SISTEMA INTELIGENTE DE MONITORAMENTO E ALERTA DE ENCHENTES BASEADO EM INTERNET DAS COISAS (IoT)

Alexandre Magnus Fernandes Guimarães[1], Artur Gabriel Rodrigues Queiroz[1], Carlos José dos Santos Neto[1], Gustavo Costa Thiago[1], João Victor Freire dos Santos[1], Júlio dos Anjos Lucas Moura[1], Letícia Rayane Pereira da Silva[1]

[1]Universidade Federal do Rio Grande do Norte (UFRN), Centro de Tecnologia, Natal–RN, Brasil (gustavo.thiago.122@ufrn.edu.br)

Resumo: As enchentes urbanas provocam prejuízos materiais e riscos à vida nas cidades brasileiras. Este trabalho apresenta um sistema inteligente de monitoramento e alerta de enchentes baseado em Internet das Coisas, com o ESP32 e sensores de nível, chuva e vazão. Os dados são processados localmente, gravados em cartão SD e convertidos em alertas sonoros e notificações via Telegram. Os testes confirmaram uma solução de baixo custo e eficaz para a prevenção de desastres, alinhada à ODS 6.

Palavras-chave: enchentes urbanas; Internet das Coisas; monitoramento ambiental; ESP32; ODS 6

INTRODUÇÃO

As enchentes urbanas estão entre os desastres mais recorrentes no Brasil e atingem, todos os anos, milhares de pessoas. Além das perdas materiais, esses eventos comprometem a infraestrutura das cidades e colocam vidas em risco. O crescimento desordenado dos centros urbanos, somado às mudanças climáticas e à fragilidade dos sistemas de drenagem, tem tornado os alagamentos cada vez mais frequentes e intensos.

Nesse cenário, a Internet das Coisas (IoT) tem se mostrado uma aliada importante no monitoramento ambiental. Ao conectar sensores a plataformas digitais, ela permite coletar e transmitir dados em tempo real, o que favorece tanto o acompanhamento contínuo de áreas críticas quanto a tomada de decisão (Gubbi et al., 2013). Trabalhos recentes reforçam esse potencial: sistemas de baixo custo aplicados ao monitoramento de cheias têm alcançado bons resultados na emissão de alertas preventivos (Santos et al., 2020; Mishra et al., 2021).

A motivação desse trabalho surgiu inicialmente por causa do projeto cobrado na terceira unidade do componente Fundamentos de circuitos e sistemas controlados. Baseado na necessidade, cada vez maior, de soluções para ajudar a minar ou até a sanar as terríveis situações causadas provenientes de enchentes, foi pesquisada a ideia e após ter sido aceita pelo professor, foi iniciado o desenvolvimento do projeto. A proposta também dialoga com a ODS 6 da ONU, voltada à gestão sustentável da água, e gera dados úteis a respeito do planejamento urbano.

Diante disso, o objetivo foi desenvolver um sistema inteligente de monitoramento e alerta de enchentes baseado em IoT, capaz de medir continuamente o nível da água, detectar chuva e acompanhar a vazão, emitindo alertas locais e remotos e mantendo um registro histórico das medições. De forma específica, buscou-se integrar os sensores ao ESP32, implementar a lógica de decisão embarcada, enviar notificações via Telegram, acionar um alerta sonoro por buzzer e armazenar os dados em cartão Micro SD como redundância.

MATERIAL E MÉTODOS

Componentes do sistema

O protótipo foi construído a partir de componentes visando o custo-benefício, escolhidos pela relação entre custo, disponibilidade e robustez. A Tabela 1 reúne os itens utilizados, suas funções e o custo aproximado de cada um, totalizando cerca de R$ 243,54.

Tabela 1. Componentes utilizados no sistema.

Componente	Função principal	Custo aprox.
Placa ESP32 (NodeMCU)	Processamento central, lógica de controle e comunicação Wi-Fi	R$ 40,88
Kit Protoboard + Fonte MB102 + Jumpers M-	Base de montagem e distribuição regulada	R$ 36,90

M	de energia (3,3 V e 5 V)	
Fonte de Alimentação 12V 3A	Energia externa para o módulo MB102	R$ 14,99
Kit Fios Jumper (Macho-Fêmea)	Conexão entre sensores e protoboard	R$ 14,30
Sensor de Vazão (YF-S201)	Medição da taxa de escoamento da água	R$ 31,34
Sensor Ultrassônico (HC-SR04)	Medição de distância para simular o nível da água	R$ 9,25
Sensor de Chuva (YL-83)	Detecção da presença e intensidade de chuva	R$ 15,99
Módulo Leitor Micro SD	Interface para gravação dos logs	R$ 19,00
Cartão Micro SD (16 GB)	Armazenamento local de backup	R$ 16,99
Buzzer Ativo	Atuador sonoro do alerta local	R$ 19,00
Kit de Resistores (1 kΩ, 2,2 kΩ etc.)	Proteção e adequação de níveis de tensão	R$ 24,90
Cabo Micro USB	Gravação do código e comunicação com o PC	Reaproveitado
Custo total aproximado		R$ 243,54

Fonte: Elaborada pelos autores (2026).

O ESP32 (NodeMCU) funciona como o núcleo do sistema: lê os sensores, executa a lógica de controle, aciona o buzzer, dispara as notificações e grava os registros no cartão SD. Sua conectividade Wi-Fi nativa e o bom poder de processamento foram decisivos para a escolha, já que substituem as placas Arduino tradicionais agregando o recurso de IoT, além de sua acessibilidade ser superior se comparada a uma placa de Arduino convencional. (Espressif Systems, 2023).

A montagem foi feita sobre uma protoboard de 830 pontos, sem solda, com a alimentação distribuída pelo módulo MB102 — responsável por converter os 12 V da fonte em linhas estáveis de 5 V e 3,3 V. Esse arranjo alivia o regulador interno do ESP32 e evita reinicializações (brownouts) durante os picos de corrente exigidos pelos sensores e pelo módulo SD.

A medição ambiental ficou a cargo de três sensores. O ultrassônico HC-SR04 estima o nível da água pelo tempo de retorno de um pulso sonoro; o sensor de chuva YL-83 identifica a precipitação pela variação de resistência em sua placa exposta; e o sensor de vazão YF-S201, dotado de uma turbina com sensor de efeito Hall, gera pulsos proporcionais ao fluxo. Em conjunto, eles tornam a leitura das condições mais confiável e reduzem alarmes indevidos, impedindo qualquer tipo se alarme falso, caso algum sensor acuse individualmente.

Como “sirene”, um buzzer ativo de 5 V emite o alerta sonoro local sempre que os limiares críticos são atingidos, dispensando circuitos com relé. O armazenamento do sistema é garantido pelo módulo Micro SD, que registra o histórico em arquivo CSV mesmo quando o Wi-Fi falha. Por fim, resistores de 1 kΩ e 2,2 kΩ protegem as entradas do microcontrolador, por meio de divisores de tensão, condicionando os sinais de 5 V aos 3,3 V suportados pelo ESP32, minando qualquer chance de sobrecarga.

Figura 1. Principais componentes de hardware do protótipo.
Fonte: Elaborada pelos autores (2026).

Funcionamento do sistema

O funcionamento pode ser descrito em cinco etapas encadeadas: coleta, processamento, decisão, emissão de alertas e armazenamento. Na coleta, o ESP32 lê continuamente os três sensores; em seguida, compara os valores obtidos com limites previamente definidos em código para identificar situações de risco.

Uma condição de alerta é reconhecida quando há chuva detectada pelo sensor YL-83 e, ao mesmo tempo, o nível da água se aproxima a menos de 15 cm do sensor ultrassônico; valores elevados de vazão funcionam como indicador complementar de criticidade. Confirmado o risco, o sistema aciona simultaneamente o buzzer e o envio de notificações remotas e grava cada medição no cartão SD. Esse registro local garante que o histórico seja preservado mesmo diante de uma eventual queda de conexão, sendo como se fosse a caixa-preta de uma aeronave.

Projeto elétrico

O circuito foi planejado respeitando os níveis de tensão de cada componente. A alimentação parte de uma fonte externa ligada ao módulo MB102, que disponibiliza as linhas de 5 V e 3,3 V. Os sensores HC-SR04 e YF-S201 e o módulo SD operam em 5 V, enquanto o ESP32 trabalha internamente em 3,3 V.

Dois cuidados foram essenciais. Como o pino ECHO do HC-SR04 devolve cerca de 5 V — acima do limite das entradas do ESP32 —, empregou-se um divisor de tensão com resistores para rebaixar o sinal a um valor seguro. Já a saída em coletor aberto do YF-S201 recebeu um resistor de pull-up de 1 kΩ entre a linha de 3,3 V e o pino de sinal, o que mantém o nível lógico

estável e reduz a influência de ruídos (Guimarães, 2026). A Figura 2 sintetiza a arquitetura de hardware e a Tabela 2 detalha as ligações ao microcontrolador.

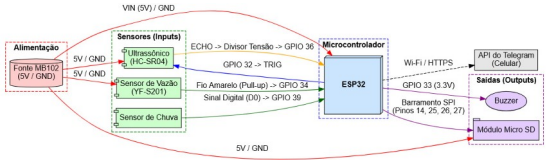

Figura 2. Arquitetura de hardware e conexões elétricas do sistema.
Fonte: Elaborada pelos autores (2026).

Tabela 2. Ligações dos componentes ao ESP32.

Componente	Pino	GPIO ESP32
Sensor Ultrassônico HC-SR04	TRIG	GPIO 32
Sensor Ultrassônico HC-SR04	ECHO	GPIO 36
Sensor de Vazão YF-S201	Sinal	GPIO 34
Sensor de Chuva	D0	GPIO 39
Buzzer	Sinal	GPIO 33
Módulo SD	CS	GPIO 25
Módulo SD	MOSI	GPIO 26
Módulo SD	MISO	GPIO 27
Módulo SD	SCK	GPIO 14

Fonte: Elaborada pelos autores (2026).

Plataforma IoT e comunicação

A comunicação remota foi implementada com a API do Telegram. Após conectar-se à rede Wi-Fi, o ESP32 estabelece uma conexão segura por SSL/TLS e passa a enviar mensagens criptografadas para um chat previamente configurado. Quando uma condição crítica é detectada, a notificação informa a presença de chuva, a distância medida, a vazão e a classificação de risco. Para não sobrecarregar o serviço, definiu-se um intervalo mínimo de 10 segundos entre mensagens.

A lógica de decisão, embarcada no próprio ESP32, funciona assim: estado de alerta resulta da combinação das leituras dos três sensores. Sempre que há chuva e a água se aproxima a menos de 15 cm do sensor, o alerta é emitido; se a vazão ultrapassa cerca de 30 L/min, a situação é classificada como crítica, indicando correnteza intensa. As Figura 3.1 e 3.2 apresentam o fluxograma do software embarcado.

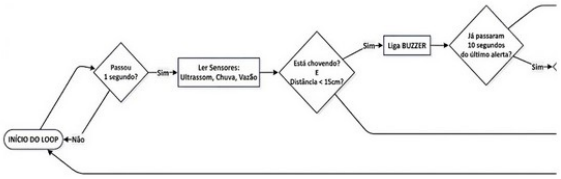

Figura 3.1. Fluxograma de funcionamento do software embarcado.
Fonte: Elaborada pelos autores (2026).

RESULTADOS E DISCUSSÃO

Concluída a montagem e a programação, o protótipo foi submetido a testes para validar tanto os sensores quanto os mecanismos de alerta. O sistema detectou corretamente a chuva, mediu a distância até a superfície da água, calculou a vazão, acionou o buzzer nas condições críticas, enviou notificações pelo Telegram e gravou as medições no cartão SD. A Tabela 3 resume os ensaios realizados.

Tabela 3. Resultados dos testes realizados.

Teste realizado	Resultado
Detecção de chuva	Aprovado
Medição do nível da água	Aprovado
Medição da vazão	Aprovado
Acionamento do buzzer	Aprovado
Envio de mensagens via Telegram	Aprovado
Armazenamento em cartão SD	Aprovado

Fonte: Elaborada pelos autores (2026).

A integração dos três sensores ao ESP32 ocorreu de forma estável, permitindo o monitoramento simultâneo das variáveis ambientais. O armazenamento em cartão SD preservou as informações coletadas, enquanto a comunicação via Telegram possibilitou acompanhar remotamente, em tempo real, as condições monitoradas.

Foi desenvolvida uma caixa protetora para o sistema, para que ficasse portátil e deixasse os sensores posicionados de forma estratégica. As Figuras 4 e 5 mostram, respectivamente, a vista externa e o detalhe da montagem interna. Externamente, o sensor de vazão simula o escoamento em tubulações e o sensor de chuva fica exposto para detecção imediata da precipitação; internamente, o ESP32 concentra o processamento, a comunicação Wi-Fi e o armazenamento redundante.

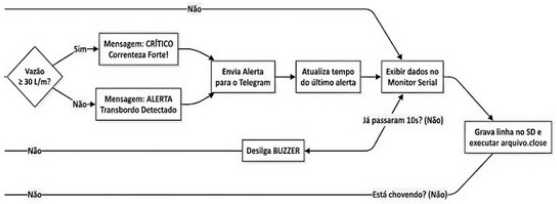

Figura 3.2. Fluxograma de funcionamento do software embarcado.
Fonte: Elaborada pelos autores (2026).

Figura 4. Vista lateral externa do protótipo final.
Fonte: Elaborada pelos autores (2026).

Figura 5. Detalhe da montagem interna e das conexões do ESP32.
Fonte: Elaborada pelos autores (2026).

Os resultados confirmam que tecnologias de baixo custo — o conjunto somou cerca de R$ 243,54 — são eficazes para o monitoramento ambiental. A lógica de decisão embarcada garantiu respostas rápidas, requisito fundamental em um sistema de alerta preventivo. Ao combinar alertas remotos via Telegram e sonoros via buzzer, o protótipo oferece uma camada extra de proteção a comunidades em áreas de risco, favorecendo evacuações preventivas e a redução de danos. Durante o desenvolvimento, os maiores desafios estiveram na integração simultânea dos sensores, na configuração do Wi-Fi e do envio de mensagens e na gravação confiável no cartão SD, superados após sucessivos ajustes de hardware e software.

CONCLUSÃO

O trabalho cumpriu os objetivos propostos: o protótipo integrou com sucesso o ESP32 aos sensores de chuva, nível e vazão, emitindo alertas locais e remotos e mantendo o registro das medições. Mostrou-se uma solução de baixo custo, escalável e de fácil replicação, capaz de unir a robustez do armazenamento local à agilidade da Internet das Coisas.

Como evolução, sugere-se o desenvolvimento de cases à prova d'água para uso em ambientes externos reais e a integração com painéis solares, em busca de autonomia energética, o uso de bombas d'água para detectar o nível enchente de forma mais precisa e resistente a água. Dentre outras e algumas melhorias citadas, é possível que o sistema ofereça uma aplicação efetiva no apoio à defesa civil e na proteção de populações vulneráveis.

AGRADECIMENTOS

Os autores agradecem ao professor Alexandre Magnus Fernandes Guimarães e à Universidade Federal do Rio Grande do Norte pelo apoio e pela orientação ao longo do desenvolvimento deste trabalho.

REFERÊNCIAS

ESPRESSIF SYSTEMS. ESP32 Technical Reference Manual. v. 4.9. Xangai: Espressif Systems, 2023. Disponível em: https://www.espressif.com. Acesso em: 18 maio 2026.

GUBBI, Jayavardhana et al. Internet of Things (IoT): a vision, architectural elements, and future directions. Future Generation Computer Systems, v. 29, n. 7, p. 1645–1660, 2013.

GUIMARÃES, Alexandre Magnus Fernandes. Elétrica básica: divisores de tensão e de corrente. Natal: UFRN, 2026. Notas de aula.

MISHRA, A. et al. IoT-based flood monitoring and forecasting system. Journal of Reliable Intelligent Environments, v. 7, n. 4, 2021.

ORGANIZAÇÃO DAS NAÇÕES UNIDAS (ONU). Objetivo de Desenvolvimento Sustentável 6: água potável e saneamento. 2026. Disponível em: https://brasil.un.org/pt-br/sdgs/6. Acesso em: 18 maio 2026.

SANTOS, João Victor et al. IoT Flood: sistema de monitoramento e alerta de enchentes baseado na Internet das Coisas. In: WORKSHOP DE COMPUTAÇÃO APLICADA À GESTÃO DO

MEIO AMBIENTE E RECURSOS NATURAIS, 2020. Anais [...]. Porto Alegre: SBC, 2020.

ANEXO A – CÓDIGO-FONTE DO SISTEMA

Apresenta-se a seguir o código desenvolvido em C++ para o ESP32, responsável pela aquisição dos dados, processamento, tomada de decisão, acionamento dos alertas e gravação dos registros no cartão Micro SD.

```
#include <SPI.h>
#include <SD.h>
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <UniversalTelegramBot.h>

// 1. CREDENCIAIS DE REDE E TELEGRAM
const char* ssid = "ssid";
const char* password = "senha";
#define BOT_TOKEN "bot_token"
#define CHAT_ID "chat_id"

WiFiClientSecure client;
UniversalTelegramBot bot(BOT_TOKEN, client);

// 2. MAPEAMENTO DE PINOS
#define PINO_SD_CS   25
#define PINO_SD_MOSI 26
#define PINO_SD_MISO 27
#define PINO_SD_SCK  14
#define PINO_VAZAO   34
#define PINO_TRIG    32
#define PINO_ECHO    36
#define PINO_CHUVA   39
#define PINO_BUZZER  33

volatile int contadorPulsos = 0;
float vazaoLPM = 0.0;
unsigned long tempoAnterior = 0;
unsigned long ultimoAlertaTelegram = 0;
const int intervaloAlerta = 10000;
SPIClass spiCustom(VSPI);

void IRAM_ATTR contarPulsos() {
  contadorPulsos++;
}

void setup() {
  Serial.begin(115200);
  Serial.println("\nIniciando Sistema de Monitoramento com IoT...");
  pinMode(PINO_TRIG, OUTPUT);
  pinMode(PINO_ECHO, INPUT);
  pinMode(PINO_CHUVA, INPUT);
  pinMode(PINO_BUZZER, OUTPUT);
  pinMode(PINO_VAZAO, INPUT);
  attachInterrupt(digitalPinToInterrupt(PINO_VAZAO), contarPulsos, FALLING);
  spiCustom.begin(PINO_SD_SCK, PINO_SD_MISO, PINO_SD_MOSI, PINO_SD_CS);
  if (!SD.begin(PINO_SD_CS, spiCustom)) {
    Serial.println("ERRO SD: Falha ao ler o Cartao.");
  } else {
    Serial.println("SUCESSO SD: Cartao SD montado.");
  }
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  client.setCACert(TELEGRAM_CERTIFICATE_ROOT);
  Serial.print("Conectando ao WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nSUCESSO: WiFi Conectado! IP: " + WiFi.localIP().toString());
  digitalWrite(PINO_BUZZER, HIGH);
  delay(200);
  digitalWrite(PINO_BUZZER, LOW);
  bot.sendMessage(CHAT_ID, "Sistema de Telemetria Iniciado e Conectado a Rede!", "");
}

void loop() {
  unsigned long tempoAtual = millis();
```

```
  if (tempoAtual - tempoAnterior >= 1000) {
    digitalWrite(PINO_TRIG, LOW);
    delayMicroseconds(2);
    digitalWrite(PINO_TRIG, HIGH);
    delayMicroseconds(10);
    digitalWrite(PINO_TRIG, LOW);
    long duracaoEco = pulseIn(PINO_ECHO, HIGH);
    float distanciaCm = duracaoEco * 0.034 / 2;
    bool chovendo = (digitalRead(PINO_CHUVA) == LOW);
    vazaoLPM = ((1000.0 / (tempoAtual - tempoAnterior)) * contadorPulsos) / 7.5;
    tempoAnterior = tempoAtual;
    contadorPulsos = 0;
    if (chovendo && distanciaCm < 15.0 && distanciaCm > 0) {
      digitalWrite(PINO_BUZZER, HIGH);
      if (tempoAtual - ultimoAlertaTelegram > intervaloAlerta || ultimoAlertaTelegram == 0) {
        String msg = "";
        if (vazaoLPM >= 30.0) {
          msg += "CRITICO: RISCO DE VIDA!\n";
          msg += "CORRENTEZA FORTE DETECTADA NA ENCHENTE!\n";
          msg += "Evite o contato com a agua e desloque-se para uma area segura.\n\n";
        } else {
          msg += "ALERTA DE TRANSBORDO!\n";
          msg += "Nivel de agua elevado detectado na area monitorada.\n\n";
        }
        msg += "Dados atuais:\n";
        msg += "Chuva: Detectada\n";
        msg += "Distancia do topo: " + String(distanciaCm) + " cm\n";
        msg += "Fluxo da Correnteza: " + String(vazaoLPM) + " L/m";
        bot.sendMessage(CHAT_ID, msg, "");
        ultimoAlertaTelegram = tempoAtual;
        Serial.println(">>> ALERTA DE EMERGENCIA ENVIADO <<<");
      }
    } else {
      digitalWrite(PINO_BUZZER, LOW);
    }
    Serial.print("Dist: "); Serial.print(distanciaCm); Serial.print(" cm | ");
    Serial.print("Vazao: "); Serial.print(vazaoLPM); Serial.print(" L/m | ");
    Serial.print("Chuva: "); Serial.println(chovendo ? "SIM" : "NAO");
    File arquivo = SD.open("/historico_chuva.csv", FILE_APPEND);
    if (arquivo) {
      arquivo.print(tempoAtual); arquivo.print(",");
      arquivo.print(distanciaCm); arquivo.print(",");
      arquivo.print(vazaoLPM); arquivo.print(",");
      arquivo.println(chovendo ? "SIM" : "NAO");
      arquivo.close();
    }
  }
}
```