



Universidade Federal de Uberlândia
Faculdade de Engenharia Elétrica

RODRIGO CASSIANO DA SILVA MELO

**SISTEMA DE MONITORAMENTO DE CONSUMO DE ÁGUA
UTILIZANDO O PROTOCOLO DE COMUNICAÇÃO MQTT**

Uberlândia
2018

RODRIGO CASSIANO DA SILVA MELO

**SISTEMA DE MONITORAMENTO DE CONSUMO DE ÁGUA
UTILIZANDO O PROTOCOLO DE COMUNICAÇÃO MQTT**

Trabalho apresentado como requisito parcial de avaliação na disciplina Trabalho de Conclusão de Curso 2 do Curso de Engenharia Elétrica com ênfase em Controle e Automação da Universidade Federal de Uberlândia.

Orientador: Márcio José da Cunha

Assinatura do Orientador

Uberlândia
2018

Dedico este trabalho a todos aqueles que de
alguma forma possibilitaram concretizá-lo.

AGRADECIMENTOS

Primeiramente a Deus, pela saúde, resignação e inspiração.

Aos meus pais e irmãos pelo apoio incondicional.

Ao Prof. Márcio José da Cunha pela paciência, incentivo e orientação deste trabalho.

RESUMO

Este trabalho de conclusão de curso apresenta um projeto acerca da IoT (*Internet of Things* – Internet das Coisas). Foi desenvolvido um pequeno sistema de monitoramento do consumo de água com o auxílio do protocolo de comunicação MQTT (*Message Queue Telemetry Transport*), da plataforma de prototipagem NodeMCU Dev Kit v1.0 e de um ambiente web voltado para aplicações de Internet das Coisas. Todo o processo de desenvolvimento do protótipo foi exposto, desde a fase de concepção de hardware e software, até a posterior implementação do sistema. O trabalho conta ainda com uma breve introdução do cenário IoT e apresenta, de forma sucinta, as principais características do protocolo MQTT.

Palavras-chave: Monitoramento, Consumo de água, IoT, ESP8266EX, MQTT.

ABSTRACT

This paper presents a project about IoT (Internet of Things). Was developed a little water consumption monitoring system with the aid of the MQTT (Message Queue Telemetry Transport) communication protocol, the NodeMCU v1.0 development board and a web environment dedicated to IoT applications. The entire prototype development process was exposed, since of the hardware and software conception phase until the system implementation. The paper still count on a short introduction of Internet of Things scenario and presents, in a brief way, the main features of MQTT protocol.

Keywords: Monitoring, Water Consumption, IoT, ESP8266EX, MQTT.

LISTA DE ILUSTRAÇÕES

<i>FIGURA 1 – Percentuais de água salgada e doce.....</i>	<i>14</i>
<i>FIGURA 2 – Pré-requisitos que definem uma aplicação IoT.....</i>	<i>16</i>
<i>FIGURA 3 – Camadas de uma possível arquitetura para IoT.....</i>	<i>17</i>
<i>FIGURA 4 – Processos em que os dados estão envolvidos.....</i>	<i>18</i>
<i>FIGURA 5 – Modelo publish/subscribe do protocolo MQTT.....</i>	<i>21</i>
<i>FIGURA 6 – Pilha TCP/IP, sobre a qual roda o protocolo MQTT.....</i>	<i>22</i>
<i>FIGURA 7 – Formato do cabeçalho fixo de um pacote de controle MQTT.....</i>	<i>22</i>
<i>QUADRO I – Comparativo entre a NodeMCU v1.0 e a Arduino UNO R3.....</i>	<i>27</i>
<i>FIGURA 8 – Especificações técnicas do SoC ESP8266EX.....</i>	<i>28</i>
<i>FIGURA 9 – Definição de pinos do ESP8266EX.....</i>	<i>29</i>
<i>FIGURA 10 – Definição de pinos do ESP-12E.....</i>	<i>29</i>
<i>FIGURA 11 – Sensor de fluxo de água YF-S201B.....</i>	<i>30</i>
<i>FIGURA 12 – Princípios de funcionamento de um sensor de efeito Hall.....</i>	<i>31</i>
<i>FIGURA 13 – Elementos do sensor de fluxo de água YF-S201B.....</i>	<i>32</i>
<i>FIGURA 14 – Divisor de tensão.....</i>	<i>33</i>
<i>FIGURA 15 – Montagem do protótipo gerada no Fritzing.....</i>	<i>37</i>
<i>FIGURA 16 – Esquemático das conexões do protótipo.....</i>	<i>37</i>

<i>FIGURA 17 – Pino VDD5V da NodeMCU.....</i>	<i>38</i>
<i>FIGURA 18 – Integração PlatformIO+Atom.....</i>	<i>39</i>
<i>FIGURA 19 – Fluxograma do processo executado pelo código.....</i>	<i>40</i>
<i>FIGURA 20 – Continuação do fluxograma anterior.....</i>	<i>41</i>
<i>FIGURA 21 – Processo de assinatura a um determinado tópico.....</i>	<i>43</i>
<i>FIGURA 22 – Processo de publicação em um dado tópico.....</i>	<i>44</i>
<i>FIGURA 23 – Plataforma de interface gráfica ThingSpeak.....</i>	<i>46</i>
<i>FIGURA 24 – Captura de tela do aplicativo ThingSpeak.....</i>	<i>46</i>
<i>FIGURA 25 – Captura da função callback após um reset do sistema.....</i>	<i>49</i>
<i>FIGURA 26 – Dashboard do sistema implementado.....</i>	<i>50</i>
<i>FIGURA 27 – Modos de economia de energia do SoC ESP8266.....</i>	<i>51</i>

LISTA DE TABELAS

<i>TABELA I – Distribuição de água pelo planeta.....</i>	<i>13</i>
<i>TABELA II – Tipos de mensagem de um pacote de controle MQTT.....</i>	<i>22</i>
<i>TABELA III – Níveis de garantia da entrega de um pacote de controle MQTT....</i>	<i>23</i>
<i>TABELA IV – Quantidade de bytes e os respectivos intervalos de valores.....</i>	<i>24</i>
<i>TABELA V – Componentes utilizados na montagem do protótipo.....</i>	<i>36</i>

LISTA DE ABREVIATURAS E SIGLAS

ACK	<i>ACKnowledgment</i> – Reconhecimento
ADC	<i>Analog-to-Digital Converter</i> – Conversor Analógico-Digital
CPU	<i>Central Process Unit</i> – Unidade de Processamento Central
DC	<i>Direct Current</i> – Corrente Contínua
DUP	<i>DUPlication</i> – Duplicação
EEPROM	<i>Electrically-Erasable Programmable Ready Only Memory</i>
GPIO	<i>General Purpose Input/Output</i> – Entradas/Saídas de Propósito Geral
I2C	<i>Inter-Integrated Circuit</i>
I2S	<i>Inter-Integrated Circuit Sound</i>
IBM	<i>International Business Machines</i>
IC	<i>Integrated Circuit</i> – Circuito Integrado
IDE	<i>Integrated Development Environment</i> – Ambiente de Desenvolvimento Integrado
IoT	<i>Internet of Things</i> – Internet das Coisas
IP	<i>Internet Protocol</i> – Protocolo da Internet
ISR	<i>Interrupt Service Routine</i> – Rotina de Interrupção de Serviço
LSB	<i>Least Significant Byte</i> – Byte Menos Significativo
M2M	<i>Machine-to-Machine</i>

MCU	<i>Microcontroller Unit</i> – Microcontroladores
MQTT	<i>Message Queue Telemetry Transport</i>
MSB	<i>Most Significant Byte</i> – Byte Mais Significativo
OTA	<i>Over The Air</i>
PING	<i>Packet INternet Grouper</i> – Procurador de Pacotes da Internet
PWM	<i>Pulse Width Modulation</i> – Modulação por Largura de Pulso
QoS	<i>Quality of Service</i> – Qualidade de Serviço
RAM	<i>Random Access Memory</i> – Memória de Acesso Aleatório
ROM	<i>Read-Only Memory</i> – Memória Somente de Leitura
RTC	<i>Real Time Clock</i>
SoC	<i>System-on-Chip</i> – Sistema em um Chip
SPI	<i>Serial Peripheral Interface</i> – Interface Periférica Serial
SPIFFS	<i>SPI Flash File System</i>
TCP	<i>Transmission Control Protocol</i> – Protocolo de Controle de Transmissão
TTL	<i>Transistor-Transistor Logic</i> – Lógica Transistor-Transistor
UART	<i>Universal Asynchronous Receiver/Transmitter</i> – Transmissor/Receptor Assíncrono Universal
UNESCO	<i>United Nations Educational, Scientific and Cultural Organization</i> – Organização das Nações Unidas para a Educação, a Ciência e a Cultura
USB	<i>Universal Serial Bus</i> – Barramento Serial Universal
UTF-8	<i>8-bit Unicode Transformation Format</i>
VDD	Tensão de Alimentação Positiva

SUMÁRIO

1 INTRODUÇÃO	13
2 DESENVOLVIMENTO	15
2.1 REVISÃO BIBLIOGRÁFICA	15
2.1.1 Internet das Coisas	15
2.1.2 Protocolo MQTT	19
2.2 METODOLOGIA	25
2.2.1 A plataforma de prototipagem	25
2.2.2 O sensor de fluxo	30
2.2.3 Esquema elétrico.....	32
2.2.4 O ambiente de programação.....	38
2.2.5 Desenvolvimento do software.....	39
2.2.6 Interface gráfica.....	45
2.3 RESULTADOS E DISCUSSÃO.....	47
3 CONCLUSÕES	52
4 REFERÊNCIAS.....	54
APÊNDICE A – ALGORITMO COMPLETO	57
APÊNDICE B – CÓDIGO UTILIZADO NO MATLAB.....	65
ANEXO I – DADOS RELEVANTES DO DATASHEET DO ESP8266EX	66
ANEXO II – DADOS RELEVANTES DO DATASHEET DO MÓDULO ESP-12E.....	70
ANEXO III – ESQUEMÁTICO DA PLACA NODEMCU DEV KIT V1.0	72
ANEXO IV – DATASHEET DO SENSOR DE FLUXO YF-S201B 1/2"	78

1 INTRODUÇÃO

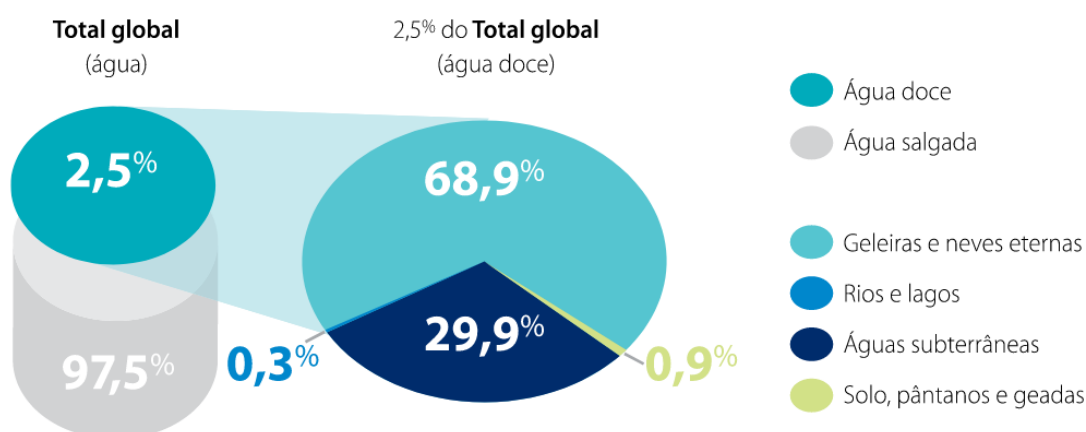
Um dos temas que vem ganhando cada vez mais relevância é o da eficiência energética, e não por acaso, afinal, vivemos em um mundo de demandas ilimitadas mas com recursos limitados (VASCONCELLOS; GARCIA, 2014). Um desses recursos, sendo provavelmente o mais importante, é a água, pois é vital à manutenção da vida. Apesar da maior parte da superfície do planeta ser coberta por água, a quantidade que é própria para o consumo é relativamente pequena, e menor ainda é a parcela deste recurso que se encontra em estado e/ou local acessíveis, como mostram a tabela I (retirada de um documento da UNESCO sobre as águas) e a figura 1.

Tabela I – Distribuição de água pelo planeta

Localização	Percentual do volume total na hidrosfera (%)	Percentual de água potável (%)
Oceanos	96,5	-
Águas subterrâneas (gravidade e capilaridade)	1,7	
Águas subterrâneas predominantemente potáveis	0,76	30,1
Umidade do solo	0,001	0,05
Geleiras e neves permanentes	1,74	68,7
Lagos:	0,013	-
Água doce	0,007	0,26
Água salgada	0,006	-
Brejos e pântanos	0,0008	0,03
Rios	0,0002	0,006
Água biológica	0,0001	0,003
Água na atmosfera	0,001	0,04
Volume total de água na hidrosfera	100	-
Total de água potável	2,53	100

Fonte: Adaptado de UNESCO, 2003.

Figura 1 – Percentuais de água doce e salgada



Fonte: GEOBRMUNDO, 2016.

Sendo assim, se a intenção da espécie humana for permanecer na Terra por mais algum tempo, é necessário que a utilização deste recurso seja a mais racional possível. E por que não fazer uso das tecnologias sempre presentes à nossa volta nos dias de hoje, afim de possibilitar não só o monitoramento dos mais variados recursos como o processamento dos dados visando à otimização do uso? Talvez o fato de saberem o quanto de água estão utilizando a qualquer hora e lugar sensibilize as pessoas sobre a importância do consumo consciente.

É em meio a estas dúvidas e afirmações que surge a proposta do presente trabalho. A ideia é fazer uso de tecnologias de baixo custo e fácil acesso (e que sejam de livre utilização) para monitorar a utilização da água, de forma a garantir que as pessoas possam usufruir deste recurso por mais tempo.

Esse trabalho tem o objetivo de implementar um protótipo que seja capaz de enviar dados coletados por meio de um dispositivo (com características modestas de hardware e custo acessível) e enviá-los a uma central que consiga apresentá-los (e até processá-los) através de uma interface gráfica, e com isso fornecer as bases para futuras aplicações que pretendam fazer uso do SoC ESP8288EX e do protocolo de comunicação MQTT.

2 DESENVOLVIMENTO

Este capítulo foi organizado de forma que o entendimento seja claro a respeito das tecnologias que estão contidas na implementação do protótipo. Serão abordados assuntos relacionados à Internet das Coisas (definições); ao protocolo de comunicação MQTT (principais características e funcionamento); aos principais componentes presentes no protótipo (SoC ESP8266EX da empresa *Espressif*, placa de desenvolvimento NodeMCU v1.0 e sensor de fluxo de água YF-S201B); ao ambiente de desenvolvimento de software (integração PlatformIO + Atom); e à interface gráfica voltada para aplicações IoT, ThingSpeak.

2.1 REVISÃO BIBLIOGRÁFICA

Nesta sessão serão apresentados os principais conceitos e definições a respeito da IoT, bem como algumas soluções que já foram implementadas, tanto no Brasil quanto no exterior; além de abordar de forma mais técnica as características principais do protocolo MQTT.

2.1.1 Internet das Coisas

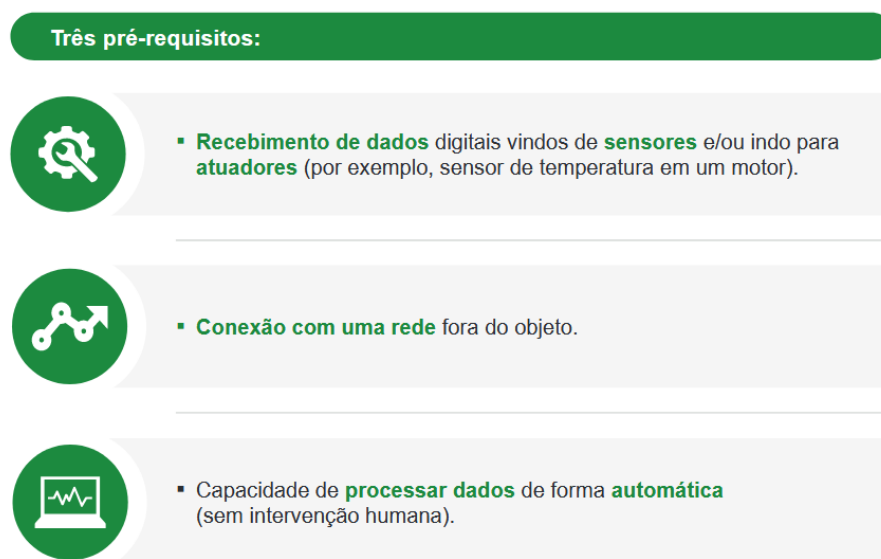
A Internet das Coisas vai muito além das tecnologias e conceitos presentes em seu escopo, pois ela implica em uma profunda mudança na maneira como pessoas e objetos se relacionam entre si. Imagine a seguinte situação, chegou o final de semana, você se dirige ao centro da cidade para fazer compras, mas após vários minutos procurando, não encontra nenhuma vaga pública disponível. E se as vagas de estacionamento possuísem sensores conectados à internet e que, além disso, fossem capazes de informar ao usuário pelo smartphone qual a vaga disponível mais próxima?

É disto que se trata a Internet das Coisas. Objetos que estejam não apenas conectados à rede mundial de computadores, mas que sejam capazes de processar dados e tomar decisões de forma inteligente sem que haja intervenção humana.

A Internet das Coisas (IoT) é uma infraestrutura global que habilita serviços avançados por meio da interconexão entre coisas (físicas e virtuais), com base nas tecnologias de informação e comunicação (TIC). Em sentido amplo, trata-se não apenas de conectar coisas, como veículos e eletrodomésticos, mas também de dotá-las do poder de processar dados, tornando-as “inteligentes”. Por isso, a IoT vem ganhando espaço não somente pelo surgimento de tecnologias disruptivas, mas também pela evolução de um conjunto de tecnologias já disponíveis, que estão se tornando mais acessíveis, possibilitando sua adoção em massa (BNDES *et al.*, 2018, p. 7).

De acordo com o BNDES *et al.* (2018), para que uma dada aplicação se enquadre no que tange à Internet das Coisas, três requisitos são considerados básicos, como mostra a figura 2.

Figura 2 – Pré-requisitos que definem uma aplicação IoT



Fonte: BNDES, 2018.

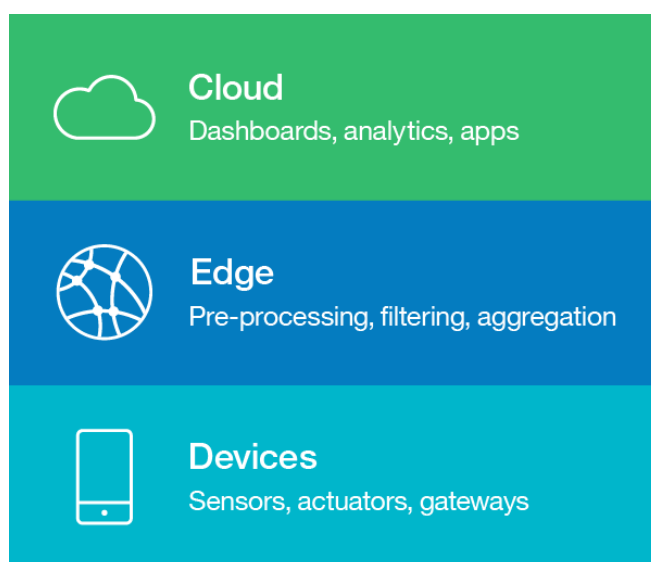
Outro aspecto muito importante no que se refere a aplicações de IoT, é a sua arquitetura, ou seja, a forma como estão dispostos os elementos de uma estrutura e a maneira pela qual estes se relacionam entre si. Em se tratando de sistemas complexos, como é o caso das soluções de Internet das Coisas, a abordagem por camadas se mostra muito eficaz, uma vez que permite uma melhor compreensão

das partes e da forma como elas estão interligadas, além de ser extremamente eficiente, pois possibilita que a resolução de problemas aconteça em parcelas específicas, evitando a paralização de todo o sistema, como já acontece na arquitetura TCP/IP da internet.

Adotar uma arquitetura com multicamadas permite melhorar o entendimento de como todos os aspectos mais importantes da arquitetura operam de forma independente, antes de serem integrados nos aplicativos de IoT. Esta abordagem modular ajuda a gerenciar a complexidade das soluções de IoT (GERBER, 2017).

Segundo Berger (2017), a figura 3 ilustra de maneira simplificada um possível modelo de arquitetura para a IoT.

Figura 3 – Camadas de uma possível arquitetura para IoT



Fonte: IBM, 2017.

A camada mais inferior, a dos dispositivos, engloba/corresponde a todo e qualquer objeto com a capacidade de coletar dados e enviá-los aos elementos da camada seguinte, conhecida como borda. Esta camada, por sua vez, realiza o pré-processamento dos dados coletados na camada subjacente, gerando, dessa forma, informação. A informação gerada é enviada para a camada superior, a nuvem, que terá como papel tornar a informação acessível a qualquer um e de qualquer lugar. Além disso, a análise dessas informações auxiliará na obtenção de conclusões, que por sua vez poderão motivar alguma ação (BERGER, 2017). A figura 4 ilustra os processos em que os dados estão envolvidos na arquitetura da Internet das Coisas,

desde o momento em que são coletados até o ponto em que poderão fomentar alguma ação.

Figura 4 – Processos em que os dados estão envolvidos



Fonte: MICROSOFT, 2018.

Outro fator muito importante quando se trata de aplicações IoT é a segurança, que diferentemente dos demais elementos deve estar presente em todas as camadas, atuando de ponta a ponta, evitando que um elemento comprometido prejudique toda a estrutura. Isto pode ser alcançado através de ações como, identificação de dispositivos, de aplicativos e de usuários, controle de acesso, utilização de criptografia na comunicação e uma série de outras medidas que visem garantir a integridade do sistema (BERGER, 2017).

No que se refere às aplicações, a Internet das Coisas apresenta uma gama praticamente ilimitada de possibilidades, podendo ser aplicada nas mais diversas áreas, entre as quais podemos destacar:

- Transportes
- Saúde
- Segurança
- Indústria
- Agricultura
- Energia e recursos
- Meio Ambiente

Ainda que a Internet das Coisas possa parecer algo para um futuro distante, já existem soluções em operação em algumas das áreas mencionadas acima. Na área dos transportes podemos citar, por exemplo, a cidade de Águas de São Pedro-SP, onde foram instalados 500 sensores de estacionamento nas principais vias da

cidade, possibilitando aos motoristas informarem-se da disponibilidade de vagas através de um aplicativo web. O projeto tinha por intuito otimizar o tráfego na região central da cidade em períodos de grande circulação de veículos, como alta temporada e finais de semana. Na cidade de São Francisco-EUA, 7 mil dos 28 mil pontos de paquímetro foram adaptados para aceitarem cartões de crédito e débito, além de ajustarem de forma dinâmica e periódica o preço do estacionamento de acordo com a demanda atual, incentivando os motoristas a buscarem por locais subutilizados e reduzindo a procura por áreas severamente congestionadas. O sistema informa, através do *site* ou de um aplicativo, a localização das vagas disponíveis e qual será o valor da tarifa (BNDES *et al.*, 2018).

Na área do meio ambiente, temos o exemplo da cidade de Paulínia-SP, onde foram instaladas cerca de 25 estações de coleta de lixo inteligentes, o sistema, um dos pioneiros na América Latina, informa à central quando os postos de coleta já se encontram cheios, reduzindo em até 30% os custos deste serviço. Na cidade de Itu-SP, foram espalhados 3.300 contêineres inteligentes, que alertam à central quando sua capacidade está próxima do limite ou quando há necessidade de reparo. A disposição dos contêineres levou em conta um estudo que indicou os principais geradores de resíduos (BNDES *et al.*, 2018).

Para não ficarem para trás, grandes empresas já lançaram suas plataformas de Internet das Coisas, a exemplo da IBM com a IBM Watson IoT Platform, da Amazon com a AWS IoT e da Microsoft com a Microsoft Azure IoT. Já empresas como a Cisco começam a apresentar seus primeiros equipamentos de infraestrutura pensados para atender às soluções de IoT, principalmente no que diz respeito ao setor industrial.

2.1.2 Protocolo MQTT

O protocolo de comunicação MQTT é voltado para aplicações M2M (*Machine-to-Machine*) e utiliza o paradigma *publish/subscribe* para troca de mensagens. O *layout* das mensagens foi pensado para atender a taxas de conexão de rede e hardwares modestos, mas que possibilitem em certo grau, garantia de entrega das

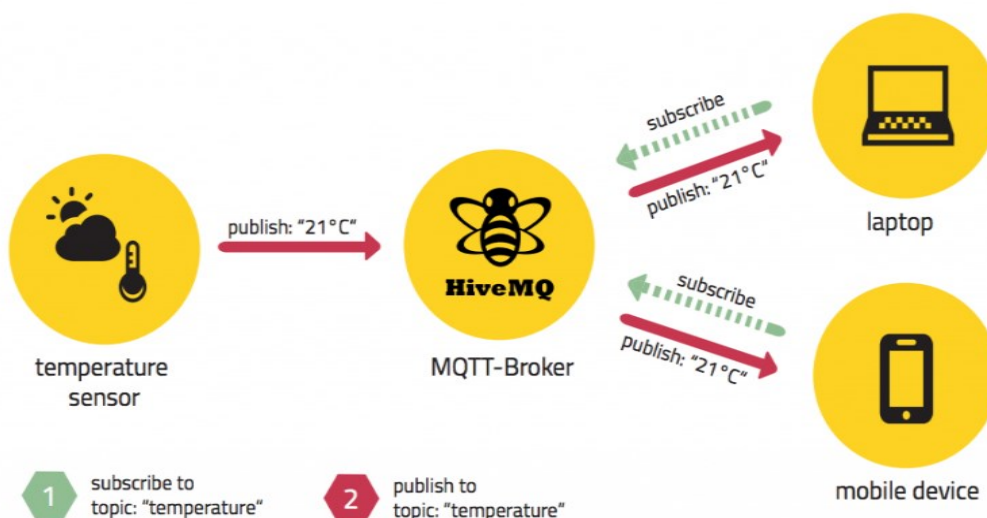
mensagens e de confiabilidade. Estas características fazem do MQTT um dos principais candidatos a protocolo de comunicação voltado para aplicações IoT ou M2M, e que também desponta quando se trata de aplicações destinadas a dispositivos móveis, onde a economia de energia é um quesito vital (MQTT.ORG, 2014).

“MQTT foi inventado pelo Dr. Andy Stanford-Clark da IBM, e Arlen Nipper da Arcom (atualmente EUROTECH), em 1999” (MQTT.ORG, 2014).

Sua aplicação original era vincular sensores em *pipelines* de petróleo a satélites. Como seu nome sugere, ele é um protocolo de mensagem com suporte para a comunicação assíncrona entre as partes. Um protocolo de sistema de mensagens assíncrono desacopla o emissor e o receptor da mensagem tanto no espaço quanto no tempo e, portanto, é escalável em ambientes de rede que não são confiáveis. Apesar de seu nome, ele não tem nada a ver com filas de mensagens, na verdade, ele usa um modelo de publicação e assinatura. No final de 2014, ele se tornou oficialmente um padrão aberto OASIS, com suporte nas linguagens de programação populares, usando diversas implementações de software livre (YUAN, 2017).

Diferente do modelo cliente/servidor, onde as mensagens são enviadas diretamente da origem para o destino, no modelo *publish/subscribe* existe a participação de uma entidade intermediária, o *Broker*, que irá receber as mensagens publicadas por um cliente, o *publisher*, e direcioná-las a um ou mais clientes, denominados *subscribers*. Dessa forma, uma camada extra de proteção é criada, quando este modelo é comparado àquele, pois ambos os clientes jamais se comunicarão diretamente. A figura 5 apresenta de forma mais clara o funcionamento do modelo *publish/subscribe*, utilizado pelo protocolo MQTT (HIVEMQ, 2015).

Figura 5 – Modelo *publish/subscribe* do protocolo MQTT



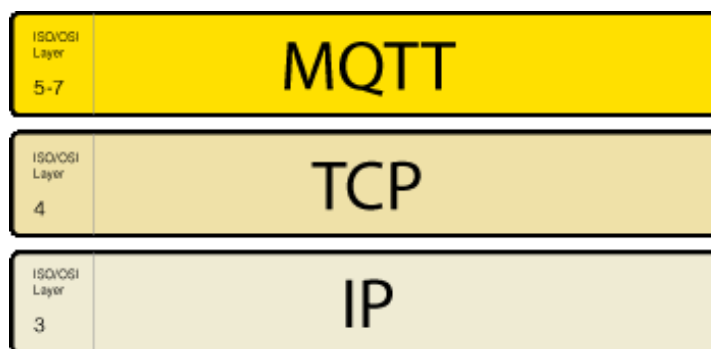
Fonte: HIVEMQ, 2015.

Como anteriormente mencionado, o protocolo MQTT proporciona um desacoplamento, entre os elementos comunicantes, nas dimensões espacial (*publisher* e *subscriber* não precisam se conhecer), temporal (não é necessário que *publisher* e *subscriber* estejam em execução simultaneamente) e de sincronização (durante publicações e recebimentos de mensagens, as operações em ambos os elementos não sofrem interrupção) (HIVEMQ, 2015).

Devido ao desacoplamento entre *publisher* e *subscriber*, a conexão acontece sempre entre cliente e *broker*. São considerados clientes MQTT, os elementos finais do processo de comunicação, ou seja, qualquer dispositivo com características de hardware suficientes para executar uma biblioteca MQTT e que esteja conectado a um *broker* MQTT; este, por sua vez, é responsável por receber todas as mensagens e filtrá-las, para que sejam enviadas aos respectivos interessados (HIVEMQ, 2015).

O protocolo da camada subjacente ao MQTT pode ser o TCP ou qualquer outro protocolo que ofereça conexão bidirecional, sem perdas e com ordenamento de dados (OASIS, 2015).

Figura 6 – Pilha TCP/IP, sobre a qual roda o protocolo MQTT



Fonte: HIVEMQ, 2015.

“O protocolo MQTT trabalha trocando uma série de pacotes de controle de maneira definida” (OASIS, 2015). Dentre as partes que compõe a estrutura dos pacotes de controle, uma que merece especial atenção é o cabeçalho fixo, pois está presente em todos os pacotes de controle MQTT (OASIS, 2015). A figura 7 ilustra o formato deste cabeçalho.

Figura 7 – Formato do cabeçalho fixo de um pacote de controle MQTT

bit	7	6	5	4	3	2	1	0
byte 1	Message Type				DUP flag	QoS level		RETAIN
byte 2	Remaining Length							

Fonte: IBM; EUROTECH, 2010.

O *byte 1* apresenta os campos *Message Type* (tabela II), *DUP flag*, *QoS level* e *RETAIN*. O *byte 2* apresenta o campo *Remaining Length* (IBM; EUROTECH, 2010).

Tabela II – Tipos de mensagem de um pacote de controle MQTT

Nome	Valor	Direção de fluxo	Descrição
Reservado	0	Proibido	Reservado
CONNECT	1	Cliente – <i>Broker</i>	Cliente solicita conexão ao <i>Broker</i>
CONNACK	2	<i>Broker</i> – Cliente	Reconhecimento de conexão
PUBLISH	3	Bidirecional	Publicar mensagem
PUBACK	4	Bidirecional	Reconhecimento de publicação
PUBREC	5	Bidirecional	Publicação recebida
PUBREL	6	Bidirecional	Publicação lançada

PUBCOMP	7	Bidirecional	Publicação completa
SUBSCRIBE	8	Cliente – <i>Broker</i>	Cliente solicita assinatura
SUBACK	9	<i>Broker</i> – Cliente	Reconhecimento de assinatura
UNSUBSCRIBE	10	Cliente – <i>Broker</i>	Pedido de cancelamento de assinatura
UNSUBACK	11	<i>Broker</i> – Cliente	Rec. de cancelamento de assinatura
PINGREQ	12	Cliente – <i>Broker</i>	Pedido de PING
PINGRESP	13	<i>Broker</i> – Cliente	Resposta de PING
DISCONNECT	14	Cliente – <i>Broker</i>	Cliente está desconectando
Reservado	15	Proibido	Reservado

Fonte: Adaptado de OASIS, 2015.

A tabela II traz os tipos de mensagens possíveis e seus respectivos valores, seguidos da direção do fluxo de dados e de uma breve descrição da função de cada mensagem. O campo *Message Type* vem definido nos *bits* de 7 a 4 do *byte* 1 (IBM; EUROTECH, 2010).

No *bit* 3 do *byte* 1, encontra-se o campo *DUP flag*, que deve ser setado quando o cliente ou o *broker* tentarem duplicar a entrega de um pacote MQTT (IBM; EUROTECH, 2010). Este procedimento (setar a *flag* DUP) se faz necessário quando o nível da Qualidade de Serviço (QoS), que será tratado na sequência, é um ou dois e uma mensagem de ACK (abreviação de *acknowledgment* - reconhecimento) é necessária (OASIS, 2015). Para uma mensagem com QoS definido em zero, a *flag* DUP não deve ser setada (IBM; EUROTECH, 2010).

Já os *bits* 2 e 1 do *byte* 1, são responsáveis por informa o nível de Qualidade do Serviço, ou seja, o grau de garantia da entrega de uma publicação. Os valores que o QoS pode assumir são mostrado na tabela abaixo (tabela III) (IBM; EUROTECH, 2010):

Tabela III – Níveis de garantia da entrega de um pacote de controle MQTT

Valor do QoS	Descrição	Significado
0	No máximo uma entrega	Enviar e esquecer
1	Pelo menos uma entrega	Recebe reconhecimento de entrega
2	Exatamente uma entrega	Entrega assegurada
3	Reservado	Não deve ser utilizado

Fonte: Adaptado de IBM; EUROTECH, 2010.

O último *bit* (0) do *byte* 1, associado à *flag* RETAIN, é responsável por informar se uma mensagem deve ou não ser mantida no *broker* após sua entrega aos atuais *subscribers* (IBM; EUROTECH, 2010).

O *byte* seguinte, *byte* 2, é responsável por informar o tamanho remanescente do pacote de controle MQTT, ou seja, o tamanho do cabeçalho variável acrescido do *payload*. Este campo, *Remaining Length*, utiliza de um a quatro *bytes*. Os *bits* no intervalo de 6 a 0 de cada *byte* informam a quantidade de *bytes* de dados, enquanto o oitavo *bit* (7), quando setado, informa a presença de mais *bytes* sendo utilizados pelo campo *Remaining Length*. Esta técnica permite à aplicação enviar mensagens de até 256MB (IBM; EUROTECH, 2010).

Tabela IV – Quantidade de *bytes* e os respectivos intervalos de valores

Bytes	Número inicial do intervalo	Número final do intervalo
1	0 (0x00)	127 (0x7F)
2	128 (0x80) (0x01)	16 383 (0xFF) (0x7F)
3	16 384 (0x80) (0x80) (0x01)	2 097 151 (0xFF) (0xFF) (0x7F)
4	2 097 152 (0x80) (0x80) (0x80) (0x01)	268 435 455 (0xFF) (0xFF) (0xFF) (0x7F)

Fonte: Adaptado de IBM; EUROTECH, 2010.

A primeira coluna da tabela IV informa a quantidade de *bytes* que está sendo utilizada pelo campo *Remaining Length*, enquanto a segunda e a terceira colunas informam os intervalos de valores possíveis para aquelas quantidades de *bytes*. Se, por exemplo, o tamanho do comprimento remanescente do pacote de controle MQTT for de até 128 *bytes* (ou seja, intervalo superior vale 127), o campo *Remaining Length* usará apenas um *byte* para expressar este valor. Mas se for, por exemplo, 401 *bytes* de comprimento remanescente (intervalo superior vale 400), serão necessários dois *bytes* para o campo *Remaining Length*. O quociente da divisão de 400 por 128 (3) é passado através do *byte* mais significativo (MSB), enquanto o resto da divisão de 400 por 128 (16) é passado através dos sete primeiros *bits* do *byte* menos significativo (LSB). Como foram necessários dois *bytes*, o oitavo *bit* do LSB é setado para indicar o uso de um segundo *byte*. Sendo assim, o

byte menos significativo carregará o valor 0x90 (128 + 16), e o *byte* mais significativo carregará o valor 0x03 (3) (IBM; EUROTECH, 2010).

Quanto ao cabeçalho fixo, ele pode ou não ser seguido de outras duas estruturas, o cabeçalho variável e o *payload*, sempre nesta ordem (OASIS, 2015).

A representação de dados adotado pelo protocolo MQTT atribui números de 7 a 0 para os *bits* dentro de um mesmo *byte*, sendo que o *bit* 7 é o mais significativo e o *bit* 0 é o menos significativo (OASIS, 2015). No processo de troca de dados, o fluxo de agrupamentos de *bits* respeita o formato *big-endian*, ou seja, o *byte* mais significativo é enviado/recebido primeiro, na sequência envia-se/recebe-se o *byte* menos significativo. Cada agrupamento de *bits* corresponde a 16 *bits* e é chamado de palavra ou inteiro de 16 *bits* (IBM; EUROTECH, 2010).

Os pacotes de controle MQTT utilizam a codificação UTF-8 (8-bit *Unicode Transformation Format*) nos campos de texto (OASIS, 2015).

2.2 METODOLOGIA

A metodologia para o desenvolvimento deste trabalho baseou-se em pesquisas bibliográficas e estudos a respeito do tema e dos componentes presentes na implementação; na elaboração de um protótipo com a montagem eletrônica realizada em uma *protoboard*; no desenvolvimento de códigos a serem executados na placa de prototipagem escolhida visando alcançar a proposta estabelecida; na escolha de uma interface gráfica voltada para aplicações IoT; na realização de testes para avaliação e validação do protótipo.

2.2.1 A plataforma de prototipagem

Dada as características do projeto, a plataforma escolhida teria que contar ao menos com funcionalidades *Wi-Fi* e suporte à utilização do protocolo de comunicação MQTT. Ademais, foram considerados os seguintes aspectos:

- Custo de aquisição
- Documentação
- Curva de aprendizagem

Quanto ao custo, a placa escolhida apresenta uma excelente relação custo/benefício se comparada a placas de prototipagem mais conhecidas como a Arduino UNO, que além de não contar com funcionalidades *Wi-Fi* (criando assim a necessidade do uso de um *shield Wi-Fi*), possui características de hardware bem modestas diante da NodeMCU.

Já no segundo quesito, a placa escolhida deixa a desejar, pois até o momento da elaboração desse trabalho, a documentação acumulada é bem pobre quando comparada com a que a Arduino acumula.

E por último, mas não menos importante, no que diz respeito a curva de aprendizagem, a diferença não se mostrou muito significativa, uma vez que, em pouco tempo, já era possível programar os módulos através da IDE Arduino apenas acrescentando algumas bibliotecas e adaptando a sintaxe às funcionalidades da placa. Soma-se a isso, a familiaridade com a placa, que já havia sido utilizado em projetos anteriores.

Sendo assim, optou-se pela utilização da NodeMCU Dev Kit v1.0, uma placa concebida pelo grupo de origem chinês *Amica* e que utiliza um módulo ESP (abreviação de *Espressif*), desenvolvido pela também chinesa, *Espressif Systems*. Geralmente estes módulos se apresentam sem pinos soldados e com distância entre contatos (2,0 mm) incompatível com o padrão usado nas *breadboards* (2,54 mm), o que dificulta a utilização e cria a necessidade do uso de adaptadores ou de montagens pouco técnicas. Com o intuito de eliminar essas carências, o grupo desenvolveu uma placa que torna extremamente acessível o uso dos módulos ESP, pois apresenta além de outros componentes passivos e ICs, o adaptador USB-serial CP2102 da *Silicon Labs*, o regulador de tensão NCP1117 de 3V3 DC, um conector micro-USB e barras de pinos de ambos os lados da placa para auxiliar em montagens com *protoboards* (NODEMCU, 2015).

O quadro I elenca as principais características das placas NodeMCU Dev Kit v1.0 e Arduino UNO R3.

Quadro I – Comparativo entre a NodeMCU v1.0 e a Arduino UNO R3

Especificações	NodeMCU Dev kit v1.0	Arduino UNO R3
MCU	Tensilica L106 32-bit	Atmega328 8-bit
Wi-Fi	802.11 b/g/n	Não
Frequência Típica	80/160 MHz	16 MHz
RAM	50-80 Kbytes	2 Kbytes
Memória <i>flash</i>	4 Mbytes	32 Kbytes
EEPROM	4 KB (emulada na memória <i>flash</i>)	1 Kbytes
GPIO	11	14
PWM	4 canais	6 canais
SPI/I2C/I2S/UART	2/1/1/2	1/1/0/1
ADC	1 canal 10-bit	6 canais 10-bit
Alimentação _{recomendada}	5-9 V	7-12 V
I_{MÁX} (por pino)	12 mA	40 mA

Fonte: Adaptado de NODEMCU, 2015; EMBARCADOS, 2013.

O módulo presente na placa é o ESP-12E, ele utiliza o SoC (*System on Chip* – Sistema em um Chip) ESP8266EX, que de acordo com o *datasheet* do fabricante, apresenta as características exibidas na figura 8 abaixo:

Figura 8 – Especificações técnicas do SoC ESP8266EX

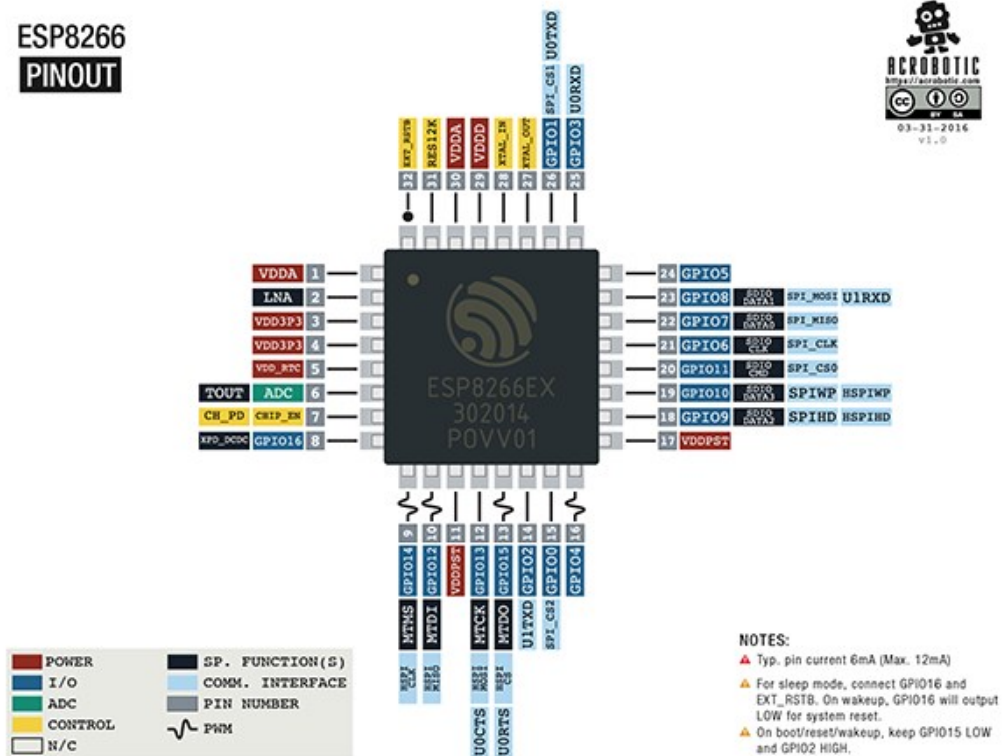
Categories	Items	Parameters
Wi-Fi	Certification	Wi-Fi Alliance
	Protocols	802.11 b/g/n
	Frequency Range	2.4G ~ 2.5G (2400M ~ 2483.5M)
	Tx Power	802.11 b: +20 dBm
		802.11 g: +17 dBm
		802.11 n: +14 dBm
	Rx Sensitivity	802.11 b: -91 dbm (11 Mbps)
		802.11 g: -75 dbm (54 Mbps)
		802.11 n: -72 dbm (MCS7)
	Antenna	PCB Trace, External, IPEX Connector, Ceramic Chip
Hardware	CPU	Tensilica L106 32-bit processor
	Peripheral Interface	UART/SDIO/SPI/I2C/I2S/IR Remote Control
		GPIO/ADC/PWM/LED Light & Button
	Operating Voltage	2.5V ~ 3.6V
	Operating Current	Average value: 80 mA
	Operating Temperature Range	-40°C ~ 125°C
	Storage Temperature Range	-40°C ~ 125°C
	Package Size	QFN32-pin (5 mm x 5 mm)
Software	External Interface	-
	Wi-Fi Mode	Station/SoftAP/SoftAP+Station
	Security	WPA/WPA2
	Encryption	WEP/TKIP/AES
	Firmware Upgrade	UART Download / OTA (via network)
	Software Development	Supports Cloud Server Development / Firmware and SDK for fast on-chip programming
	Network Protocols	IPv4, TCP/UDP/HTTP/FTP
	User Configuration	AT Instruction Set, Cloud Server, Android/iOS App

Fonte: ESPRESSIF, 2018.

A seguir, são apresentadas nas figuras 9 e 10, as pinagens do SoC e do módulo ESP-12E:

Figura 9 – Definição de pinos do ESP8266EX

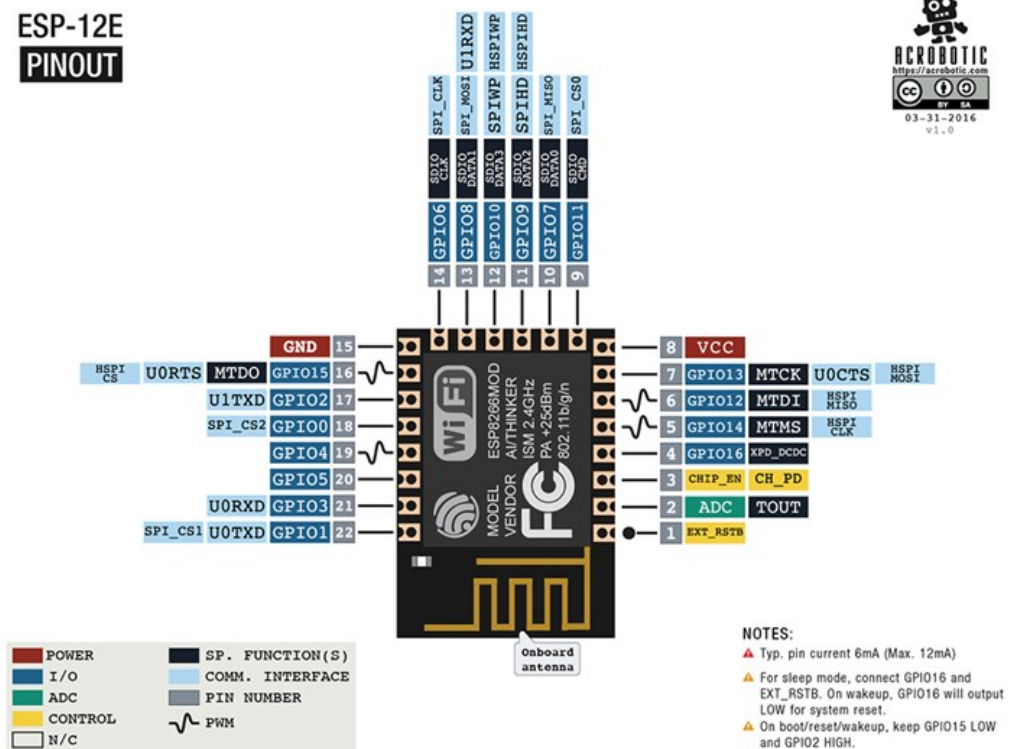
ESP8266 PINOUT



Fonte: ACROBOTIC, 2016.

Figura 10 – Definição de pinos do ESP-12E

ESP-12E PINOUT



Fonte: ACROBOTIC, 2016.

2.2.2 Sensor de fluxo

O sensor de fluxo de água utilizado neste projeto é o YF-S201B (figura 11), que apesar de apresentar qualidade inferior e documentação carente, tem custo acessível e atendeu bem ao propósito do projeto.

Figura 11 – Sensor de fluxo de água YF-S201B



Fonte: SILVATRONICS, 2018.

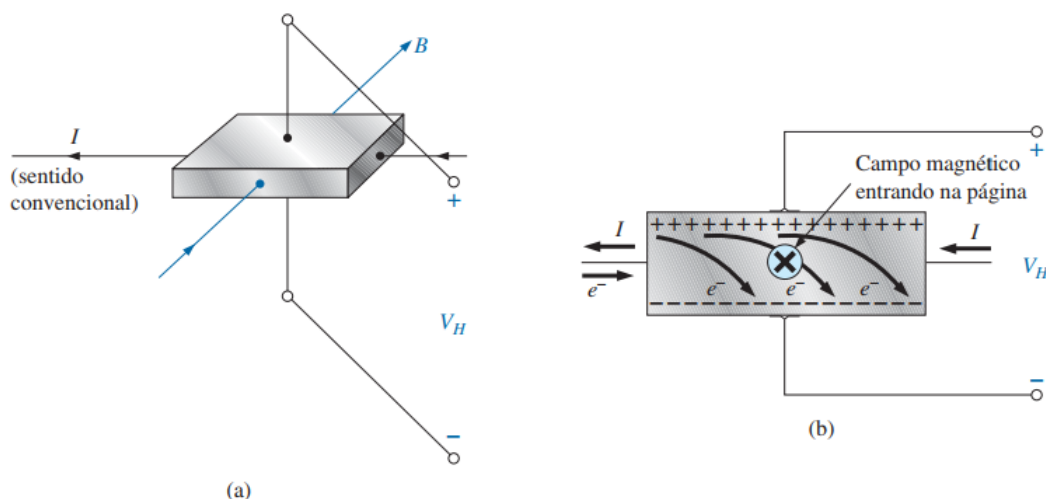
De acordo com o vendedor (*Hobbytronics*), o sensor apresenta as seguintes características:

- Modelo: YF-S201B
- Tipo de sensor: Efeito *Hall*
- Tensão de operação: 5~24V DC
- Corrente máxima: 15 mA (5V)
- Tipo de saída: TTL (Onda quadrada)
- Faixa de fluxo: 1~30 (l/min)
- Pressão máxima: 2 MPa
- Pulsos por litro: 450
- Frequência (Hz) = 7.5*Fluxo (l/min)
- Temperatura de trabalho: -25°C ~ +80°C

- Umidade de trabalho: 35% ~ 80% RH
- Precisão: $\pm 10\%$
- Tempo de borda de subida: $0.04 \mu\text{s}$
- Tempo de borda de descida: $0.18 \mu\text{s}$
- Durabilidade média: 300000 ciclos
- Comprimento do cabo: 15 cm
- Dimensão da conexão: 1/2"
- Diâmetro interno: 0.78"
- Dimensões: 2.5" x 1.4" x 1.4"

O funcionamento do sensor se deve a dois fatores, o primeiro deles é a presença de um elemento responsável por gerar um campo magnético (no caso do sensor utilizado, um pequeno ímã permanente acoplado à turbina), o segundo é o uso de um sensor de efeito *Hall*. Segundo Boylestad (2012), o sensor de efeito *Hall* consiste de um semicondutor que, percorrido por uma corrente elétrica I , ao se sujeitar à ação de um campo magnético B , fará surgir ortogonalmente à corrente e ao campo uma tensão de saída V_H , como ilustra a figura 12(a). A tensão é resultado da separação entre cargas positivas e negativas, causada pela força de Lorentz, figura 12(b). Graças a este efeito é possível construir mecanismos que nos permitem saber o número de vezes que o sensor de efeito *Hall* foi excitado, apenas contando o número de variações da tensão (pulsos).

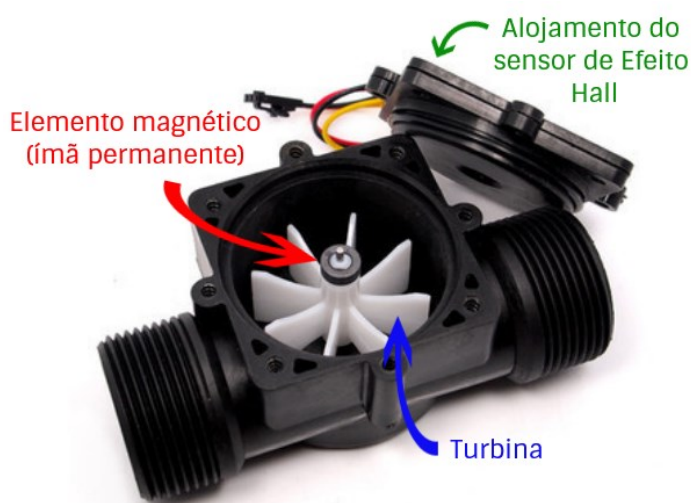
Figura 12 – Princípios de funcionamento de um sensor de efeito *Hall*



Fonte: BOYLESTAD, 2012.

Um desses mecanismos é o medidor de vazão, figura 13. Quando o fluxo de líquido provoca a movimentação da turbina e por conseguinte do ímã permanente, o sensor de efeito *Hall* sofre a ação do campo magnético gerado pelo ímã permanente a intervalos que são proporcionais à vazão do líquido. Sendo assim, quanto maior for o fluxo de líquido, maior será a frequência dos pulsos gerados na saída do sensor de efeito *Hall*.

Figura 13 – Elementos do sensor de fluxo de água YF-S201B



Fonte: SILVATRONICS, 2018.

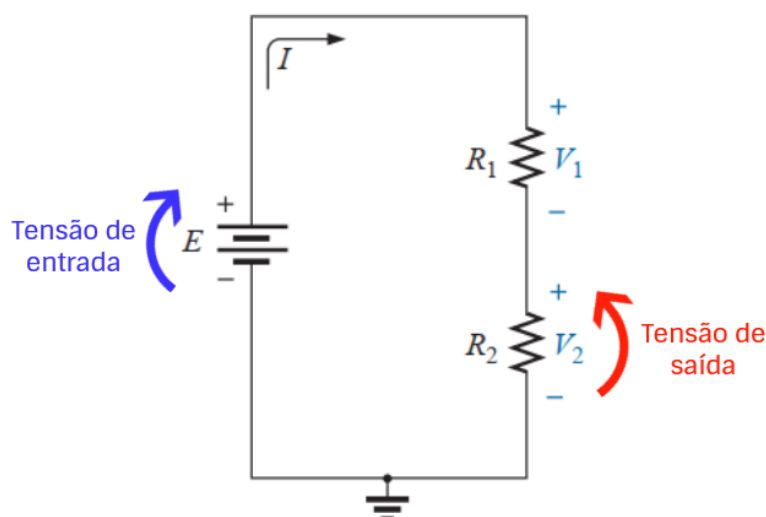
2.2.3 Esquema elétrico

Uma das vantagens da plataforma de prototipagem escolhida foi simplificar a montagem elétrica, pois a única necessidade apresentada foi adequar o sinal de saída do sensor de fluxo, uma vez que a faixa de alimentação do sensor (e consequentemente do sinal de saída) é incompatível com o nível de tensão de operação das GPIOs da placa, enquanto aquele está sendo alimentado com 5V (proveniente da interface USB), estas só trabalham com 3V3.

A adequação do nível de tensão foi feito com o auxílio de um divisor de tensão, solução que apesar de simples e barata, atendeu bem à necessidade.

Um divisor resistivo (figura 14), como também é chamado, nada mais é do que um arranjo de dois ou mais resistores em série, com valores escolhidos de modo a gerar na saída do circuito uma tensão desejada. Entretanto, o uso deste recurso fica limitado ao efeito de carga, que ocorre quando a resistência da carga a qual o circuito está conectado afeta o valor da resistência equivalente do divisor resistivo, o que por sua vez altera o valor de tensão de saída calculado previamente, comprometendo o sistema. Para encontrar uma equação que defina a tensão de saída $V_{saída}$ em função da tensão de entrada $V_{entrada}$ e dos resistores R_1 e R_2 , lançaremos mão da lei de Kirchhoff para tensões, segundo a qual a soma algébrica das tensões em um circuito fechado é igual a zero (BOYLESTAD, 2012).

Figura 14 – Divisor de tensão



Fonte: BOYLESTAD, 2012.

Para o equacionamento, adotaremos sinal positivo para elevação de tensão e sinal negativo para queda de tensão. Percorrendo a malha no sentido da corrente, ou seja, anti-horário, temos que:

$$+E - V_1 - V_2 = 0 \quad (1)$$

A tensão de entrada é a própria tensão da fonte, $V_{entrada} = E$. Além disso, para encontrarmos as quedas de tensão sobre os resistores, faremos uso de outra ferramenta, a lei de Ohm, segundo a qual a tensão sobre um condutor à temperatura constante é diretamente proporcional à corrente que o percorre. A constante de

proporcionalidade entre estas duas grandezas é conhecida como resistência elétrica. O condutor sujeito a esta lei é conhecido como resistor ôhmico (BOYLESTAD, 2012).

Pela lei de Ohm, temos que:

$$V_1 = R_1 \times I \quad (2)$$

$$V_2 = R_2 \times I \quad (3)$$

Substituindo (2) e (3) em (1), temos:

$$\begin{aligned} +V_{entrada} - V_1 - V_2 &= 0 \\ +V_{entrada} - R_1 \times I + R_2 \times I &= 0 \\ V_{entrada} &= (R_1 + R_2) \times I \end{aligned} \quad (4)$$

A tensão de saída é igual à queda de tensão sobre o resistor R_2 , logo:

$$\begin{aligned} V_{saída} &= R_2 \times I \\ I &= \frac{V_{saída}}{R_2} \end{aligned} \quad (5)$$

Portanto, substituindo (4) em (5), encontramos a seguinte relação, que descreve o comportamento da tensão de saída do divisor de tensão em função da tensão de entrada e dos valores dos resistores:

$$V_{saída} = \left(\frac{R_2}{R_1 + R_2} \right) \times V_{entrada} \quad (6)$$

Pela equação (6), podemos concluir que a tensão de saída será sempre uma fração da tensão de entrada, uma vez que razão entre os resistores é sempre menor do que 1.

Para o projeto abordado, temos que a tensão de entrada é de 5V, porém necessitamos de uma tensão de saída de aproximadamente 3V3. Sendo assim, utilizaremos a equação (6) para encontrar uma relação entre R_1 e R_2 .

$$\begin{aligned}
3.3 &= \left(\frac{R_2}{R_1 + R_2} \right) \times 5 \\
0.66 &= \left(\frac{R_2}{R_1 + R_2} \right) \\
0.66 \times (R_1 + R_2) &= R_2 \\
0.66 \times R_1 + 0.66 \times R_2 &= R_2 \\
0.66 \times R_1 &= (1 - 0.66) \times R_2 \\
0.66 \times R_1 &= 0.34 \times R_2 \\
\mathbf{1.94} &\cong \left(\frac{R_2}{R_1} \right)
\end{aligned} \tag{7}$$

Através da relação obtida em (7), podemos encontrar o valor de um dos resistores se fixarmos o valor do outro. Para o valor de resistência a ser fixado, façamos duas considerações:

- Limitar a potência dissipada
- Escolher um valor disponível comercialmente

Para que possamos limitar a potência dissipada, faz-se necessário utilizar um resistor da ordem de 10^3 Ohms, pois um valor alto de resistência limitará a corrente e conseqüentemente a potência dissipada, uma vez que esta varia com o quadrado daquela (BOYLESTAD, 2012), segundo a equação (8):

$$P_{dissipada} = R \times I^2 \tag{8}$$

Tendo feito esta consideração, o valor escolhido para R_2 será de $22K\Omega$, valor este disponível comercialmente. Para encontrar R_1 , utilizaremos a relação obtida em (8):

$$\begin{aligned}
1.94 &\cong \frac{22K\Omega}{R_1} \\
R_1 &\cong 11.34K\Omega
\end{aligned}$$

Como o valor obtido inexistia comercialmente, teremos de associar um resistor de 10KΩ a um de 1KΩ, obtendo com isso 11KΩ. Todavia, os cálculos da tensão de saída deverão ser refeitos, uma vez que o valor de R_1 foi ajustado. Pela equação (7), temos:

$$V_{saída} = \left(\frac{22K}{11K + 22K} \right) \times 5$$

$$V_{saída} \cong 3.33V$$

Segundo o manual da fabricante do módulo ESP-12E (*Ai-Thinker*), o valor de tensão de entrada admitido como sendo nível lógico alto deve estar entre 0.75VDD e VDD+0.3, considerando VDD = 3V3. Portanto, os valores encontrados para R_1 e R_2 e o ajuste realizado em R_1 são satisfatórios, uma vez que o valor da tensão de saída recalculado está dentro do que exigem as especificações ($2.48 \leq 3.33 \leq 3.60$).

Vale ressaltar que a solução utilizada neste projeto para a adequação do nível de tensão só foi possível graças ao elevado valor da impedância de entrada da porta digital (da ordem de 10^6 Ohms), caso contrário o ajuste da tensão seria comprometido devido ao efeito de carga mencionado anteriormente.

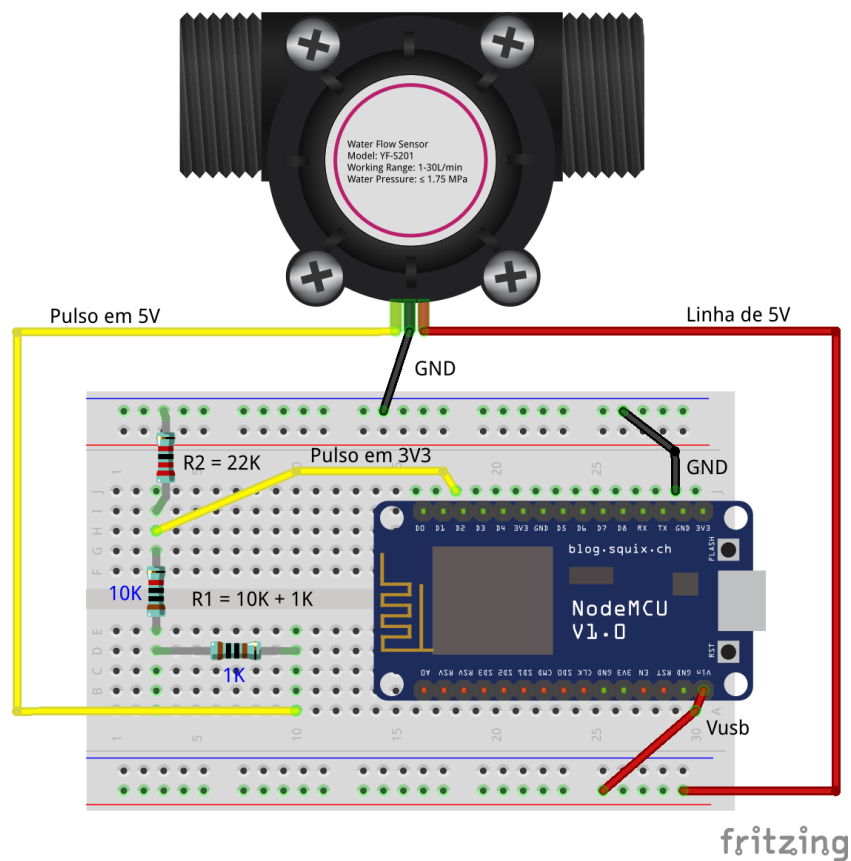
Concluído os cálculos, procedeu-se com a montagem do protótipo, tal como exibido nas figuras 15 e 16, utilizando-se os seguintes componentes (tabela V):

Tabela V – Componentes utilizados na montagem do protótipo

Componentes	Quantidade
Matriz de contatos – 400 pontos	1
Sensor de fluxo YF-S201B ½"	1
Placa NodeMCU v1.0	1
Resistor 1/4W 1KΩ ±1%	1
Resistor 1/4W 10KΩ ±1%	1
Resistor 1/4W 22KΩ ±1%	1
Jumpers M-M 10cm	3
Jumpers M-M 20cm	3

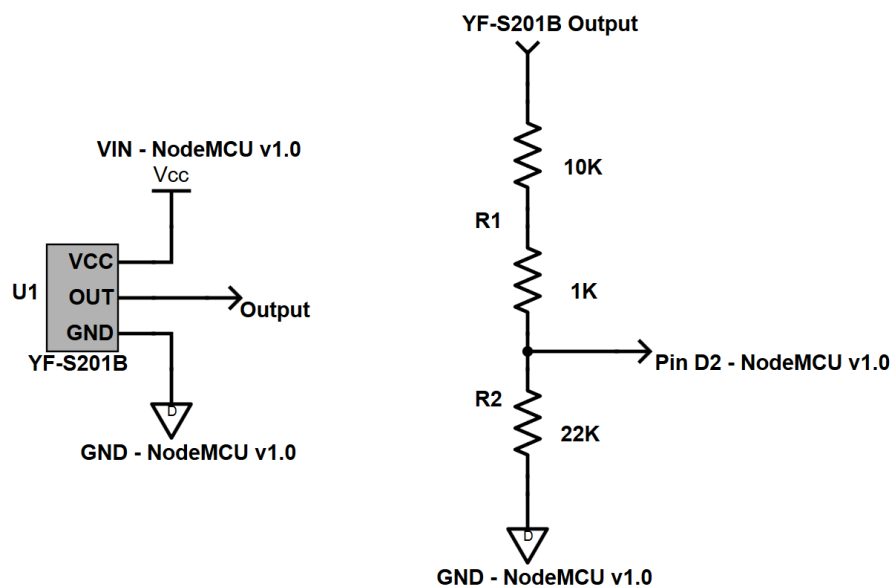
Fonte: PROPRIA, 2018.

Figura 15 – Montagem do protótipo gerada no *Fritzing*



Fonte: PRÓPRIA, 2018.

Figura 16 – Esquemático das conexões do protótipo

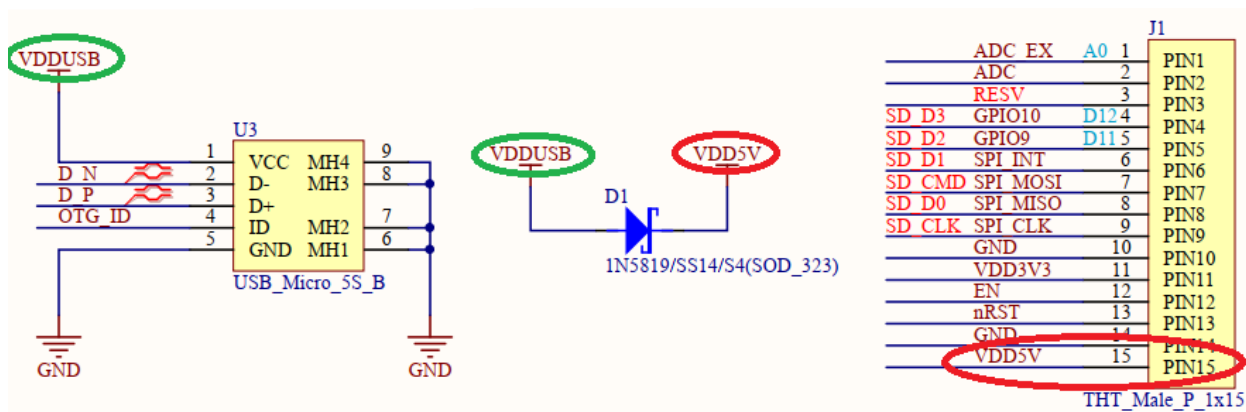


Fonte: PRÓPRIA, 2018.

Por se encontrar ainda em fase de protótipo, a placa foi alimentada através da porta micro-USB (5V DC), conectada à entrada USB do computador. De acordo com

a documentação da fabricante da placa (*Amica*), a tensão proveniente da entrada micro-USB é disponibilizada no pino 15 (PIN15) da barra de pinos J1 da NodeMCU após passar pelo diodo D1, como mostra a figura 17.

Figura 17 – Pino VDD5V da NodeMCU



Fonte: NODEMCU, 2015.

Esta tensão (VDD5V) foi utilizada para alimentar a linha positiva inferior da *breadboard*, e consequente o sensor de fluxo. O sinal que sai do YF-S201B (~5V) passa pelo divisor resistivo e após sofrer duas quedas de tensão (provocadas pelos resistores de 1KΩ e 10KΩ em série) é entregue (com ~3.33V) no pino D2 (GPIO 4) da placa de prototipagem, uma GPIO configurada como entrada digital e que permite o uso de interrupções.

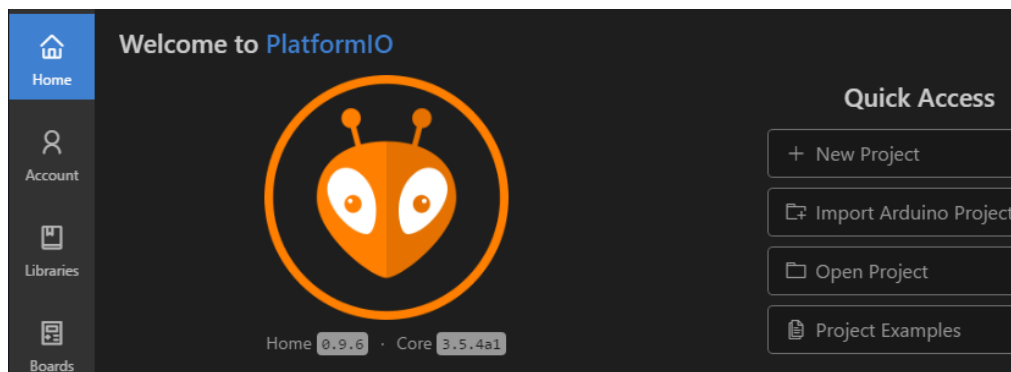
2.2.4 O ambiente de programação

Para o desenvolvimento do software, utilizou-se a integração PlatformIO + Atom (Figura 18). PlatformIO é um ecossistema de código aberto voltado para o desenvolvimento de aplicações de IoT. Possui suporte a diversas plataformas de desenvolvimento, dentre as quais:

- Atmel AVR
- Microchip PIC32
- ST STM32

- Espressif 32
- Espressif 8266

Figura 18 – Integração PlatformIO + Atom



Fonte: PRÓPRIA, 2018.

Atualmente, o PlatformIO permite a utilização de mais de 400 placas de desenvolvimento, dentre as quais a Arduino UNO e a NodeMCU v1.0. A plataforma conta com uma variedade de ferramentas pré-instaladas, depuradores, diversos *frameworks* e recursos para subir os códigos desenvolvidos para as placas, sendo suportada pelos principais OS (*Operating Systems* - Sistemas Operacionais), como Windows, Linux e Mac. Tal como no ambiente de desenvolvimento Arduino, é possível instalar uma infinidade de bibliotecas criadas por usuários e que ficam hospedadas no repositório *GitHub* (PLATFORMIO, 2018).

Atom é um editor de textos repleto de recursos e extremamente versátil, pois oferece ao usuário a possibilidade de instalar uma variedade de pacotes, além de permitir a utilização de diversas linguagens de programação, como C/C++ e Python. Também possui código aberto e é suportado pelos principais OS (ATOM, 2018).

2.2.5 Desenvolvimento do software

O código utilizado na implementação do projeto (desenvolvido em C/C++) encontra-se no apêndice ao final do trabalho, completo e devidamente comentado.

Figura 19 – Fluxograma do processo executado pelo código

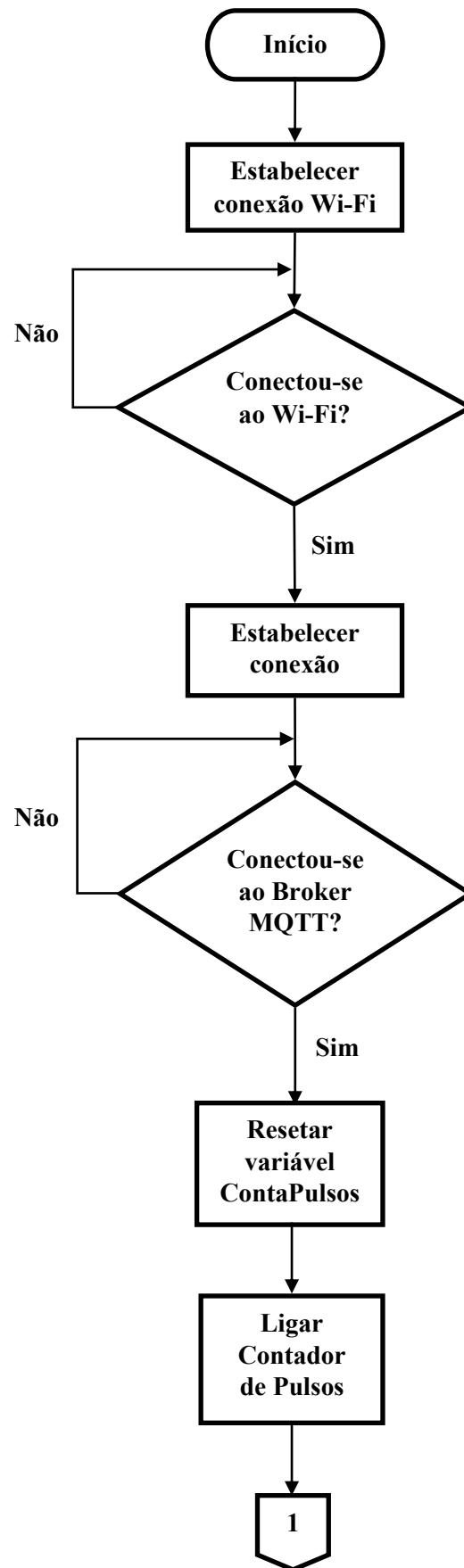
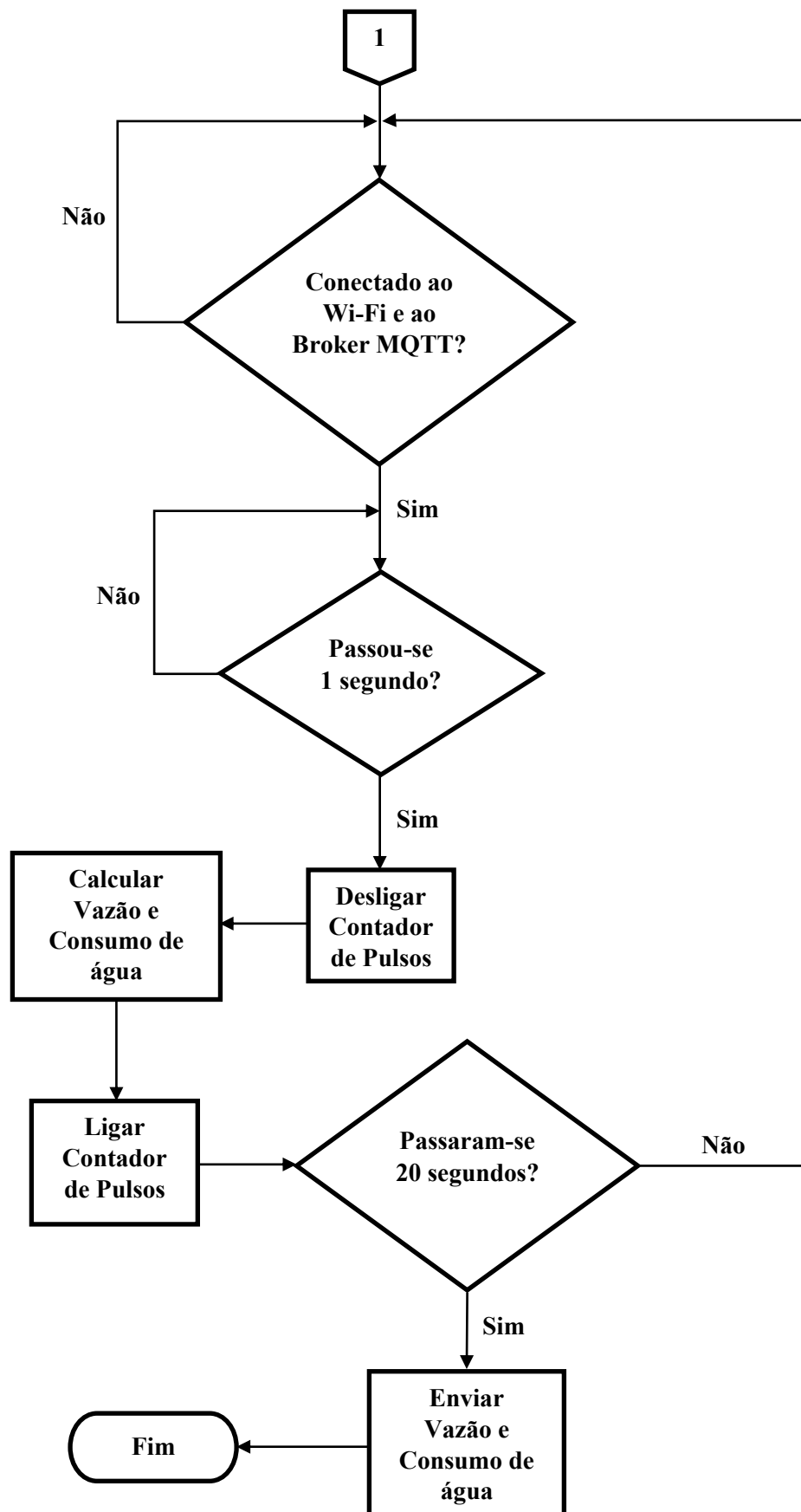


Figura 20 – Continuação do fluxograma anterior

Fonte: PRÓPRIA, 2018.

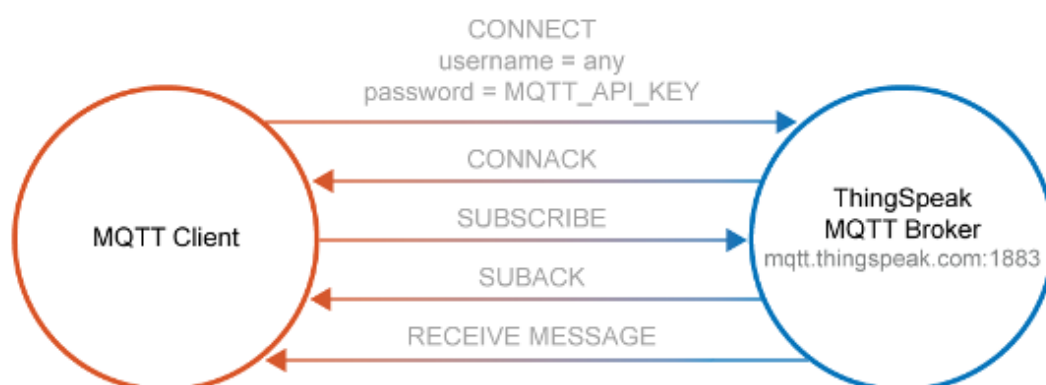
Afim de auxiliar na compreensão, foi utilizada uma representação por fluxograma (figuras 19 e 20) que ilustra de maneira simplificada o processo executado pelo algoritmo embarcado na placa de prototipagem.

Na elaboração do código, foram utilizadas duas bibliotecas, a primeira dela, WiFiClient, fornece os recursos necessários para que seja possível estabelecer uma conexão *Wi-Fi*, enquanto que a segunda, PubSubClient, provê os meios necessários para a utilização do protocolo MQTT.

Na primeira etapa, o código realiza a conexão com a rede *Wi-Fi* através da função `setWiFiConnection`. Enquanto a conexão com a rede *Wi-Fi* não é estabelecida, novas tentativas serão efetuadas. Uma mensagem de sucesso será enviada à porta serial assim que a conexão for bem sucedida.

Estabelecida a conexão *Wi-Fi*, é a vez de conectar-se ao *broker* MQTT. A função encarregada desta tarefa é a `setMQTTConnection`. Inicia-se configurando o *broker* MQTT ao qual se deseja conectar e a porta da respectiva conexão. Para o presente trabalho, utilizou-se o *broker* `mqtt.thingspeak.com` conectado à porta 1883 (com o TCP no papel de protocolo de camada subjacente e sem o uso de criptografia), como sugere uma das recomendações da documentação da plataforma (MATHWORKS, 2018). São passados como parâmetros da conexão MQTT, o identificador da sessão/conexão (`mqttID`), o nome do cliente (`mqttUserName`) e a senha do canal (`mqttAPIKey`), respectivamente. Tal como acontece na conexão *Wi-Fi*, a função `setMQTTConnection` só é abandonada quando a conexão com o *broker* é bem sucedida; sendo que, uma vez estabelecida a conexão, o cliente efetuará a assinatura do tópico no qual ele começará a publicar em instantes. Este passo final do processo de conexão ao *broker* ThingSpeak será melhor justificado na sessão de Resultados e Discussão. A figura 21 ilustra as etapas do processo de assinatura a um determinado tópico, desde a conexão ao *broker* MQTT até o início do recebimento das mensagens destinadas àquele tópico.

Figura 21 – Processo de assinatura a um determinado tópico



Fonte: MATHWORKS, 2018.

Segundo a documentação, ao efetivar a assinatura a um determinado tópico, o assinante receberá a última publicação daquele tópico, se houver alguma (MATHWORKS, 2018).

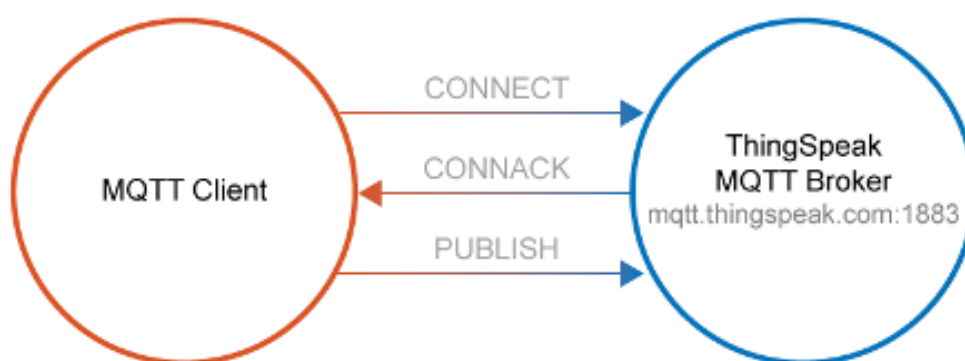
O recebimento das publicações direcionadas a um determinado tópico é feito através da função callback, que é acionada sempre que ocorre um evento de publicação no tópico ao qual se está inscrito (O'LEARY, 2018). Pode ou não haver uma ação definida caso este evento ocorra. No nosso caso a ação é armazenar o último valor (publicado no tópico) na variável `consumption`, que é responsável por guardar o consumo acumulado.

As etapas até agora descritas correspondem às ações iniciais realizadas pelo código. Quanto ao processo principal, este acontece na função `loop`. Uma vez dentro desta função, o primeiro procedimento é verificar se as conexões *Wi-Fi* e MQTT estão ativas, caso contrário deverão ser efetuadas.

Na sequência, visando obter uma base fixa de tempo, uma condicional fora utilizada. Esta base de tempo, definida em 1 segundo, é fundamental para os cálculos da vazão e do consumo. A cada 1 segundo o processo de registro de interrupções é cessado (evitando erros de cálculo) e a função `calculeFlowRateAndConsumption` é chamada. O valor da vazão é obtido da razão entre o número de interrupções registradas e a constante que define a quantidade de pulsos por litro. O resultado é o valor da vazão em litros por segundo. Já o consumo é obtido do acúmulo sucessivo dos valores de vazão. Ao final, o valor da vazão é convertido para litros por minuto quando multiplicado por 60. Terminado os cálculos, a variável que armazena o número de interrupções é resetada e o processo de registro de

interrupções é restabelecido. Os valores de vazão e consumo são enviados a intervalos de 1 segundo para a porta serial, enquanto que para as publicações no *broker* MQTT, o intervalo dependerá do que foi definido na variável *interval* (que para este projeto, foi definida em 20 segundos), sendo que este valor não deverá ser menor do que 15 segundos (para as contas gratuitas na plataforma ThingSpeak), tal como recomenda a documentação da plataforma (MATHWORKS, 2018). Após estabelecida uma conexão, os parâmetros nome do tópico e mensagem deverão ser passados à função que realiza a publicação. A figura 22 ilustra as etapas envolvidas no processo de publicação de mensagens pela aplicação.

Figura 22 – Processo de publicação em um dado tópico



Fonte: MATHWORKS, 2018.

Um ponto fundamental para a manutenção da conexão com o *broker* e o processamento das mensagens recebidas, é chamar regularmente a função de loop MQTT (`mqttClient.loop`) (O'LEARY, 2018).

Além das funções já descritas, foram definidas outras quatro funções que abrangem todo o processo relativo às interrupções. A primeira delas, que é responsável por dar início ao processo de contagem de pulsos (habilitando as interrupções), utiliza três parâmetros segundo a documentação da placa Arduino (2018), o pino ao qual as interrupções estão associadas (para este projeto é o pino D2 da NodeMCU v1.0), a função a ser chamada quando ocorrer uma interrupção (`PulsesCounter`) e qual tipo de evento deverá ser interpretado como uma interrupção (o evento utilizado no projeto foi o de subida - `RISING`). A segunda função, a qual é responsável pelo tratamento das interrupções, incrementa a variável armazenadora de interrupções sempre que é chamada. Esta função é a ISR (*Interrupt Service*

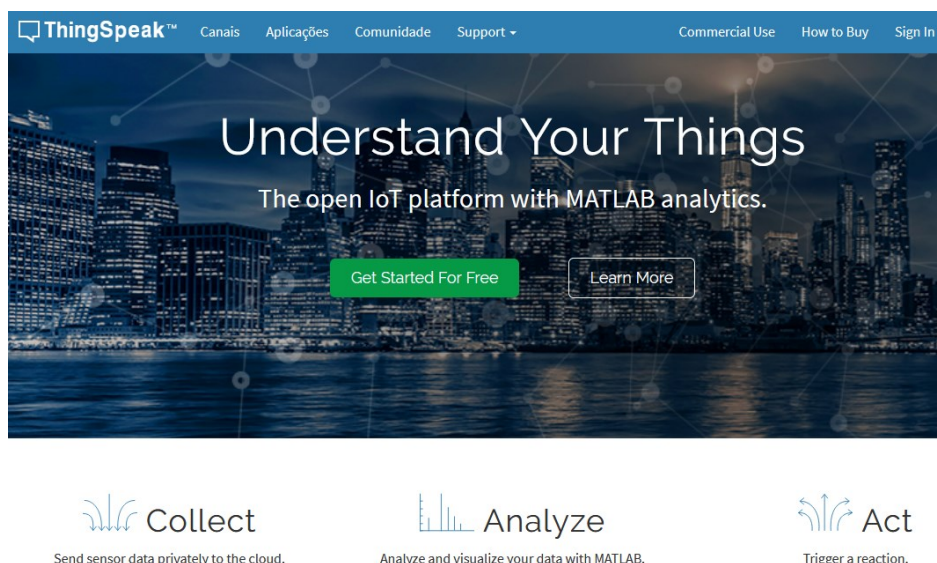
Routine – Rotina de Serviço de Interrupção) propriamente dita. A terceira função, que age de maneira oposta à primeira, é responsável por suspender o processo de contagem de pulsos ao desabilitar as interrupções. A última função que permeia este processo é responsável por zerar a variável que armazena o número de interrupções.

2.2.6 Interface gráfica

Uma maneira eficaz de exibir dados é através da utilização de imagens, pois além de serem mais intuitivos do que textos, reduzem significativamente a quantidade de informação apresentada. Este foi um dos pontos que pesaram na escolha do ThingSpeak (figura 23), uma plataforma com serviços de análise voltada para IoT e integrada ao MATLAB, que permite visualizar, analisar e processar fluxos de dados na nuvem. A plataforma permite aos usuários projetarem e construírem aplicações IoT sem a necessidade de configurar servidores ou desenvolver aplicações web. As principais funcionalidades do ThingSpeak incluem:

- Configuração de dispositivos para o envio de dados usando REST API ou MQTT;
- Gerar visualizações a partir de dados instantâneos ou histórico de dados;
- Pré-processamento e análise de dados através da integração com o MATLAB;
- Iniciar a análise automática de dados de forma programada ou eventual;
- Atuar a partir de dados recebidos, comunicando a serviços de terceiros como o *Twitter* (MATHWORKS, 2018).

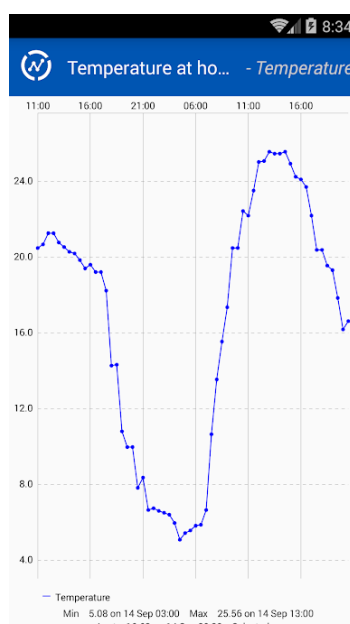
Figura 23 – Plataforma de interface gráfica ThingSpeak



Fonte: THINGSPEAK, 2018.

Outros dois pontos pesaram na escolha do ThingSpeak, a possibilidade de obter uma conta gratuita com recursos que atendessem à proposta do trabalho e o suporte à utilização do protocolo de comunicação MQTT, um dos objetos de estudo do presente trabalho. A plataforma, destinada inicialmente para *desktops/notebooks*, conta também com o ThingView (desenvolvido pela *Cinetica*), um aplicativo voltado para dispositivos móveis que permite o acesso a canais do ThingSpeak, bastando apenas do número de identificação do canal que se queira monitorar.

Figura 24 – Captura de tela do aplicativo ThingView



Fonte: GOOGLE, 2018.

2.3 RESULTADOS E DISCUSSÃO

Nesta sessão serão abordados os resultados obtidos na implementação do trabalho. Após incontáveis revisões no código, seguidas de inúmeras falhas, a implementação do protótipo obteve seus primeiros sucessos. Contudo, uma nova questão foi apresentada. E se houvesse falha da alimentação do módulo ou se por algum motivo ele sofresse um *reset*, como lidar com os dados que precisam estar sendo constantemente acessados e registrados (como é o caso da variável que armazena o consumo)?

Talvez a primeira ideia fosse gravá-los frequentemente em uma memória não-volátil, para que estes dados pudessem ser posteriormente retomados após uma eventual queda do sistema. Porém, segundo a documentação do ESP8266EX, não há presença de memória ROM programável no SoC (ESPRESSIF, 2018), diferentemente do ATMEGA328, que embarca 1KB de EEPROM (EMBARCADOS, 2013). Felizmente, são oferecidas duas soluções para sanar este problema, a primeira delas, segundo Minatel (2016), é utilizar a “EEPROM” emulada sobre a memória *flash* (com aproximadamente 4KB); a segunda solução é fazer uso do sistema de arquivos SPIFFS (que no caso da NodeMCU v1.0, pode ser configurado com 1MB ou 3MB de capacidade). Em ambas as soluções, existem bibliotecas que auxiliam na implementação (PEDROMINATEL, 2017). Apesar de se mostrarem razoáveis, estas soluções devem ser cuidadosamente analisadas antes de serem implementadas, pois carregam uma limitação imposta pela memória sobre a qual elas operam, a *flash*. Segundo o manual de uma das fabricantes do modelo de memória flash utilizado pelos módulos ESP, é possível executar cerca de 100.000 operações de escrita (WINBOND, 2007b). Ainda que seja um valor consideravelmente elevado, este número de ciclos de escrita pode ser rapidamente atingido se mal administrado.

Após um tempo repassando as características do protocolo MQTT, uma nova solução foi encontrada. Como já mencionado na descrição do protocolo, uma das *flags* que compõe a parte fixa do cabeçalho dos pacotes MQTT é a RETAIN.

Considerando um determinado tópico, quando esta *flag* é setada, o *broker* é informado de que a mensagem enviada ao tópico deve ser armazenada mesmo após ser entregue aos atuais assinantes, juntamente com o valor de QoS; quando uma nova assinatura é realizada para este tópico, cabe ao *broker* entregar a última mensagem com a *flag* RETAIN setada ao novo assinante do tópico em questão. Entretanto, quando o nível de Qualidade de Serviço da mensagem é zero e a *flag* RETAIN é setada, o *broker* deve armazenar a nova mensagem com a *flag* RETAIN setada e descartar as anteriores, podendo optar inclusive por descartar esta última mensagem, ou seja, se um novo cliente assinasse o tópico em questão, não receberia nenhuma mensagem (OASIS, 2015).

Portanto, se o *broker* tem por regra manter as mensagens com a *flag* RETAIN setada, o fato de setar esta *flag* garante que o *broker* forneça um serviço de *backup* dos dados enviados a ele. Logo, toda vez que ocorrer uma baixa do sistema, basta que ao reiniciá-lo, o sistema assine o tópico no qual até pouco tempo atrás ele estava publicando e, ao receber a última mensagem por ele mesmo enviada, carregá-la na devida variável.

Sendo assim, era necessário descobrir qual a procedência adotada pelo *broker* utilizado no presente trabalho. Segundo a documentação, o *broker* ThingSpeak só implementa o protocolo MQTT com nível de QoS 0 e recomenda que a *flag* RETAIN não seja setada (MATHWORKS, 2018). Não foram encontradas informações a respeito do tratamento dado às mensagens, ou seja, se estas são ou não mantidas pelo *broker*, entretanto é bem provável que ao menos a última mensagem publicada em um dado tópico seja retida pelo *broker* ThingSpeak, pois de acordo com a documentação, um novo assinante de um dado tópico receberá a última mensagem publicada neste tópico, se houver alguma (MATHWORKS, 2018). Com o intuito de certificar-se deste fato, o seguinte experimento foi realizado. Um trecho de código fora inserido na função callback de modo que a informação capturada fosse exibida na porta serial. Vale lembrar que o papel da função callback é reagir aos eventos de publicação no(s) tópico(s) ao(s) qual(is) se está assinado, recuperando a(s) mensagem(s) publicada(s). Sendo assim, a última mensagem publicada no tópico assinado (e que estava recebendo publicações deste mesmo dispositivo até recentemente) deveria ser visualizada. O resultado deste experimento foi obtido após um *reset* do sistema e é mostrado na figura 25 abaixo.

Figura 25 – Captura da função callback após um *reset* do sistema

```

Tentando se conectar ao Broker MQTT mqtt.thingspeak.com
Conectado com sucesso ao broker MQTT!

ets Jan  8 2018, rst cause:2, boot mode:(3,7)

load 0x4010f000, len 1384, room 16
tail 8
chksum 0x2d
csum 0x2d
v614f7c32
~ld

-----Conexão WI-FI-----

Conectando-se na rede Casa
Aguarde.....
Conectado com sucesso na rede Casa
IP obtido: 192.168.1.6

-----Conexão ao Broker MQTT-----

Tentando se conectar ao Broker MQTT mqtt.thingspeak.com
Conectado com sucesso ao broker MQTT!

Mensagem da função callback: 40.02
Vazão: 0.00 l/min
Consumo: 40.02 litros

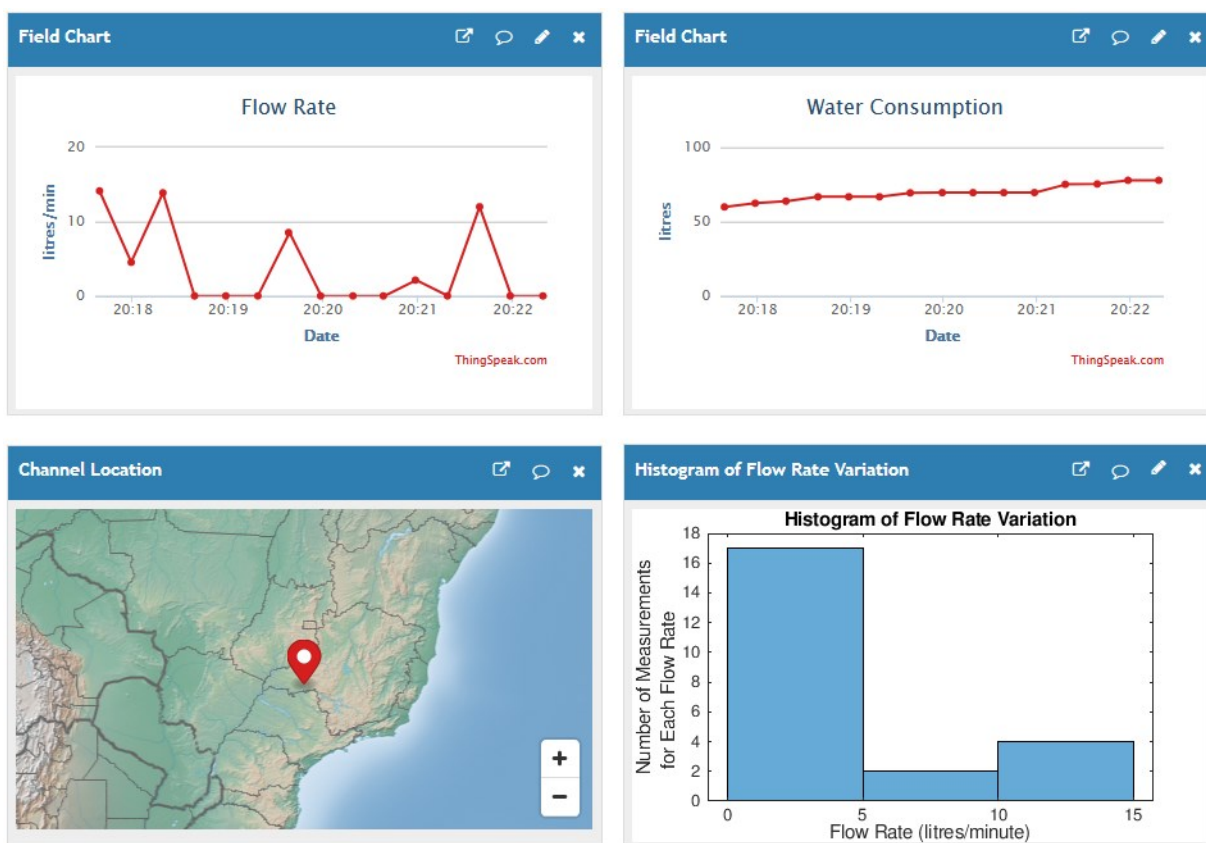
```

Fonte: PRÓPRIA, 2018.

Após vários testes, esta solução apresentou excelentes resultados e vem sendo utilizada até o momento, pois através de recursos do próprio protocolo MQTT, foi possível proteger as informações importantes contra quedas do sistema sem, contudo, comprometer a memória *flash*.

Na figura 26 é apresentado o resultado final da implementação, o *dashboard* do supervisor criado na plataforma ThingSpeak, que permite monitorar a vazão e o consumo, além de informar a localização do sistema monitorado. O primeiro quadro traz um gráfico da vazão (em litros/minuto) pela data (e hora do dia), o segundo quadro apresenta um gráfico do consumo acumulado até aquela data e hora, enquanto que o terceiro quadro exibe a localização aproximada do sistema.

Figura 26 – *Dashboard* do sistema implementado



Fonte: PRÓPRIA, 2018.

Como já mencionado, uma das vantagens da utilização do ThingSpeak, é sua integração com o MATLAB. Na figura 26, o último quadro apresenta um histograma da variação da vazão nos últimos 30 minutos. Este histograma foi obtido do processamento das informações enviadas à plataforma e que foram devidamente tratadas e apresentadas com o auxílio das ferramentas fornecidas pelo MATLAB.

Quanto aos cálculos de vazão e consumo, é importante frisar que eles carregam um erro mínimo a cada iteração, uma vez que, para realizá-los, as interrupções são desabilitadas. Todavia o acumulado destes erros pode atingir um valor significativo a longo prazo e deverá ser tratado em implementações futuras para que a proposta central do trabalho, uso eficiente da água, seja assegurada.

No que diz respeito a trabalhos futuros que venham a utilizar o SoC ESP8266EX, serão feitas duas considerações. Primeiro quanto à economia de energia e, segundo, quanto a atualizações de código remotas.

Em se tratando de implementações onde o fator energético é crucial, o ESP8266 pode se mostrar uma excelente escolha, uma vez que conta com 3 modos

de operação no que se refere ao consumo de energia, possibilitando uma economia extremamente significativa. A figura 27 ilustra estes modos, apresentando suas características e os valores aproximados de corrente.

Figura 27 – Modos de economia de energia do SoC ESP8266

Item		Modem-sleep	Light-sleep	Deep-sleep
Wi-Fi		OFF	OFF	OFF
System clock		ON	OFF	OFF
RTC		ON	ON	ON
CPU		ON	Pending	OFF
Substrate current		15 mA	0.4 mA	~ 20 μ A
Average current	DTIM = 1	16.2 mA	1.8 mA	
	DTIM = 3	15.4 mA	0.9 mA	-
	DTIM = 10	15.2 mA	0.55 mA	

Notes:

- SDK provides interfaces to enable Modem-sleep and Light-sleep modes, and the system bottom layer decides when to go into sleep. For details, please refer to **Chapter 2. Modem-sleep** and **Chapter 3. Light-sleep**.
- Deep-sleep mode is controlled by users. Users can call the function to enable Deep-sleep instantly. For details, please refer to **Chapter 4. Deep-sleep**.
- RTC (Real-Time Clock).
- DTIM (Delivery Traffic Indication Message).

Fonte: ESPRESSIF, 2016.

Como mostra a figura, o modo a ser utilizado dependerá das características do projeto, como é o caso do modo deep-sleep, que apesar de reduzir o consumo de corrente a um valor insignificante, impossibilitará a utilização das funcionalidades *Wi-Fi* e da CPU (*Central Process Unit* – Unidade Central de Processamento), mantendo ligado apenas o RTC (*Real Time Clock*), que será responsável por reativar o SoC através de algum evento externo (PEDROMINATEL, 2016).

Além disso é possível, a depender da capacidade da memória *flash* do módulo, realizar a atualização dos códigos embarcados via *Wi-Fi*, recurso esse conhecido como atualização OTA (*Over The Air*), bastando a utilização de uma biblioteca e acrescentar algumas linhas a mais no algoritmo a ser usado. Este recurso se mostra muito útil quando o dispositivo contendo o SoC se encontra em locais perigosos ou de difícil acesso (PEDROMINATEL, 2016).

3 CONCLUSÕES

O presente trabalho se propôs a desenvolver um pequeno sistema de monitoramento de consumo de água, a nível de protótipo, com o uso de uma placa de prototipagem e um sensor de fluxo de água de baixo custo, associados a uma plataforma web que permitisse visualizar os dados adquiridos. As etapas do projeto consistiram em implementar um protótipo, realizando a montagem eletrônica para testes, desenvolver o algoritmo a ser embarcado no SoC e utilizar uma interface web para apresentar os dados coletados. Todas as etapas foram devidamente implementadas e o protótipo apresentou resultados satisfatórios.

Na primeira das etapas, uma vez definido o projeto, a implementação foi efetuada sem muitos problemas, pois apesar de utilizada uma técnica simples para adequação do nível de tensão, esta se mostrou eficaz e os resultados pretendidos foram atingidos. Porém a alma do processo é o algoritmo, que ainda deveria ser desenvolvido.

O desenvolvimento do código, realizado na segunda etapa, não se mostrou uma tarefa fácil, contudo após inúmeros testes, modificações e várias pesquisas, foi-se obtido o efeito desejado, ou seja, conseguir adquirir os dados, processá-los e transformá-los em informação, para que pudessem ser apresentadas ao usuário através da porta serial.

A última etapa consistia em tornar as informações acessíveis a qualquer momento e em qualquer lugar, objetivo último do projeto. Esta meta foi alcançada através da utilização da plataforma ThingSpeak, disponível tanto para dispositivos fixos (*desktops/notebooks*) quanto para dispositivos móveis (*smartphones*), e que permite a criação de contas de acesso gratuito. A plataforma oferece uma interface gráfica com diversas ferramentas e, o mais interessante, conta com a possibilidade de utilizar o MATLAB e seus inúmeros recursos no processamento dos dados recebidos pelo ThingSpeak.

A elaboração deste trabalho teve um valor muito especial, uma vez que o projeto desenvolvido tinha por objetivo motivar as pessoas a utilizarem a água de uma forma mais consciente e racional, para assegurar que tenham acesso a este recurso por um longo tempo. Além disso, o projeto foi uma forma de retribuir à

comunidade todo o conhecimento obtido, de forma gratuita, ao longo do processo de graduação nesta instituição de ensino superior de natureza pública e que, portanto, só existe graças à contribuição de toda a sociedade brasileira.

Por fim, mesmo com todos os obstáculos, vale ressaltar os inúmeros ganhos, profissionais e pessoais, que a elaboração e implementação deste trabalho possibilitaram, além da satisfação proporcionada ao ver o projeto funcionando conforme o esperado.

4 REFERÊNCIAS

AI-THINKER. **ESP-12E WiFi Module Version 1.0**. KLOPPENBORG, 2015. Disponível em: <<https://www.kloppenborg.net/images/blog/esp8266/esp8266-esp12e-specs.pdf>>. Acesso em: 10 jan. 2017.

ARDUINO. **attachInterrupt()**. ARDUINO, 2018. Disponível em: <<https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/>>. Acesso em: 13 jan. 2018.

BERGER, A. **Simplifique o desenvolvimento de soluções de IoT com arquiteturas de IoT**. IBM, 2017. Disponível em: <<https://www.ibm.com/developerworks/br/library/iot-lp201-iot-architectures/index.html>>. Acesso em: 03 jun. 2018.

BERTOLETI, P. H. F. **Water On IoT**. HACKSTER, 2016. Disponível em: <https://www.hackster.io/phfbertoleti/water-on-iot-84fb94?ref=challenge&ref_id=35&offset=14>. Acesso em: 11 dez. 2017.

BOYLESTAD, R. L. **Introdução à análise de circuitos**. Tradução de Daniel Vieira e Jorge Ritter. 12º ed. São Paulo: Pearson Prentice Hall, 2012.

BRASIL. BANCO NACIONAL DE DESENVOLVIMENTO ECONÔMICO E SOCIAL; MINISTÉRIO DO PLANEJAMENTO, DESENVOLVIMENTO E GESTÃO; MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES. **Cartilha de Cidades**. BNDES, 2018. Disponível em: <<https://www.bndes.gov.br/wps/wcm/connect/site/db27849e-dd37-4fbd-9046-6fda14b53ad0/produto-13-cartilha-das-cidades-publicada.pdf?MOD=AJPERES&CVID=m7tz8bf>>. Acesso em: 12 maio 2018.

ESPRESSIF. **ESP8266EX Datasheet Version 5.8**. ESPRESSIF, 2018. Disponível em: <https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf>. Acesso em: 07 fev. 2017.

INTERNACIONAL BUSINESS MACHINES CORPORATION; EUROTECH. **MQTT V3.1 Protocol Specification**. IBM, 2010. Disponível em: <<http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html>>. Acesso em: 07 mar. 2017.

MICROSOFT. **Microsoft Azure IoT Reference Architecture**. Estados Unidos, 2018, 3p.

MINATEL, P. **Modos de economia de energia no ESP8266**. PEDROMINATEL, 2016. Disponível em: <<http://pedrominate.com.br/pt/esp8266/modos-de-economia-de-energia-no-esp8266/>>. Acesso em: 03 fev. 2017.

MINATEL, P. **OTA – Como programar o ESP8266 pelo WiFi no platformIO**. PEDROMINATEL, 2016. Disponível em: <<http://pedrominate.com.br/pt/esp8266/ota-como-programar-o-esp8266-pelo-wifi-no-platformio/>>. Acesso em: 26 jan. 2017.

MINATEL, P. **Utilizando a “EEPROM” do ESP8266**. PEDROMINATEL, 2016. Disponível em: <<http://pedrominate.com.br/pt/esp8266/utilizando-EEPROM-do-esp8266/>>. Acesso em: 18 out. 2016.

MINATEL, P. **Webserver log no ESP8266 com SPIFFS**. PEDROMINATEL, 2017. Disponível em: <<http://pedrominate.com.br/pt/esp8266/webserver-log-no-esp8266-com-spiiffs/>>. Acesso em: 02 jul. 2017.

MQTT.ORG. **Frequently Asked Questions**. MQTT.ORG, 2014. Disponível em: <<http://mqtt.org/faq>>. Acesso em: 03 mar. 2017.

NODE MCU TEAM. **NODE MCU DEVKIT V1.0**. GITHUB, 2015. Disponível em: <https://github.com/nodemcu/nodemcu-devkit-v1.0/blob/master/NODEMCU_DEVKIT_V1.0.PDF>. Acesso em: 19 jun. 2016.

O’LEARY, N. **Arduino Client for MQTT**. GITHUB, 2018. Disponível em: <<https://github.com/knolleary/pubsubclient>>. Acesso em: 20 jan. 2018.

OASIS. **MQTT Version 3.1.1 Plus Errata 01**. OASIS, 2015. Disponível em: <<http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>>. Acesso em: 05 mar. 2017.

SACCO, F. **Níveis e limites de tensões digitais**. EMBARCADOS, 2014. Disponível em: <<https://www.embarcados.com.br/niveis-e-limites-de-tensoes-digitais/>>. Acesso em: 04 abr. 2018.

SOUZA, F. **Arduino UNO**. EMBARCADOS, 2013. Disponível em: <<https://www.embarcados.com.br/arduino-uno/>>. Acesso em: 27 set. 2017.

UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION. **Water for People Water for Life**. Estados Unidos, 2003, 68p.

VASCONCELLOS, M. A. S.; GARCIA, M. E. **Fundamentos de Economia**. 5º ed. São Paulo: Saraiva, 2014.

WINBOND. **8M-BIT, 16M-BIT AND 32M-BIT SERIAL FLASH MEMORY WITH DUAL AND QUAD SPI**. China, 2007b, 5p.

YF-S201 Hall Effect Water Flow Meter / Sensor. HOBBYTRONICS. Disponível em: <<http://www.hobbytronics.co.uk/yf-s201-water-flow-meter>>. Acesso em: 20 mar. 2018.

YIFA the plastics Ltd Product Introduction. HOBBYTRONICS. Disponível em: <<http://www.hobbytronics.co.uk/datasheets/sensors/YF-S201.pdf>>. Acesso em: 27 mar. 2018.

YUAN, M. **Conhecendo o MQTT**. IBM, 2017. Disponível em: <<https://www.ibm.com/developerworks/br/library/iot-mqtt-why-good-for-iot/index.html>>. Acesso em: 10 fev. 2018.

APÊNDICE A – ALGORITMO COMPLETO

Algoritmo baseado nos exemplos e discussões do repositório da biblioteca PubSubClient, na documentação referente à programação com a plataforma Arduino, nos exemplos contidos no site MathWorks sobre a utilização do ThingSpeak + MQTT e no projeto Water On IoT desenvolvido por Pedro Bertoleti, disponível em:

https://www.hackster.io/phfbertoleti/water-on-iot-84fb94?ref=challenge&ref_id=35&offset=14

```

/***** Filename: config.h *****/
// Definição de pinos/GPIOs
#define D2 4

// Parâmetros da conexão Wi-Fi
const char* wifiSSID = "My_NetworkName"; // Nome da rede Wi-Fi na qual se deseja
conectar
const char* wifiPassword = "My_WifiPassword"; // Senha da rede Wi-Fi na qual se
deseja se conectar

// Parâmetros da conexão com o Broker MQTT
const char* mqttID = "My_MQTT_ID"; // ID para identificação da sessão de conexão
ao broker MQTT; deve ser única
const char* mqttUserName = "My_MQTT_UserName"; // Nome de usuário para se conectar
ao broker ThingSpeak
const char* mqttAPIKey = "MY_MQTT_API_Key"; // Senha para se conectar ao broker
ThingSpeak
const char* writeAPIKey = "My_Write_API_Key"; // chave de escrita no canal
const char* readAPIKey = "My_Read_API_Key"; // Chave de leitura do canal
const char* channelID = "My_Channel_ID"; // ID de identificação do canal na
plataforma ThingSpeak
const char* mqttServer = "mqtt.thingspeak.com"; // URL do broker MQTT que se
deseja utilizar
uint16_t mqttPort = 1883; // Porta do Broker MQTT

// Tópico de leitura
const char* topicRead = "channels/channelID/subscribe/fields/field1/readAPIKey";

```

```

/***** Filename: main.cpp *****/
#include <ESP8266WiFi.h> // Biblioteca que possibilita utilizar o Wi-Fi
#include <PubSubClient.h> // Biblioteca que permite utilizar o protocolo MQTT

#include "config.h" // Biblioteca com os parâmetros de configuração

// Inicialização das bibliotecas WiFiClient e PubSubClient
WiFiClient nodeMCUClient; // Construtor de classe "WiFiClient" utilizado para
    criar a instância "nodeMCUClient"
PubSubClient mqttClient(nodeMCUClient); // Construtor de classe "PubSubClient",
    utilizado para criar a instância "mqttClient", que recebe o objeto "nodeMCUClient"

// Variáveis globais
long interruptionsCounter;
unsigned long lastMillis;
unsigned long lastSendTime;
const unsigned long interval = 20000;
float flowRate;
float consumption;
unsigned int flag = 0;

// Protótipos das funções
void setWiFiConnection(void);
void setMQTTConnection(void);
void mqttCallback(char* topic, byte* payload, unsigned int length);
void checkWiFiAndMQTTconnection(void);

void pulsesCounter(void);
void resetPulsesCounter(void);
void calculateFlowRateAndConsumption(void);
void sendFlowRateAndConsumptionByMQTT(void);
void sendFlowRateAndConsumptionToSerial(void);
void turnOnPulsesCounter(void);
void turnOffPulsesCounter(void);

// Estabelece a conexão Wi-Fi
void setWiFiConnection(void)
{
    Serial.println();

```

```

Serial.println();
Serial.println("-----Conexão Wi-Fi-----");
Serial.println();
Serial.print("Conectando-se na rede ");
Serial.println(wifiSSID);
Serial.print("Aguarde");

if (WiFi.status() == WL_CONNECTED) {
    return;
}

WiFi.begin(wifiSSID, wifiPassword);

while (WiFi.status() != WL_CONNECTED) {
    delay(100);
    Serial.print(".");
}

Serial.println();
Serial.print("Conectado com sucesso na rede ");
Serial.print(wifiSSID);
Serial.println();
Serial.print("IP obtido: ");
Serial.print(WiFi.localIP());
Serial.println();
}

// Estabelece a conexão com o Broker MQTT
void setMQTTConnection()
{
    mqttClient.setServer(mqttServer,mqttPort);
    mqttClient.setCallback(mqttCallback);

    while (!mqttClient.connected()) {
        Serial.println();
        Serial.println("-----Conexão Broker MQTT-----");
        Serial.println();
        Serial.print("Tentando se conectar ao Broker MQTT ");
        Serial.println(mqttServer);
    }
}

```

```

    if (mqttClient.connect(mqttID,mqttUserName,mqttAPIKey)) {
        Serial.println("Conectado com sucesso ao broker MQTT!");

        mqttClient.subscribe(topicRead); // realiza a assinatura do tópic
"topicRead"
    }
    else {
        Serial.println("Falha ao reconectar no broker!");
        Serial.println("Causa: ");
        Serial.print(mqttClient.state());
        Serial.println();
        Serial.println("Haverá nova tentativa de conexão em 5s");
        delay(5000);
    }
}

// Função chamada quando há publicações no tópico ao qual se está assinando
void mqttCallback(char* topic, byte* payload, unsigned int length)
{
    payload[length] = '\0';
    String s = String((char*)payload);
    float salveConsumption = s.toFloat();
    consumption = consumption + salveConsumption;

    Serial.println();
    Serial.print("Mensagem da função callback: ");
    Serial.println(salveConsumption);
    flag = 1;
}

// Verifica as conexões Wi-Fi e MQTT
void checkWiFiAndMQTTconnection(void)
{
    if (WiFi.status() != WL_CONNECTED) {
        setWiFiConnection();
    }
    if (!mqttClient.connected()) {
        setMQTTConnection();
    }
}

```

```

    }
}

```

```

// Conta a quantidade de pulsos

```

```

void pulsesCounter(void)
{
    *(volatile long*)&interruptionsCounter += 1;
}

```

```

// Reseta a variável "interruptionsCounter"

```

```

void resetPulsesCounter(void)
{
    interruptionsCounter = 0;
}

```

```

// Calcula a vazão e o consumo de água

```

```

void calculateFlowRateAndConsumption(void)
{
    flowRate = (float) ((float)interruptionsCounter / (float)450.0);
    consumption = consumption + flowRate;
    flowRate = flowRate*60.0;
}

```

```

// Envia a vazão e o consumo de água através do protocolo MQTT

```

```

void sendFlowRateAndConsumptionByMQTT(void)
{
    String data = String("field1=" + String(consumption));
    data = String(data + String("&field2=" + String(flowRate)));
    int length = data.length();
    char msgBuffer[length];
    data.toCharArray(msgBuffer, length+1);

    String topic = String("channels/" + String(channelID));
    topic = String(topic + String("/publish/" + String(writeAPIKey)));
    length = topic.length();
    char topicBuffer[length];
}

```

```

    topic.toCharArray(topicBuffer, length+1);

    mqttClient.publish(topicBuffer, msgBuffer); // Publica a vazão e o consumo no
    tópico "topicBuffer"

    lastSendTime = millis();
}
// Publica a vazão e o consumo na porta serial
void sendFlowRateAndConsumptionToSerial(void)
{
    Serial.println();
    Serial.print("Vazão: ");
    Serial.print(flowRate);
    Serial.println(" l/min");
    Serial.print("Consumo: ");
    Serial.print(consumption);
    Serial.println(" litros");
}

// Habilita as interrupções
void turnOnPulsesCounter(void)
{
    attachInterrupt(digitalPinToInterrupt(D2), pulsesCounter, RISING); // Habilita
    as interrupções associadas ao pino D2, sendo que a função "pulseCounter" deverá
    ser chamada sempre que houver um evento de borda de subida no pino D2
}

// Desabilita as interrupções
void turnOffPulsesCounter(void)
{
    detachInterrupt(digitalPinToInterrupt(D2)); // Desabilita as interrupções
    associadas ao pino D2
}

void setup()
{
    Serial.begin(115200); // Inicia a serial com a taxa de 115200 baud rate
    delay(10);
}

```

```

WiFi.mode(WIFI_STA); // Configura o ESP como STATION, ou seja, cliente
setWiFiConnection();
setMQTTConnection();

resetPulsesCounter();
flowRate = 0.0;
consumption = 0.0;

pinMode(D2, INPUT); // Configura o pino D2 da NodeMCU como entrada

turnOnPulsesCounter();
lastMillis = millis();
lastSendTime = lastMillis;
}

void loop()
{
    checkWiFiAndMQTTconnection(); // Verifica as conexões Wi-Fi e MQTT

    if ((millis() - lastMillis) >= 1000) { // Base de tempo para realização dos
        cálculos de vazão e consumo (1 segundo)
        if (flag == 1) {
            mqttClient.unsubscribe(topicRead); // Cancela a assinatura ao tópico
            "topicRead"
        }

        turnOffPulsesCounter(); // Desabilita as interrupções

        calculateFlowRateAndConsumption(); // Calcula a vazão e o fluxo

        if ((millis() - lastSendTime) >= interval) { // Publica a vazão e o
            consumo a intervalos pré-determinados (neste caso, a cada 20 segundos)
            sendFlowRateAndConsumptionByMQTT();
        }

        sendFlowRateAndConsumptionToSerial(); // Envia a vazão e o consumo para a
        porta serial
    }
}

```

```
    lastMillis = millis(); // Atualiza o último valor que havia sido  
armazenado na variável "lastMillis"
```

```
    resetPulsesCounter(); // Reseta a variável que armazena o número de  
interrupções
```

```
    turnOnPulsesCounter(); // Habilita as interrupções  
}
```

```
    mqttClient.loop(); // Garante a manutenção da conexão ao broker e o  
recebimento de publicações  
}
```

APÊNDICE B – CÓDIGO UTILIZADO NO MATLAB

Algoritmo baseado nos exemplos contidos no site MathWorks sobre a utilização do ThingSpeak +

MATLAB, disponível em:

<https://www.mathworks.com/help/thingspeak/thingspeakread.html>

```

/***** MATLAB Code: Histogram of Flow Rate Variation *****/

% Read flow rate for the last 30 minutes from a ThingSpeak channel and
% visualize flow rate variations using the MATLAB HISTOGRAM function.

% Channel My_Channel_ID contains data of Monitoring of Water Consumption,
% located in Araguari, Brazil. The data is collected three times per minute.
% Field 2 contains flow rate data.
% Channel ID to read data from
readChannelID = My_Channel_ID;

% Flow Rate Field ID
flowRateFieldID = 2;

% Channel Read API Key
% If your channel is private, then enter the read API
% Key between the '' below:
readAPIKey = 'My_Read_API_Key';
% Get flow rate data from field 2 for the last 30 minutes.
% Learn more about the THINGSPEAKREAD function by going to
% the Documentation tab on the right side pane of this page.

flowRateF = thingSpeakRead(readChannelID,'Fields',flowRateFieldID,...
'NumMinutes',30, 'ReadKey',readAPIKey);

histogram(flowRateF);
xlabel('Flow Rate (litres/minute)');
ylabel('Number of Measurements\nnewline for Each Flow Rate');
title('Histogram of Flow Rate Variation');

```

ANEXO I – DADOS RELEVANTES DO DATASHEET DO ESP8266EX

Obtido da página do fabricante do SoC (Espressif),

https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

Figure 2-1 shows the pin layout for 32-pin QFN package.

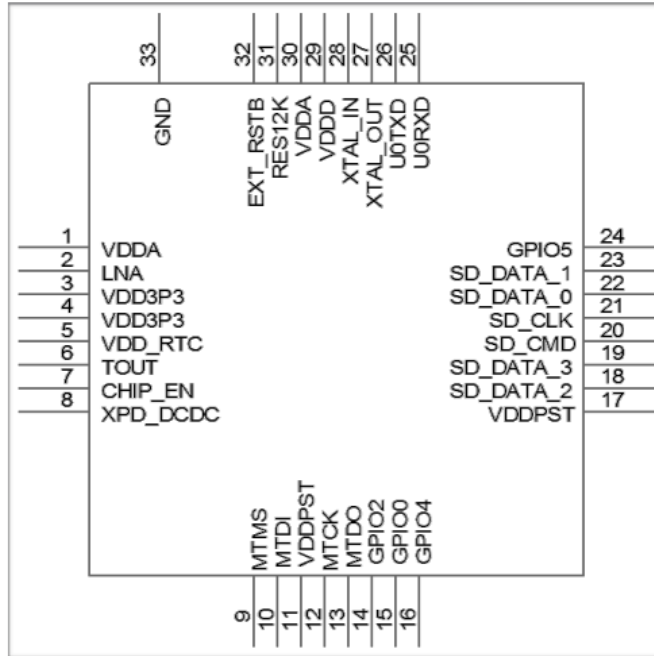


Figure 2-1. Pin Layout

Table 2-1 lists the definitions and functions of each pin.

Table 2-1. ESP8266EX Pin Definitions

Pin	Name	Type	Function
1	VDDA	P	Analog Power 2.5V ~ 3.6V
2	LNA	I/O	RF antenna interface Chip output impedance=39+j6 Ω. It is suggested to retain the π-type matching network to match the antenna.
3	VDD3P3	P	Amplifier Power 2.5V ~ 3.6V
4	VDD3P3	P	Amplifier Power 2.5V ~ 3.6V
5	VDD_RTC	P	NC (1.1V)
6	TOUT	I	ADC pin. It can be used to test the power-supply voltage of VDD3P3 (Pin3 and Pin4) and the input power voltage of TOUT (Pin 6). However, these two functions cannot be used simultaneously.
7	CHIP_PU	I	Chip Enable High: On, chip works properly Low: Off, small current consumed

Pin	Name	Type	Function
8	XPD_DCDC	I/O	Deep-sleep wakeup (need to be connected to EXT_RSTB); GPIO16
9	MTMS	I/O	GPIO 14; HSPI_CLK
10	MTDI	I/O	GPIO 12; HSPI_MISO
11	VDDPST	P	Digital/IO Power Supply (1.8V ~ 3.6V)
12	MTCK	I/O	GPIO 13; HSPI_MOSI; UART0_CTS
13	MTDO	I/O	GPIO 15; HSPI_CS; UART0_RTS
14	GPIO2	I/O	UART Tx during flash programming; GPIO2
15	GPIO0	I/O	GPIO0; SPI_CS2
16	GPIO4	I/O	GPIO4
17	VDDPST	P	Digital/IO Power Supply (1.8V ~ 3.6V)
18	SDIO_DATA_2	I/O	Connect to SD_D2 (Series R: 200Ω); SPIHD; HSPIHD; GPIO9
19	SDIO_DATA_3	I/O	Connect to SD_D3 (Series R: 200Ω); SPIWP; HSPIWP; GPIO10
20	SDIO_CMD	I/O	Connect to SD_CMD (Series R: 200Ω); SPI_CS0; GPIO11
21	SDIO_CLK	I/O	Connect to SD_CLK (Series R: 200Ω); SPI_CLK; GPIO6
22	SDIO_DATA_0	I/O	Connect to SD_D0 (Series R: 200Ω); SPI_MISO; GPIO7
23	SDIO_DATA_1	I/O	Connect to SD_D1 (Series R: 200Ω); SPI_MOSI; GPIO8
24	GPIO5	I/O	GPIO5
25	U0RXD	I/O	UART Rx during flash programming; GPIO3
26	U0TXD	I/O	UART Tx during flash programming; GPIO1; SPI_CS1
27	XTAL_OUT	I/O	Connect to crystal oscillator output, can be used to provide BT clock input
28	XTAL_IN	I/O	Connect to crystal oscillator input
29	VDDD	P	Analog Power 2.5V ~ 3.6V
30	VDDA	P	Analog Power 2.5V ~ 3.6V
31	RES12K	I	Serial connection with a 12 kΩ resistor and connect to the ground
32	EXT_RSTB	I	External reset signal (Low voltage level: active)

Table 5-1. Electrical Characteristics

Parameters	Conditions	Min	Typical	Max	Unit
Storage Temperature Range	-	-40	Normal	125	°C
Maximum Soldering Temperature	IPC/JEDEC J-STD-020	-	-	260	°C
Working Voltage Value	-	2.5	3.3	3.6	V
I/O	V_{IL}/V_{IH}	-0.3/0.75 V_{IO}	-	0.25 V_{IO} / 3.6	V
	V_{OL}/V_{OH}	N/0.8 V_{IO}	-	0.1 V_{IO} /N	
	I_{MAX}	-	-	12	mA
Electrostatic Discharge (HBM)	TAMB=25°C	-	-	2	KV
Electrostatic Discharge (CDM)	TAMB=25°C	-	-	0.5	KV

Table 5-2. Power Consumption

Parameters	Min	Typical	Max	Unit
Tx802.11b, CCK 11Mbps, $P_{OUT}=+17$ dBm	-	170	-	mA
Tx 802.11g, OFDM 54Mbps, $P_{OUT}=+15$ dBm	-	140	-	mA
Tx 802.11n, MCS7, $P_{OUT}=+13$ dBm	-	120	-	mA
Rx 802.11b, 1024 bytes packet length, -80 dBm	-	50	-	mA
Rx 802.11g, 1024 bytes packet length, -70 dBm	-	56	-	mA
Rx 802.11n, 1024 bytes packet length, -65 dBm	-	56	-	mA
Modem-sleep ^①	-	15	-	mA
Light-sleep ^②	-	0.9	-	mA
Deep-sleep ^③	-	20	-	μA
Power Off	-	0.5	-	μA

 Notes:

- ① Modem-sleep mode is used in the applications that require the CPU to be working, as in PWM or I2S applications. According to 802.11 standards (like U-APSD), it shuts down the Wi-Fi Modem circuit while maintaining a Wi-Fi connection with no data transmission to optimize power consumption. E.g. in DTIM3, maintaining a sleep of 300 ms with a wakeup of 3 ms cycle to receive AP's Beacon packages at interval requires about 15 mA current.
- ② During Light-sleep mode, the CPU may be suspended in applications like Wi-Fi switch. Without data transmission, the Wi-Fi Modem circuit can be turned off and CPU suspended to save power consumption according to the 802.11 standards (U-APSD). E.g. in DTIM3, maintaining a sleep of 300 ms with a wakeup of 3ms to receive AP's Beacon packages at interval requires about 0.9 mA current.
- ③ During Deep-sleep mode, Wi-Fi is turned off. For applications with long time lags between data transmission, e.g. a temperature sensor that detects the temperature every 100s, sleeps for 300s and wakes up to connect to the AP (taking about 0.3 ~ 1s), the overall average current is less than 1mA. The current of 20 μA is acquired at the voltage of 2.5V.

3.1.2. Memory

ESP8266EX Wi-Fi SoC integrates memory controller and memory units including SRAM and ROM. MCU can access the memory units through iBus, dBus, and AHB interfaces. All memory units can be accessed upon request, while a memory arbiter will decide the running sequence according to the time when these requests are received by the processor.

According to our current version of SDK, SRAM space available to users is assigned as below.

- RAM size < 50 kB, that is, when ESP8266EX is working under the Station mode and connects to the router, programmable space accessible in heap + data section is around 50 kB.
- There is no programmable ROM in the SoC, therefore, user program must be stored in an external SPI flash.

External Flash

ESP8266EX uses external SPI flash to store user programs, and supports up to 16 MB memory capacity theoretically.

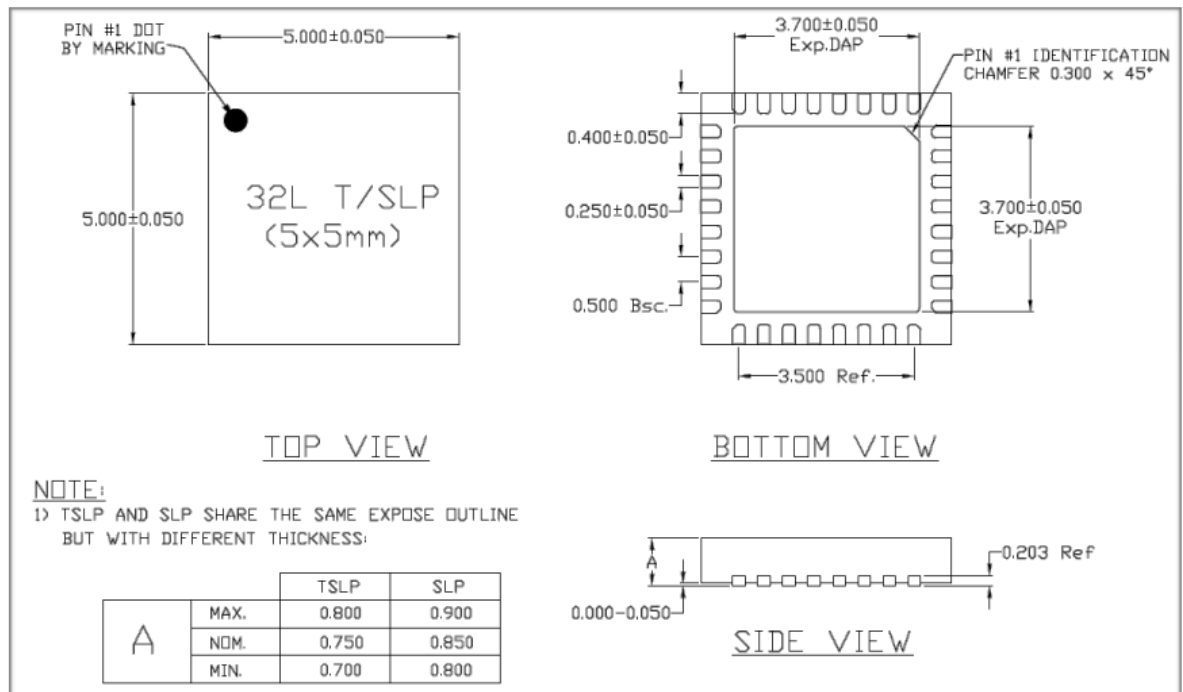


Figure 6-1. ESP8266EX Package

ANEXO II – DADOS RELEVANTES DO DATASHEET DO MÓDULO ESP-12E

Obtido de sítio de terceiros (kloppenborg),

<https://www.kloppenborg.net/images/blog/esp8266/esp8266-esp12e-specs.pdf>

Table 2 ESP-12E Pin design

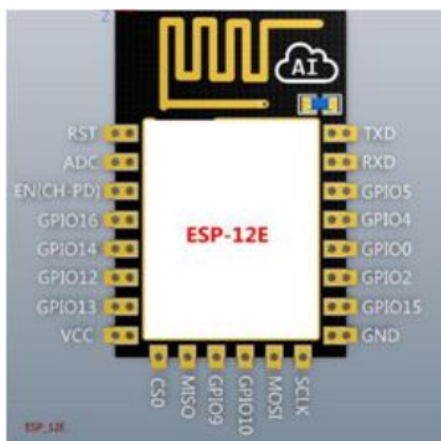


Table 3 Pin Descriptions

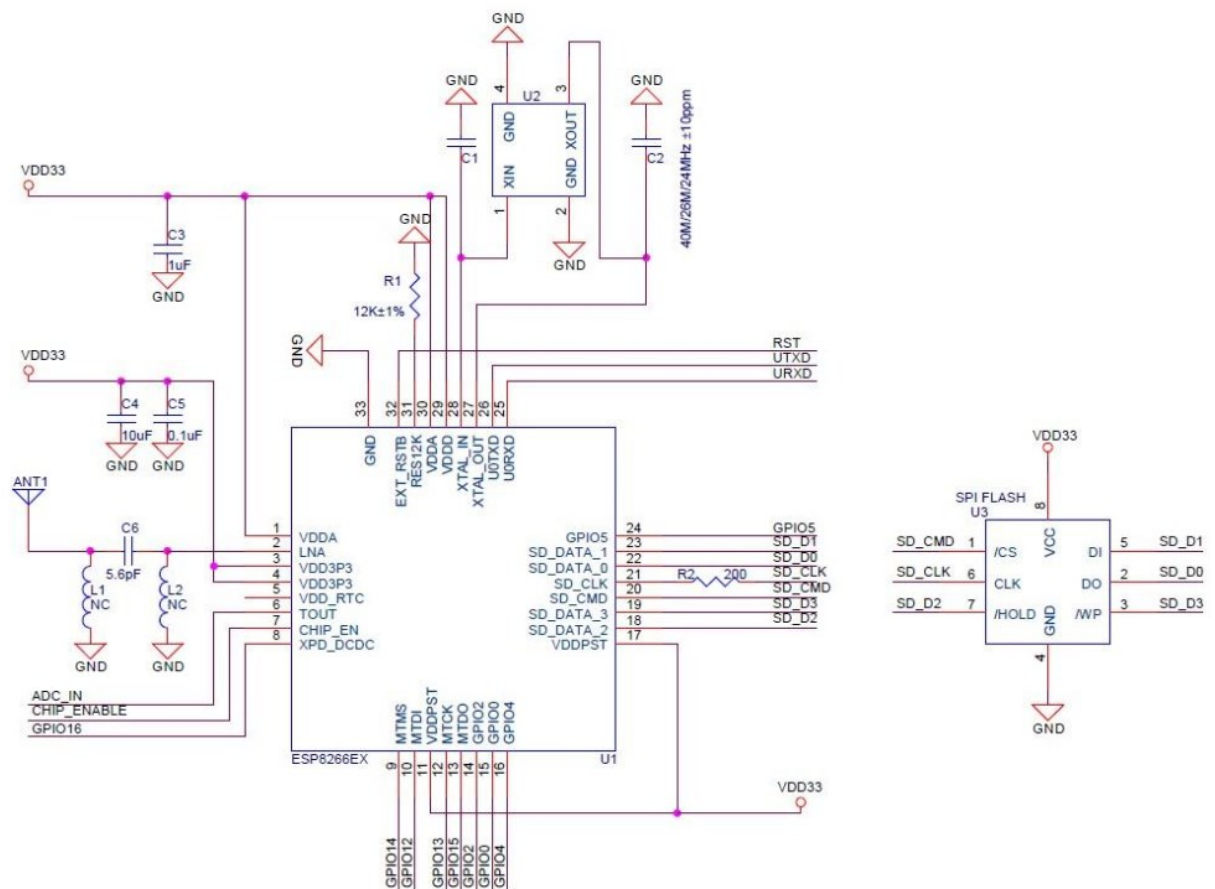
NO.	Pin Name	Function
1	RST	Reset the module
2	ADC	A/D Conversion result.Input voltage range 0-1v,scope:0-1024
3	EN	Chip enable pin.Active high
4	IO16	GPIO16; can be used to wake up the chipset from deep sleep mode.
5	IO14	GPIO14; HSPI_CLK
6	IO12	GPIO12; HSPI_MISO
7	IO13	GPIO13; HSPI_MOSI; UART0_CTS
8	VCC	3.3V power supply (VDD)
9	CS0	Chip selection
10	MISO	Salve output Main input
11	IO9	GPIO9
12	IO10	GPIO10
13	MOSI	Main output slave input
14	SCLK	Clock
15	GND	GND
16	IO15	GPIO15; MTDO; HSPICS; UART0_RTS
17	IO2	GPIO2; UART1_TXD
18	IO0	GPIO0
19	IO4	GPIO4
20	IO5	GPIO5
21	RXD	UART0_RXD; GPIO3
22	TXD	UART0_TXD; GPIO1

Table 4 Pin Mode

Mode	GPIO15	GPIO0	GPIO2
UART	Low	Low	High
Flash Boot	Low	High	High

Table 8 Recommended Operating Conditions

Operating Condition	Symbol	Min	Typ	Max	Unit
Operating Temperature		-40	20	125	°C
Supply voltage	VDD	3.0	3.3	3.6	V

Figure 4 Schematics of Esp-12E WiFi Module

POWER

U4: MCP1117SF33TG

VIN: 100µF 25V

VOUT: 100µF

D1: IN5819/SS14/S4(SOD_323)

Working Output: 3.3V 800mA
Working Current Limit: 1000mA
Max Current: 1000mA
Max Supply Voltage: 20V
Voltage Dropout: 1.2V@800mA

VER	DATE
1.0	28/01/2015

ORGANIZATION
NODE MCU TEAM
WEBSITE
WWW.NODEMCU.COM

IO CONN

[illegible]

THT_Male_P_1x15 THT_Male_P_1x15



VER	DATE
0.9	20/11/2014
ORGANIZATION	
NODE MCU TEAM	
WEBSITE	
WWW.NODEMCU.COM	

ADC

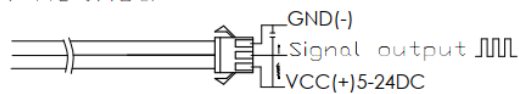
VER	DATE
0.9	20/11/2014
ORGANIZATION	
NODE MCU TEAM	
WEBSITE	
WWW.NODEMCU.COM	

ANEXO IV – DATASHEET DO SENSOR DE FLUXO YF-S201B ½”

Obtido da página do vendedor do sensor,
https://img.filipeflop.com/files/download/Datasheet_YF-S201.pdf

YIFA the plastics Ltd Prodcut Introduction

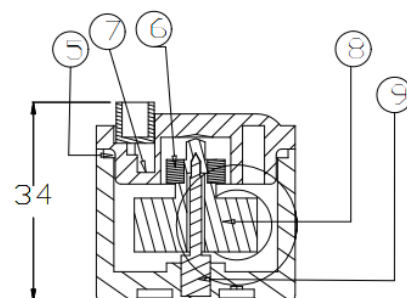
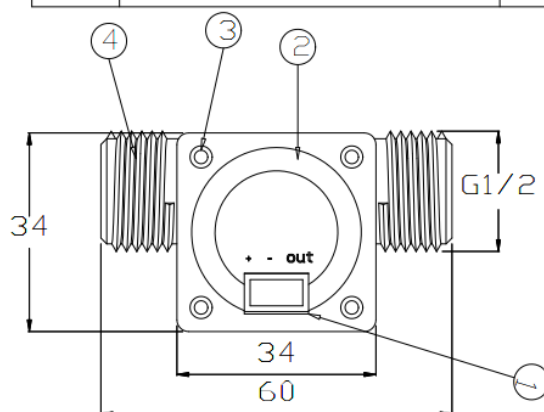
- 1.Modle:YF 21
- 2.Product Name:Hall sensor
- 3.Flow Range: 1-30L/MIN
- 4.(1)Connection Method



- (2)Voltage Range 3.5-24VDC, Pluse Characteristic:F=7Q(L/MIN).
- (3)Extent of error:±5%.
- (4)Flow-Pulse
 2L/MIN=16HZ 4L/MIN=32.5HZ 6L/MIN=49.3HZ
 8L/MIN=65.5HZ 10L/MIN=82HZ

5.Bom

No.	Item	Material	Qty.
1	Connection wire		1
2	Bonnet	PA	1
3	Screw		4
4	Valve body	PA	1
5	Leak press valve		1
6	Magnet		1
7	Hall		1
8	Impeller	POM	1
9	Rustless steel axis	SUS304	1



Flow-Pulse	Flow-Curve	Connection method																																																																																																																																																																
→.Flow Range:100L/H-/1800H-L/H <table border="1"><thead><tr><th>Flow (L/H)</th><th>Frezq.(HZ)</th><th>Erro range</th></tr></thead><tbody><tr><td>120L/H</td><td>16</td><td rowspan="6">±10</td></tr><tr><td>240L/H</td><td>32.5</td></tr><tr><td>360L/H</td><td>49.3</td></tr><tr><td>480L/H</td><td>65.5</td></tr><tr><td>600L/H</td><td>82</td></tr><tr><td>720L/H</td><td>90.2</td></tr></tbody></table> L/H <table><tr><td>840</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>720</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>600</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>480</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>360</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>240</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>120</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> 16 32.5 49.5 65.5 82 90.2 Freq.(HZ) Connection method: Black Yellow Red GND(-) Signal output VCC(+) 5-24DC			Flow (L/H)	Frezq.(HZ)	Erro range	120L/H	16	±10	240L/H	32.5	360L/H	49.3	480L/H	65.5	600L/H	82	720L/H	90.2	840																		720																		600																		480																		360																		240																		120																		0																	
Flow (L/H)	Frezq.(HZ)	Erro range																																																																																																																																																																
120L/H	16	±10																																																																																																																																																																
240L/H	32.5																																																																																																																																																																	
360L/H	49.3																																																																																																																																																																	
480L/H	65.5																																																																																																																																																																	
600L/H	82																																																																																																																																																																	
720L/H	90.2																																																																																																																																																																	
840																																																																																																																																																																		
720																																																																																																																																																																		
600																																																																																																																																																																		
480																																																																																																																																																																		
360																																																																																																																																																																		
240																																																																																																																																																																		
120																																																																																																																																																																		
0																																																																																																																																																																		