = 5v	75,366
ESP only client	VCC = 5v	71,983
Modem Sleep	VCC = 5v	71,933
Light Sleep – CPU ativa	VCC = 5v	16,4
Light Sleep – CPU desativada	VCC = 5v	2,23
Deep Sleep	VCC = 5v	0,133
	VCC = 3.3v	0,011
Transmit 802.11b	VCC = 5v e POUT = +20.5dBm	75,233
	VCC = 5v e POUT = +14dBm	74,866
Transmit 802.11g	VCC = 5v e POUT = +20.5dBm	74,050
	VCC = 5v e POUT = +14dBm	71,116
Transmit 802.11n	VCC = 5v e POUT = +20.5dBm	71,433
	VCC = 5v e POUT = +14dBm	71,250

[1] O código referido pode ser acessado por este link do github: https://github.com/W8jonas/Internet-das-Vacas/blob/master/programacao/codigo_modos_de_operacao/codigo_modos_de_operacao.ino
[2] O código referido pode ser acessado por este link do github: https://github.com/W8jonas/Internet-das-Vacas/blob/master/programacao/codigo_modos_transmissao/codigo_modos_transmissao.ino

8 Apêndice B - Código Modos de Transmissão

```
1
2  /*
3   * Instituto Federal de Educação, Ciência e Tecnologia Minas Gerais
4   * IFMG - Campus Avançado Conselheiro Lafaiete
5   *
6   *  Internet das Vacas código 4
7   *  Autor: Jonas Henrique Nascimento
8   *  PIBIC-Junior
9   *
10  *  Data de início: 11/06/2018
11  *  Data da ultima atualização: 22/07/2018
12  *  Data de término: 17/06/2018
13  *
14  *  Tem o objetivo de estabelecer modelos de transmissão de dados via
        WiFi.
15  *  Cada modelo foi alterado separadamente, sendo necessário o re-upload
        do
16  *  código ao ESP. Cada modo possui vantagens e desvantagens, sendo essas
17  *  incorporadas em sua velocidade de transmissão, potência de alcance ou
18  *  consumo de energia.
19  *  O código tem a especial necessidade de aferição do consumo energético
20  *  para a futura comparação entre eles. Sendo assim, não foi preocupado
21  *  os dados que seriam transmitidos. A distância entre os nós de toda a
22  *  rede permaneceram constantes durante todos os experimentos.
23  *
24  *  Inicialmente, foram testados o modo 802.11b, e seus respectivos
        valores
25  *  de potência de sinal.
26  *  Após, testado o 802.11g, e novamente, com seus respectivos valores de
27  *  potência, +20.5 dBm, +18.5 dBm, + 16 dBm e +14 dBm.
28  *  Por fim, foi testado o modelo 802.11n. Também com os mesmos valores
        de
29  *  potência de saída.
30  *
31  *  Este código está disponível sempre no endereço abaixo, para livre
        aperfeiçoamento.
32  *  Todavia, pede-se por educação, que ao compartilharem o código,
        mantenham os autores
33  *  originais, tão bem quanto o nome da instituição.
34  *
35  *  https://github.com/W8jonas/Internet-das-Vacas/blob/master/programacao
        /codigo_modos_transmissao/codigo_modos_transmissao.ino
36  *
37  */
38
39
40 #include <ESP8266WiFi.h>
41 #include <WiFiClient.h>
42 #include <ESP8266WebServer.h>
```

```
#include <ESP8266mDNS.h>

extern "C" {
  #include "user_interface.h"
}

const char* ssid = "esp8266";
const char* password = "1234567890";

ESP8266WebServer server(80);

void handleRoot() {

  String textoHTML;

  textoHTML = "Ola!! Aqui &eacute; o <b>ESP8266</b> falando! ";
  textoHTML += "Porta A0: ";
  textoHTML += analogRead(A0);

  server.send(200, "text/html", textoHTML);
}

void handleNotFound(){
  String message = "File Not Found\n\n";
  message += "URI: ";
  message += server.uri();
  message += "\nMethod: ";
  message += (server.method() == HTTP_GET)?"GET":"POST";
  message += "\nArguments: ";
  message += server.args();
  message += "\n";
  for (uint8_t i=0; i<server.args(); i++){
    message += " " + server.argName(i) + ": " + server.arg(i) + "\n";
  }
  server.send(404, "text/plain", message);
}

void setup(void){
  Serial.begin(115200);

  WiFi.setPhyMode(WIFI_PHY_MODE_11N);
  WiFi.setOutputPower(14);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

//wifi_set_user_fixed_rate;
//wifi_set_user_sup_rate;
//wifi_set_user_rate_limit;

   wifi_set_user_fixed_rate(1, 54);

```

```
94    Serial.println("");
95    int contador = 0;
96    while (WiFi.status() != WL_CONNECTED) {
97      contador ++;
98      delay(50);
99      Serial.print("tentativa numero: ");
100     Serial.println(contador);
101   }
102   Serial.println("");
103   Serial.print("Conectando em: ");
104   Serial.println(ssid);
105   Serial.print("IP: ");
106   Serial.println(WiFi.localIP());
107
108   server.on("/", handleRoot);
109
110   server.on("/inline", [](){
111     server.send(200, "text/plain", "this works as well");
112   });
113
114   server.onNotFound(handleNotFound);
115
116   server.begin();
117   Serial.println("Servidor iniciado");
118 }
119
120 void loop(void){
121   server.handleClient();
122 }
```

9 Apêndice C - Código Modos Wi-Fi

```
1
2  /*
3   * Instituto Federal de Educação, Ciência e Tecnologia Minas Gerais
4   * IFMG - Campus Avançado Conselheiro Lafaiete
5   *
6   *  Internet das Vacas código 3
7   *  Autor: Jonas Henrique Nascimento
8   *  PIBIC-Junior
9   *
10  *  Data de início: 05/06/2018
11  *  Data da ultima atualização: 13/08/2018
12  *  Data de término: 10/06/2018
13  *
14  *  Tem o objetivo de estabelecer modelos de operação de economia de
        energia.
15  *  Sendo, no total, 6 funções. A primeira, abilita o ESP em modo standby
        , a
16  *  qual nao realiza nenhuma função. Já a segunda, abilita o ESP a
        trabalhar
17  *  em modo client e server ao mesmo tempo. De modo semelhante, a função
        3
18  *  abilita o ESP a trabalhar somente em modo client. A partir da
        modo_light_sleep
19  *  o ESP recebe sua configuração voltada realmente a economia de energia
        .
20  *  Sendo o primeiro modelo, o light sleep com a cpu ainda ativa para ser
21  *  inicializada rapidamente. Em contrapartida a próxima função realiza
22  *  o light sleep, todavia com a dpu desativada, sendo a unica maneira de
23  *  retornar ao estado de funcionamento do ESP por uma interrupção
        externa.
24  *  Por fim, o modelo mais economico, o Deep sleep, a qual deixa o ESP
        quase
25  *  que inteiramente desligado por um período de tempo definido pelo
        programador.
26  *  Toda via, esse modelo requer uma pequena modificação na placa, a qual
         deverá
27  *  constar um jump que interlique a GPIO D0 ao pino Reset do ESP.
28  *
29  *  Este código está disponível sempre no endereço abaixo, para livre
        aperfeiçoamento.
30  *  Todavia, pede-se por educação, que ao compartilharem o código,
        mantenham os autores
31  *  originais, tão bem quanto o nome da instituição.
32  *
33  *  https://github.com/W8jonas/Internet-das-Vacas/blob/master/programacao
        /codigo_modos_wifi/codigo_modos_wifi.ino
34  *
35  */
36
```

```
#include <ESP8266WiFi.h>

extern "C" {
  #include "user_interface.h"
}

#define entrada_botao D6

void modo_standby();
void modo_server_and_client();
void modo_only_client();
void modo_modem_sleep();
void modo_light_sleep();
void modo_light_sleep_CPU_OFF();
void modo_DEEP_SLEEP();

int operacao = 0;
boolean leitura = true;

void setup() {
   pinMode(LED_BUILTIN, OUTPUT);
   pinMode (entrada_botao, INPUT);
   Serial.begin(115200);
}

void loop() {
  leitura = digitalRead(entrada_botao);
  Serial.println(leitura);
  if (leitura == LOW ){
     operacao++;
     delay(500);
  }
   switch (operacao){
      case 0:
         modo_standby();
         break;  // ESP em modo standby
      case 1:
         modo_server_and_client();
         break;  // ESP em modo server and client
      case 2:
         modo_only_client();
         break;  // Esp em modo only client
      case 3:
         modo_modem_sleep();
         break;  // Esp em modem sleep
      case 4:
         modo_light_sleep();
         break;  // Esp em modo light - sleep com cpu ativa
      case 5:
         modo_light_sleep_CPU_OFF();
         break;  // Esp em light - sleep cpu desligada OBS: Para que se
```

```
            possa chegar no deep sleep é preciso comentar a chamada da
            modo_light_sleep_CPU_OFF.
88        case 6:
89           modo_DEEP_SLEEP();
90           break;  // Esp em deep sleep
91        case 7:
92        default:
93           operacao=0;
94           break;  //
95     }
96  }
97
98  void modo_standby (){
99     Serial.println("ESP em modo standby");
100 }
101
102 void modo_server_and_client() {
103    Serial.println("ESP em modo server and client");
104    WiFi.mode(WIFI_AP);
105 }
106
107 void modo_only_client() {
108    Serial.println("Esp em modo only client");
109    WiFi.mode(WIFI_STA);
110 }
111
112 void modo_modem_sleep() {
113    Serial.println("Esp em modem sleep");
114    WiFi.mode(WIFI_OFF);
115 }
116
117 void modo_light_sleep() {
118    Serial.println("Esp em modo forçado de sleep");
119    wifi_fpm_open();
120    WiFi.forceSleepBegin();
121    delay(3000);
122    WiFi.forceSleepWake();
123    wifi_fpm_close();
124    wifi_fpm_do_wakeup();
125    digitalWrite(LED_BUILTIN, HIGH); delay(200); digitalWrite(LED_BUILTIN,
           LOW); delay(200);
126    digitalWrite(LED_BUILTIN, HIGH); delay(200); digitalWrite(LED_BUILTIN,
           LOW); delay(200);
127 }
128
129 void modo_light_sleep_CPU_OFF() {
130    Serial.println("Esp em modo Light - sleep");
131    wifi_station_disconnect();
132    wifi_set_opmode(NULL_MODE);
133    wifi_fpm_set_sleep_type(LIGHT_SLEEP_T);
134    wifi_fpm_open();
```

```
135     gpio_pin_wakeup_enable(GPIO_ID_PIN(15), GPIO_PIN_INTR_HILEVEL);
136     Serial.flush();
137     wifi_fpm_do_sleep(0xFFFFFFF);
138     delay(100);
139 }
140 
141 void modo_DEEP_SLEEP() {
142     Serial.println("Esp em deep sleep");
143     ESP.deepSleep(10000000 , WAKE_RF_DEFAULT);
144     delay(100);
145 }
```

10 Apêndice D - Código servidor para teste de consumo

```
1
2  /*
3   * Instituto Federal de Educação, Ciência e Tecnologia Minas Gerais
4   * IFMG - Campus Avançado Conselheiro Lafaiete
5   *
6   *  Internet das Vacas código 3
7   *  Autor: Jonas Henrique Nascimento
8   *  PIBIC-Junior
9   *
10  *  Data de início: 05/06/2018
11  *  Data da ultima atualização: 13/08/2018
12  *  Data de término: 10/06/2018
13  *
14  *  Tem o objetivo de estabelecer modelos de operação de economia de
        energia.
15  *  Sendo, no total, 6 funções. A primeira, abilita o ESP em modo standby
        , a
16  *  qual nao realiza nenhuma função. Já a segunda, abilita o ESP a
        trabalhar
17  *  em modo client e server ao mesmo tempo. De modo semelhante, a função
        3
18  *  abilita o ESP a trabalhar somente em modo client. A partir da
        modo_light_sleep
19  *  o ESP recebe sua configuração voltada realmente a economia de energia
        .
20  *  Sendo o primeiro modelo, o light sleep com a cpu ainda ativa para ser
21  *  inicializada rapidamente. Em contrapartida a próxima função realiza
22  *  o light sleep, todavia com a dpu desativada, sendo a unica maneira de
23  *  retornar ao estado de funcionamento do ESP por uma interrupção
        externa.
24  *  Por fim, o modelo mais economico, o Deep sleep, a qual deixa o ESP
        quase
25  *  que inteiramente desligado por um período de tempo definido pelo
        programador.
26  *  Toda via, esse modelo requer uma pequena modificação na placa, a qual
         deverá
27  *  constar um jump que interlique a GPIO D0 ao pino Reset do ESP.
28  *
29  *  Este código está disponível sempre no endereço abaixo, para livre
        aperfeiçoamento.
30  *  Todavia, pede-se por educação, que ao compartilharem o código,
        mantenham os autores
31  *  originais, tão bem quanto o nome da instituição.
32  *
33  *  https://github.com/W8jonas/Internet-das-Vacas/blob/master/programacao
        /codigo_modos_wifi/codigo_modos_wifi.ino
34  *
35  */
36
```

```
#include <ESP8266WiFi.h>

extern "C" {
  #include "user_interface.h"
}

#define entrada_botao D6

void modo_standby();
void modo_server_and_client();
void modo_only_client();
void modo_modem_sleep();
void modo_light_sleep();
void modo_light_sleep_CPU_OFF();
void modo_DEEP_SLEEP();

int operacao = 0;
boolean leitura = true;

void setup() {
   pinMode(LED_BUILTIN, OUTPUT);
   pinMode (entrada_botao, INPUT);
   Serial.begin(115200);
}

void loop() {
  leitura = digitalRead(entrada_botao);
  Serial.println(leitura);
  if (leitura == LOW ){
     operacao++;
     delay(500);
  }
   switch (operacao){
      case 0:
         modo_standby();
         break;  // ESP em modo standby
      case 1:
         modo_server_and_client();
         break;  // ESP em modo server and client
      case 2:
         modo_only_client();
         break;  // Esp em modo only client
      case 3:
         modo_modem_sleep();
         break;  // Esp em modem sleep
      case 4:
         modo_light_sleep();
         break;  // Esp em modo light - sleep com cpu ativa
      case 5:
         modo_light_sleep_CPU_OFF();
         break;  // Esp em light - sleep cpu desligada OBS: Para que se
```

```
            possa chegar no deep sleep é preciso comentar a chamada da
            modo_light_sleep_CPU_OFF.
88        case 6:
89           modo_DEEP_SLEEP();
90           break;  // Esp em deep sleep
91        case 7:
92        default:
93           operacao=0;
94           break;  //
95     }
96  }
97
98  void modo_standby (){
99     Serial.println("ESP em modo standby");
100 }
101
102 void modo_server_and_client() {
103    Serial.println("ESP em modo server and client");
104    WiFi.mode(WIFI_AP);
105 }
106
107 void modo_only_client() {
108    Serial.println("Esp em modo only client");
109    WiFi.mode(WIFI_STA);
110 }
111
112 void modo_modem_sleep() {
113    Serial.println("Esp em modem sleep");
114    WiFi.mode(WIFI_OFF);
115 }
116
117 void modo_light_sleep() {
118    Serial.println("Esp em modo forçado de sleep");
119    wifi_fpm_open();
120    WiFi.forceSleepBegin();
121    delay(3000);
122    WiFi.forceSleepWake();
123    wifi_fpm_close();
124    wifi_fpm_do_wakeup();
125    digitalWrite(LED_BUILTIN, HIGH); delay(200); digitalWrite(LED_BUILTIN,
           LOW); delay(200);
126    digitalWrite(LED_BUILTIN, HIGH); delay(200); digitalWrite(LED_BUILTIN,
           LOW); delay(200);
127 }
128
129 void modo_light_sleep_CPU_OFF() {
130    Serial.println("Esp em modo Light - sleep");
131    wifi_station_disconnect();
132    wifi_set_opmode(NULL_MODE);
133    wifi_fpm_set_sleep_type(LIGHT_SLEEP_T);
134    wifi_fpm_open();
```

```
135     gpio_pin_wakeup_enable(GPIO_ID_PIN(15), GPIO_PIN_INTR_HILEVEL);
136     Serial.flush();
137     wifi_fpm_do_sleep(0xFFFFFFF);
138     delay(100);
139 }
140 
141 void modo_DEEP_SLEEP() {
142     Serial.println("Esp em deep sleep");
143     ESP.deepSleep(10000000 , WAKE_RF_DEFAULT);
144     delay(100);
145 }
```

11 Apêndice E - Código testes de consumo

```
1
2  /*
3   * Instituto Federal de Educação, Ciência e Tecnologia Minas Gerais
4   * IFMG - Campus Avançado Conselheiro Lafaiete
5   *
6   *  Internet das Vacas código 3
7   *  Autor: Jonas Henrique Nascimento
8   *  PIBIC-Junior
9   *
10  *  Data de início: 13/08/2018
11  *  Data da ultima atualização: 15/09/2018
12  *  Data de término: 02/09/2018
13  *
14  *  Tem o objetivo de estabelecer modelos de operação de economia de
        energia.
15  *  Sendo, no total, 6 funções. A primeira, abilita o ESP em modo standby
        , a
16  *  qual nao realiza nenhuma função. Já a segunda, abilita o ESP a
        trabalhar
17  *  em modo client e server ao mesmo tempo. De modo semelhante, a função
        3
18  *  abilita o ESP a trabalhar somente em modo client. A partir da
        modo_light_sleep
19  *  o ESP recebe sua configuração voltada realmente a economia de energia
        .
20  *  Sendo o primeiro modelo, o light sleep com a cpu ainda ativa para ser
21  *  inicializada rapidamente. Em contrapartida a próxima função realiza
22  *  o light sleep, todavia com a dpu desativada, sendo a unica maneira de
23  *  retornar ao estado de funcionamento do ESP por uma interrupção
        externa.
24  *  Por fim, o modelo mais economico, o Deep sleep, a qual deixa o ESP
        quase
25  *  que inteiramente desligado por um período de tempo definido pelo
        programador.
26  *  Toda via, esse modelo requer uma pequena modificação na placa, a qual
         deverá
27  *  constar um jump que interlique a GPIO D0 ao pino Reset do ESP.
28  *
29  *  Este código está disponível sempre no endereço abaixo, para livre
        aperfeiçoamento.
30  *  Todavia, pede-se por educação, que ao compartilharem o código,
        mantenham os autores
31  *  originais, tão bem quanto o nome da instituição.
32  *
33  *  https://github.com/W8jonas/Internet-das-Vacas/blob/master/programacao
        /codigo_teste_de_consumo/codigo_teste_de_consumo.ino
34  *
35  */
36
```

```
37
38 #include <ESP8266WiFi.h>
39 #include <WiFiClient.h>
40 #include <ESP8266WebServer.h>
41 #include <ESP8266mDNS.h>
42
43 #define tempo_para_desligar 250000
44 #define tempo_modo_funcionamento 200000
45 #define tempo_modo_funcionamento2 250000
46
47 #define Pino_transmit D8
48
49 extern "C" {
50   #include "user_interface.h"
51 }
52
53 void modo_DEEP_SLEEP();
54 void handleNotFound();
55 void handleRoot();
56 void MAX_CPU();
57
58 const char* ssid = "82W8JH";
59 const char* password = "w8989jonas";
60
61 ESP8266WebServer server(80);
62
63 void setup() {
64
65    Serial.begin(115200);
66    Serial.println("Inicializando o setup");
67    WiFi.setPhyMode(WIFI_PHY_MODE_11N);
68    WiFi.setOutputPower(14);
69    WiFi.mode(WIFI_STA);
70    WiFi.begin(ssid, password);
71    wifi_set_user_fixed_rate(1, 54);
72    int contador = 0;
73
74    pinMode(Pino_transmit, OUTPUT);
75    digitalWrite(Pino_transmit, HIGH);
76
77    while (WiFi.status() != WL_CONNECTED) {
78       contador ++;
79       delay(50);
80    }
81
82    server.on("/", handleRoot);
83    server.on("/inline", [] () { server.send(200, "text/plain", "this
         works as well"); } );
84
85    server.onNotFound(handleNotFound);
86    server.begin();
```

```
87      delay(500);
88
89      Serial.println("");
90      Serial.print("Conectando em: ");
91      Serial.println(ssid);
92      Serial.print("IP: ");
93      Serial.println(WiFi.localIP());
94  }
95
96
97  void loop() {
98      unsigned long valor_atual_contador = millis();
99      Serial.println(valor_atual_contador);
100     digitalWrite(Pino_transmit, LOW);
101     if (valor_atual_contador < tempo_modo_funcionamento) {
102        MAX_CPU();
103        digitalWrite(Pino_transmit, LOW);
104     }
105
106     if ((valor_atual_contador >= tempo_modo_funcionamento) && (
          valor_atual_contador < tempo_modo_funcionamento2)) {
107        server.handleClient();
108        handleRoot();
109     }
110
111     if (valor_atual_contador >= tempo_para_desligar) {
112        modo_DEEP_SLEEP();
113     }
114
115 }
116
117
118 void MAX_CPU() {
119     Serial.println("MAX_CPU");
120     int cont = 1;
121     float resp = 1;
122     float resp2 = 1;
123     for(int AA = 0; AA < 500; AA++){
124        resp = 3 + sin(resp)/cos(resp*resp/2) * sqrt(sqrt(resp*resp));
125        resp = resp * 0.5;
126        resp2 = sqrt(AA);
127        yield();
128    }
129 }
130
131
132 void handleRoot() {
133     Serial.println("HandleRoot");
134     String textoHTML;
135     textoHTML = " PARA DE ME COPIAR DANIELLE ";
136     server.send(200, "text/html", textoHTML);
```

```
}


void handleNotFound(){
    Serial.println("handleNotFound");
    String message = "Sem arquivo\n\n";
    message += "URI: ";
    message += server.uri();
    message += "\nMethod: ";
    message += (server.method() == HTTP_GET)?"GET":"POST";
    message += "\nArguments: ";
    message += server.args();
    message += "\n";
    for (uint8_t i=0; i<server.args(); i++){
      message += " " + server.argName(i) + ": " + server.arg(i) + "\n";
    }
    server.send(404, "text/plain", message);
}


void modo_DEEP_SLEEP() {
    digitalWrite(Pino_transmit, LOW);
    digitalWrite(Pino_transmit, HIGH);
    delay(800);
    digitalWrite(Pino_transmit, LOW);
    Serial.println("Deep-Sleep");
    delay(500);
    ESP.deepSleep(100000000 , WAKE_RF_DEFAULT);
    delay(500);
}
```

12 Apêndice F - Código testes de consumo ESP32

```
1
2  /*
3   * Instituto Federal de Educação, Ciência e Tecnologia Minas Gerais
4   * IFMG - Campus Avançado Conselheiro Lafaiete
5   *
6   *  Internet das Vacas código 3
7   *  Autor: Jonas Henrique Nascimento
8   *  PIBIC-Junior
9   *
10  *  Data de início: 13/08/2018
11  *  Data da ultima atualização: 02/09/2018
12  *  Data de término: 02/09/2018
13  *
14  *  Tem o objetivo de estabelecer modelos de operação de economia de
        energia.
15  *  Sendo, no total, 6 funções. A primeira, abilita o ESP em modo standby
        , a
16  *  qual nao realiza nenhuma função. Já a segunda, abilita o ESP a
        trabalhar
17  *  em modo client e server ao mesmo tempo. De modo semelhante, a função
        3
18  *  abilita o ESP a trabalhar somente em modo client. A partir da
        modo_light_sleep
19  *  o ESP recebe sua configuração voltada realmente a economia de energia
        .
20  *  Sendo o primeiro modelo, o light sleep com a cpu ainda ativa para ser
21  *  inicializada rapidamente. Em contrapartida a próxima função realiza
22  *  o light sleep, todavia com a dpu desativada, sendo a unica maneira de
23  *  retornar ao estado de funcionamento do ESP por uma interrupção
        externa.
24  *  Por fim, o modelo mais economico, o Deep sleep, a qual deixa o ESP
        quase
25  *  que inteiramente desligado por um período de tempo definido pelo
        programador.
26  *  Toda via, esse modelo requer uma pequena modificação na placa, a qual
         deverá
27  *  constar um jump que interlique a GPIO D0 ao pino Reset do ESP.
28  *
29  *  Este código está disponível sempre no endereço abaixo, para livre
        aperfeiçoamento.
30  *  Todavia, pede-se por educação, que ao compartilharem o código,
        mantenham os autores
31  *  originais, tão bem quanto o nome da instituição.
32  *
33  *  https://github.com/W8jonas/Internet-das-Vacas/blob/master/programacao
        /codigo_teste_de_consumo_ESP32/codigo_teste_de_consumo_ESP32.ino
34  *
35  */
36
```

```
37
38 #include <WiFi.h>
39
40 #define tempo_para_desligar 25000
41 #define tempo_modo_funcionamento 20000
42 #define tempo_modo_funcionamento2 25000
43 #define Pino_transmit 4
44
45 void modo_DEEP_SLEEP();
46 void Servidor_ON(unsigned long contador_interno);
47 void MAX_CPU();
48
49 const char* ssid = "82W8JH";
50 const char* password = "w8989jonas";
51
52 WiFiServer server(80);
53
54
55
56 void setup() {
57
58    Serial.begin(115200);
59    Serial.println("Inicializando o setup");
60
61  //  WiFi.setPhyMode(WIFI_PHY_MODE_11N);
62  //  WiFi.setOutputPower(14);
63  //  WiFi.mode(WIFI_STA);
64
65    WiFi.begin(ssid, password);
66  //  wifi_set_user_fixed_rate(1, 54);
67
68    int contador = 0;
69
70    while (WiFi.status() != WL_CONNECTED) {
71       contador ++;
72       delay(50);
73    }
74
75    server.begin();
76    delay(500);
77
78    Serial.println("");
79    Serial.print("Conectando em: ");
80    Serial.println(ssid);
81    Serial.print("IP: ");
82    Serial.println(WiFi.localIP());
83    esp_sleep_enable_timer_wakeup(100000000);
84
85    pinMode(Pino_transmit, OUTPUT);
86    digitalWrite(Pino_transmit, HIGH);
87 }
```

```
88
89
90  void loop() {
91     unsigned long valor_atual_contador = millis();
92     Serial.println(valor_atual_contador);
93
94     if (valor_atual_contador < tempo_modo_funcionamento) {
95        MAX_CPU();
96     }
97
98     if ((valor_atual_contador >= tempo_modo_funcionamento) && (
          valor_atual_contador < tempo_modo_funcionamento2)) {
99        Servidor_ON(valor_atual_contador);
100    }
101
102 //    client.stop();
103
104    if (valor_atual_contador >= tempo_para_desligar) {
105       modo_DEEP_SLEEP();
106    }
107
108 }
109
110
111 void MAX_CPU() {
112    Serial.println("MAX_CPU");
113    int cont = 1;
114    float resp = 1;
115    float resp2 = 1;
116    for(int AA = 0; AA < 500; AA++){
117       resp = 3 + sin(resp)/cos(resp*resp/2) * sqrt(sqrt(resp*resp));
118       resp = resp * 0.5;
119       resp2 = sqrt(AA);
120       yield();
121    }
122 }
123
124 void Servidor_ON(unsigned long valor_cont){
125
126     WiFiClient client = server.available();
127     String currentLine = "";                // make a String to hold
          incoming data from the client
128     while (client.connected()) {            // loop while the client's
          connected
129       if (client.available()) {             // if there's bytes to read
            from the client,
130         char c = client.read();             // read a byte, then
131         Serial.write(c);                    // print it out the serial
              monitor
132         if (c == '\n') {                    // if the byte is a newline
              character
```

```
133
134             // if the current line is blank, you got two newline characters
                    in a row.
135             // that's the end of the client HTTP request, so send a
                   response:
136             if (currentLine.length() == 0) {
137               // HTTP headers always start with a response code (e.g. HTTP
                     /1.1 200 OK)
138               // and a content-type so the client knows what's coming, then
                      a blank line:
139               client.println("HTTP/1.1 200 OK");
140               client.println("Content-type:text/html");
141               client.println();
142
143               // the content of the HTTP response follows the header:
144               client.print("Contador");
145               client.print(valor_cont);
146
147               // The HTTP response ends with another blank line:
148               client.println();
149               // break out of the while loop:
150               break;
151             } else {    // if you got a newline, then clear currentLine:
152               currentLine = "";
153             }
154           } else if (c != '\r') {  // if you got anything else but a
                    carriage return character,
155             currentLine += c;      // add it to the end of the currentLine
156           }
157
158         }
159       }
160
161
162   }
163
164   void modo_DEEP_SLEEP() {
165     digitalWrite(Pino_transmit, LOW);
166     digitalWrite(Pino_transmit, HIGH);
167     delay(800);
168     digitalWrite(Pino_transmit, LOW);
169
170     Serial.println("Deep-Sleep");
171     delay(500);
172     esp_deep_sleep_start();
173     delay(500);
174   }
```

13 Apêndice G - Código servidor de consumo V2

```
1
2  /*
3   * Instituto Federal de Educação, Ciência e Tecnologia Minas Gerais
4   * IFMG - Campus Avançado Conselheiro Lafaiete
5   *
6   * Internet das Vacas código 2
7   * Autor: Jonas Henrique Nascimento
8   * PIBIC-Junior
9   *
10  * Data de início: 06/04/2018
11  * Data da ultima atualização: 13/05/2018
12  * Data de término: 13/05/2018
13  *
14  * Tem o objetivo de estabelecer um servidor dentro do ESP que
        possibilite ligar 8 LEDs independentes e optar por 2 estados de
        processamento;
15  * O primeiro, a qual não será realizado nenhuma operação adicional além
        do próprio servidor.
16  * No segundo, o processador, além de manter o próprio servidor,
        executará contas aritméticas adicinais afim de se usar 100% da CPU.
17  * O propósito do programa é a relaziação dos teste com relação ao
        consumo da placa.
18  *
19  *  Este código está disponível sempre no endereço abaixo, para livre
        aperfeiçoamento.
20  *  Todavia, pede-se por educação, que ao compartilharem o código,
        mantenham os autores
21  *  originais, tão bem quanto o nome da instituição.
22  *
23  * https://github.com/W8jonas/Internet-das-Vacas/blob/master/programacao/
        codigo_do_servidor---consumov2/codigo_do_servidor---consumov2.ino
24  *
25  */
26
27
28  #include <ESP8266WiFi.h>
29
30  const char* ssid = "esp8266";
31  const char* password = "1234567890";
32
33  WiFiServer server(80); //Shield irá receber as requisições das páginas (o
        padrão WEB é a porta 80)
34
35  String HTTP_req;
36  String URLValue;
37
38  void processaPorta(byte porta, byte posicao, WiFiClient cl);
39  void lePortaDigital(byte porta, byte posicao, WiFiClient cl);
40  void lePortaAnalogica(byte porta, byte posicao, WiFiClient cl);
```

```
41
42  void conta_1();
43
44  String getURLRequest(String *requisicao);
45  bool mainPageRequest(String *requisicao);
46
47  const byte qtdePinosDigitais = 9;
48  byte pinosDigitais[qtdePinosDigitais] = {2      , 0      , 4      , 5
        , 15     , 13     , 12      , 14      , 16     };
49  byte modoPinos[qtdePinosDigitais]     = {OUTPUT , OUTPUT , OUTPUT ,
    OUTPUT , OUTPUT , OUTPUT , OUTPUT  , OUTPUT  , OUTPUT };
50  boolean flag = true;
51  int condicao = 2;
52
53
54
55  void setup(){
56      Serial.begin(115200);
57
58      Serial.println();
59      Serial.print("Conectando a ");
60      Serial.println(ssid);
61
62      WiFi.begin(ssid, password);
63
64      while (WiFi.status() != WL_CONNECTED) {
65        delay(500);
66        Serial.print(".");
67      }
68      Serial.println("");
69      Serial.println("WiFi conectado!");
70
71
72      server.begin();
73      Serial.println("Server iniciado");
74
75      Serial.println(WiFi.localIP());
76
77      for (int nP=0; nP < qtdePinosDigitais; nP++) {
78          pinMode(pinosDigitais[nP], modoPinos[nP]);
79      }
80  }
81
82
83  void loop(){
84
85      WiFiClient  client = server.available();
86
87      if (client) {
88          boolean currentLineIsBlank = true;
89          while (client.connected()) {
```

```
90              if (client.available()) {
91                  char c = client.read();
92                  HTTP_req += c;
93
94                  if (c == '\n' && currentLineIsBlank ) {
95
96                      if ( mainPageRequest(&HTTP_req) ) {
97                          URLValue = getURLRequest(&HTTP_req);
98                          Serial.println(HTTP_req);
99
100                         client.println("HTTP/1.1 200 OK");
101                         client.println("Content-Type: text/html");
102                         client.println("Connection: keep-alive");
103                         client.println();
104
105                         //Conteudo da Página HTML
106                         client.println("<!DOCTYPE html>");
107                         client.println("<html>");
108
109                         client.println("<head>");
110                         client.println("<title>Servidor de teste</title>"
                                );
111
112                         client.println("<script>");
113                         client.println("function LeDadosDoArduino() {");
114                         client.println("nocache = \"&nocache=\" + Math.
                                random() * 1000000;");
115                         client.println("var request = new XMLHttpRequest
                                ();");
116                         client.println("var posIni;");
117                         client.println("var valPosIni;");
118                         client.println("var valPosFim;");
119                         client.println("request.onreadystatechange =
                                function() {");
120                         client.println("if (this.readyState == 4) {");
121                         client.println("if (this.status == 200) {");
122                         client.println("if (this.responseText != null) {"
                                );
123
124                         for (int nL=0; nL < qtdePinosDigitais; nL++) {
125                             client.print("posIni = this.responseText.
                                    indexOf(\"PD");
126                             client.print(pinosDigitais[nL]);
127                             client.println("\");");
128                             client.println("if ( posIni > -1) {");
129                             client.println("valPosIni = this.responseText
                                    .indexOf(\"#\", posIni) + 1;");
130                             client.println("valPosFim = this.responseText
                                    .indexOf(\"|\", posIni);");
131                             client.print("document.getElementById(\"pino"
                                    );
```

```
132                             client.print(pinosDigitais[nL]);
133                             client.println("\").checked = Number(this.
                                    responseText.substring(valPosIni,
                                    valPosFim));");
134                             client.println("}");
135                         }
136
137
138                         client.println("}}}}");
139                         client.println("request.open(\"GET\", \"
                                solicitacao_via_ajax\" + nocache, true);");
140                         client.println("request.send(null);");
141                         client.println("setTimeout('LeDadosDoArduino()',
                                5000);");
142                         client.println("}");
143                         client.println("</script>");
144
145                         client.println("</head>");
146
147                         client.println("<body onload=\"LeDadosDoArduino()
                                \">");
148                         client.println("<h1>Porta analogica</h1>");
149
150                         client.println("<br/>");
151                         client.println("<h1>PORTAS digitais</h1>");
152                         client.println("<form method=\"get\">");
153
154                         for (int nL=0; nL < qtdePinosDigitais; nL++) {
155                             processaPorta(pinosDigitais[nL], nL, client);
156                             client.println("<br/>");
157                         }
158
159                         client.println("<button type=\"submit\">Envia as
                                opcoes para o ESP8266</button>");
160                         client.println("</form>");
161
162                         client.println("</body>");
163
164                         client.println("</html>");
165
166                     } else if (HTTP_req.indexOf("solicitacao_via_ajax") >
                            -1) {
167
168                         Serial.println(HTTP_req);
169
170                         client.println("HTTP/1.1 200 OK");
171                         client.println("Content-Type: text/html");
172                         client.println("Connection: keep-alive");
173                         client.println();
174
175
```

```
                        for (int nL=0; nL < qtdePinosDigitais; nL++) {
                            lePortaDigital(pinosDigitais[nL], nL, client)
                                ;
                        }

                    } else {

                        Serial.println(HTTP_req);
                        client.println("HTTP/1.1 200 OK");
                    }
                    HTTP_req = "";
                    break;
                }

                if (c == '\n') {
                    currentLineIsBlank = true;
                }
                else if (c != '\r') {
                    currentLineIsBlank = false;
                }
            }
        }
        delay(1);
        client.stop();
    }

  if(condicao == 1)conta_1();

}


void processaPorta(byte porta, byte posicao, WiFiClient cl){
static boolean LED_status = 0;
String cHTML;

    cHTML = "P";
    cHTML += porta;
    cHTML += "=";
    cHTML += porta;

    if (modoPinos[posicao] == OUTPUT) {

        if (URLValue.indexOf(cHTML) > -1) {
           LED_status = HIGH;
        } else {
           LED_status = LOW;
        }
        if((porta == 16) && (LED_status == HIGH)){
          condicao = 1;
        } else{
          condicao = 0;
```

```
226             }
227             digitalWrite(porta, LED_status);
228         } else {
229 
230             LED_status = digitalRead(porta);
231         }
232 
233         cl.print("<input type=\"checkbox\" name=\"P");
234         cl.print(porta);
235         cl.print("\" value=\"");
236         cl.print(porta);
237 
238         cl.print("\"");
239 
240         cl.print(" id=\"pino");
241         cl.print(porta);
242         cl.print("\"");
243 
244         if (LED_status) {
245             cl.print(" checked ");
246         }
247 
248         if (modoPinos[posicao] != OUTPUT) {
249             cl.print(" disabled ");
250         }
251 
252         cl.print(">Porta ");
253         cl.print(porta);
254 
255         cl.println();
256 }
257 
258 
259 void lePortaDigital(byte porta, byte posicao, WiFiClient cl){
260         if (modoPinos[posicao] != OUTPUT) {
261            cl.print("PD");
262            cl.print(porta);
263            cl.print("#");
264 
265            if (digitalRead(porta)) {
266               cl.print("1");
267            } else {
268               cl.print("0");
269            }
270            cl.println("|");
271         }
272 }
273 
274 
275 void lePortaAnalogica(byte porta, byte posicao, WiFiClient cl){
276    cl.print("PA");
```

```
277     cl.print(porta);
278     cl.print("#");
279 
280     cl.print(analogRead(porta));
281 
282     //especifico para formatar o valor da porta analogica A0
283     if (porta == A0) {
284        cl.print(" (");
285        cl.print(map(analogRead(A0),0,1023,0,179));
286        cl.print("&deg;)");
287     }
288 
289     cl.println("|");
290 }
291 
292 
293 String getURLRequest(String *requisicao) {
294 int inicio, fim;
295 String retorno;
296 
297   inicio = requisicao->indexOf("GET") + 3;
298   fim = requisicao->indexOf("HTTP/") - 1;
299   retorno = requisicao->substring(inicio, fim);
300   retorno.trim();
301 
302   return retorno;
303 }
304 
305 
306 bool mainPageRequest(String *requisicao) {
307 
308 
309 String valor;
310 bool retorno = false;
311 
312   valor = getURLRequest(requisicao);
313   valor.toLowerCase();
314 
315   if (valor == "/") {
316      retorno = true;
317   }
318 
319   if (valor.substring(0,2) == "/?") {
320      retorno = true;
321   }
322 
323   if (valor.substring(0,10) == "/index.htm") {
324      retorno = true;
325   }
326 
327   return retorno;
```

```
}


void conta_1(){
  int cont = 1;
  float resp = 1;
  float resp2 = 1;
  for(int AA = 0; AA < 500; AA++){
     resp = 3 + sin(resp)/cos(resp*resp/2) * sqrt(sqrt(resp*resp));
     resp = resp * 0.5;
     Serial.println(resp);
     Serial.print("   ");
     resp2 = sqrt(AA);
     Serial.print(resp2);
     Serial.print("   ");
     yield();
  }
}
```

REFERÊNCIAS

[1] STA-ELETRONICA. Pilhas e baterias Rontek. Disponível em: *http://www.sta-eletronica.com.br/produtos/pilhas-e-baterias/rontek-recarregaveis-industrial/nicd/tamanho-aaa_2*. Acesso em 21 de abril de 2018.

[2] GOLDPOWER. Pilhas e baterias Ni-mh. Disponível em: *http://www.goldpower.com.br/aaa-800mah-1-2v.php*. Acesso em 21 de abril de 2018.

[3] GOLDPOWER. Pilhas e baterias Ni-mh. Disponível em: *http://www.goldpower.com.br/aaa-1000mah-1-2v.php*. Acesso em 21 de abril de 2018.

[4] STA-ELETRONICA. Pilhas e baterias Rontek. Disponível em: *http://www.sta-eletronica.com.br/produtos/pilhas-e-baterias/rontek-recarregaveis-consumidor/aa/12v-2100mah_3*. Acesso em 21 de abril de 2018.

[5] MOXDOTCELL. Pilha recarregável MO-AA2700. Disponível em: *http://www.moxdotcell.com.br/pilha-recarregavel-mo-aa2700-com-2-unidades-rtu.html*. Acesso em 21 de abril de 2018.

[6] COMP DISTRIBUIDORA. Bateria recarregável Knup. Disponível em: *https://www.compdistribuidora.com.br/bateria-recarregavel-9v-knup-kp-bt9v.html*. Acesso em 21 de abril de 2018.

[7] FLEXGOLD. Flex X-cell. Disponível em: *http://www.flexgold.com.br/produto/fx-9v45b1/*. Acesso em 22 de abril de 2018

[8] FULLYMAX. Bateria Fullymax SYMA. Disponível em: *http://www.asaseletricas.com.br/loja/product_info.php?products_id=4448*. Acesso em 15 de maio de 2018.

[9] STA-ELETRONICA. Pilhas e baterias Rontek. Disponível em: *http://www.sta-eletronica.com.br/produtos/pilhas-e-baterias/recarregaveis/para-cameras/37v-680mah*. Acesso em 23 de abril de 2018.

[10] STA-ELETRONICA. Pilhas e baterias Rontek. Disponível em: *http://www.sta-eletronica.com.br/produtos/pilhas-e-baterias/recarregaveis/para-brinquedos-e-modelismo/72v-1800mah*. Acesso em 23 de abril de 2018.

[11] STA-ELETRONICA. Pilhas e baterias Rontek. Disponível em: *http://www.sta-eletronica.com.br/produtos/pilhas-e-baterias/recarregaveis/para-brinquedos-e-modelismo/72v-3000mah*. Acesso em 23 de abril de 2018.

[12] STA-ELETRONICA. Pilhas e baterias Rontek. Disponível em: *http://www.sta-eletronica.com.br/produtos/pilhas-e-baterias/recarregaveis/para-telefones-sem-fio-24v-36-48-e-6v/36v-1300mah*. Acesso em 23 de abril de 2018.

[13] MOXDOTCELL. Pilha recarregável MO-AA2700. Disponível em: *http://www.moxdotcell.com.br/bateria-mo-086b-3aaa-3-6v-700-mah-para-talk-about.html*. Acesso em 21 de abril de 2018.

[14] STA-ELETRONICA. Pilhas e baterias Rontek. Disponível em: *http://www.sta-eletronica.com.br/produtos/pilhas-e-baterias/recarregaveis/pilhas-botao/36v-80mah*. Acesso em 1 de maio de 2018.

[15] STA-ELETRONICA. Pilhas e baterias Rontek. Disponível em: *http://www.sta-eletronica.com.br/produtos/pilhas-e-baterias/recarregaveis/para-telefones-sem-fio-24v-36-48-e-6v/36v-600mah*. Acesso em 1 de maio de 2018.

[16] STA-ELETRONICA. Pilhas e baterias Rontek. Disponível em: *http://www.sta-eletronica.com.br/produtos/pilhas-e-baterias/recarregaveis/para-radios-de-comunicacao/72v-600mah*. Acesso em 1 de maio de 2018.

[17] WIKI WEMOS. Esquemático completo. Disponível em: *https://wiki.wemos.cc/_media/products:d1:sch_d1_mini_v3.0.0.pdf*. Acesso em 15 de maio de 2018.

[18] DATASHEET ME6211. High Speed LDO Regulators, Low ESR Cap. Disponível em: *https://datasheet.lcsc.com/szlcsc/ME6211C33M5G-N_C82942.pdf*. Acesso em 15 de maio de 2018.

[19] FULLYMAX. Bateria Fullymax SYMA. Disponível em: *http://www.asaseletricas.com.br/loja/product_info.php?products_id=4301*. Acesso em 15 de maio de 2018.

[20] DATASHEET ESP8266EX. Disponível em: *https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf* . Acesso em 21 de maio de 2018.

[21] MINAMOTO. LiFePO4 Polymer MODELS. Disponível em: *http://www.minamoto.com/lifepo4-polymer/* Acesso em 05 de Junho de 2018.

[22] MINAMOTO. LiFePO4 Cylindrical MODELS. Disponível em: *http://www.minamoto.com/lifepo4-cylindrical/* Acesso em 05 de Junho de 2018.

[23] FULLYMAX. Bateria Fullymax SYMA. Disponível em: *http://www.asaseletricas.com.br/loja/product_info.php?products_id=4449.* Acesso em 11 de julho de 2018.