Nesta etapa vamos usar nosso código da comunicação full-duplex para criar esse sistema com um dispositivo Mestre e um dispositivo escravo. De maneira que, o master é aquele responsável por autorizar o início do slave, receber os dados do slave e enviar os dados para a visualização na tela do computador, além de enviar comandos para o slave executar. Enquanto, a missão do slave é: Receber os dados do sistema (pelos sensores), enviar os dados para o mestre e porfim, obedecer os comandos do mestre.

Para isso, faremos uma simulação utilizando os recursos da própria placa de desenvolvimento, IOT-DevKit, que são: Sensor de temperatura e Umidade, LDR - sensor de luminosidade e o Push Button, seram nossas entradas de dados.

1. Master: **(05/02/25):**

C/C++

```
#if !defined(ARDUINO_ESP32_DEV) // Verifica se o ESP32 está sendo usado
#error Use este exemplo com o ESP32
#endif

// ------------------------------------------------
// Bibliotecas necessárias
#include <RoboCore_SMW_SX1276M0.h>
#include <HardwareSerial.h> // Comunicação serial via hardware

// ------------------------------------------------
// Definição de variáveis
HardwareSerial LoRaSerial(2); // Configura a comunicação LoRa na UART2
#define RXD2 16 // Pino RX do LoRa
#define TXD2 17 // Pino TX do LoRa
#define LED_PIN 13 // Pino do LED de indicação

SMW_SX1276M0 lorawan(LoRaSerial); // Instância do módulo LoRa
CommandResponse response; // Armazena resposta dos comandos LoRa

// Endereços para comunicação ponto a ponto
const char DEVADDR[] = "00000001"; // Endereço do dispositivo atual
const char DEVADDR_P2P[] = "00000002"; // Endereço do destino

//const char APPSKEY[] = "0000000000000000000000000000000c"; // Chave de sessão do aplicativo
//const char NWKSKEY[] = "0000000000000000000000000000000b"; // Chave de sessão da rede

const unsigned long PAUSE_TIME = 15000; // Intervalo de envio (15s)
unsigned long timeout; // Controle de tempo
int count = 0; // Contador de transmissões

// ------------------------------------------------
// Protótipo da função de tratamento de eventos
void event_handler(Event);

// ------------------------------------------------
void setup() {
          Serial.begin(115200);
          Serial.println(F("--- SMW_SX1276M0 P2P ---"));

          pinMode(LED_PIN, OUTPUT); // Configura LED como saída
          digitalWrite(LED_PIN, LOW); // LED começa apagado

          lorawan.setPinReset(5); // Define pino de reset do LoRa
          lorawan.reset(); // Reinicializa módulo LoRa

          LoRaSerial.begin(115200, SERIAL_8N1, RXD2, TXD2); // Inicia comunicação serial com LoRa

          lorawan.event_listener = &event_handler; // Define função de evento
          Serial.println(F("Handler configurado"));
```

```
}

// ------------------------------------------------
void loop() {
          lorawan.listen(); // Mantém LoRa escutando mensagens

          if (timeout < millis()) { // Verifica tempo para próxima transmissão
          count++;
          if (count > 255) count = 0; // Evita estouro de variável

          char data[] = "envie os dados"; // Dados a serem enviados

          Serial.println("=====================================");
          Serial.print(F("Mensagem Enviada Para Slave: "));
          Serial.println(data);
          Serial.println("=====================================");

          response = lorawan.sendT(10, data); // Envia mensagem para a porta 10
          timeout = millis() + PAUSE_TIME; // Atualiza o tempo de envio
          }
}

// ------------------------------------------------
// Função de tratamento de eventos LoRa
void event_handler(Event type) {
          if (type == Event::JOINED) {
          Serial.println(F("Conectado"));
          }
          else if (type == Event::RECEIVED) { // Mensagem recebida
          Serial.println(F("Mensagem de texto recebida"));
          delay(50);
          lorawan.flush();

          uint8_t port;
          Buffer buffer;
          response = lorawan.readT(port, buffer); // Lê dados recebidos
          if (response == CommandResponse::OK) {
          String mensagemRecebida = "";
          while (buffer.available()) {
                    mensagemRecebida += (char)buffer.read(); // Converte dados para string
          }
          Serial.print(F("Mensagem: "));
          Serial.println(mensagemRecebida);
          Serial.print(F(" na porta "));
          Serial.println(port);

          // Se a mensagem for "ACESSO LIBERADO!", acende o LED
          if (mensagemRecebida.equals("aqui os dados")) {
                    Serial.println(F("Recebido: ACESSO LIBERADO! - Ligando LED."));
                    digitalWrite(LED_PIN, HIGH);
          }
          }
          }
          else if (type == Event::RECEIVED_X) { // Mensagem recebida em hexadecimal
          Serial.println(F("!Mensagem Hexadecimal recebida e convertida em texto!"));
          delay(50);
          lorawan.flush();

          uint8_t port;
          Buffer buffer;
          response = lorawan.readX(port, buffer); // Lê dados hexadecimais
          if (response == CommandResponse::OK) {
          Serial.print(F("Mensagem Recebida do slave: "));
          String mensagemHex = "";

          while (buffer.available()) {
                    uint8_t byteHex = buffer.read();
                    mensagemHex += (char)byteHex; // Converte para string
          }

          String mensagemConvertida = "";
          for (int i = 0; i < mensagemHex.length(); i += 2) {
                    String hexByte = mensagemHex.substring(i, i + 2);
                    char caractere = (char)strtol(hexByte.c_str(), NULL, 16);
                    mensagemConvertida += caractere; // Converte HEX para ASCII
          }

          Serial.print(mensagemConvertida);
          Serial.print(F(" | Na porta: "));
          Serial.println(port);

          // Se a mensagem for "ACESSO LIBERADO!", acende e pisca o LED
```

```
          if (mensagemConvertida.equals("ACESSO LIBERADO!")) {
                    Serial.println(F("Recebido: ACESSO LIBERADO! - Ligando LED."));
                    digitalWrite(LED_PIN, HIGH);
                    delay(1000);
                    digitalWrite(LED_PIN, LOW);
                    delay(1000);
          }
          }
          }
}
```

2. Slave: **(05/02/25):**

C/C++

```
#if !defined(ARDUINO_ESP32_DEV) // Verifica se o ESP32 está sendo usado
#error Use este exemplo com o ESP32
#endif

// ------------------------------------------------
// Bibliotecas
#include <RoboCore_SMW_SX1276M0.h>
#include <string.h>
#include <DHT.h> // Biblioteca para o DHT11
#include <HardwareSerial.h> // Comunicação serial por hardware

// ------------------------------------------------
// Variáveis
HardwareSerial LoRaSerial(2); // Usa a UART2 para LoRa
#define RXD2 16 // Pino RX para comunicação LoRa
#define TXD2 17 // Pino TX para comunicação LoRa

// Declaração das variáveis do pino de dados do DHT11
const int pinoDHT = 12;

SMW_SX1276M0 lorawan(LoRaSerial); // Objeto LoRaWAN
CommandResponse response; // Armazena a resposta dos comandos LoRa

// Criação da instância DHT, em função do pino do sensor e do tipo do DHT
DHT dht(pinoDHT, DHT11);

// Endereços para comunicação P2P
const char DEVADDR[] = "00000002"; // Endereço do dispositivo atual
const char DEVADDR_P2P[] = "00000001"; // Endereço de destino P2P

// ------------------------------------------------
// Protótipo da função de tratamento de eventos
void event_handler(Event);

// ------------------------------------------------
void setup() {
          Serial.begin(115200); // Inicializa a UART para debug
          Serial.println(F("--- SMW_SX1276M0 P2P ---"));

          lorawan.setPinReset(5); // Define o pino de reset do módulo
          lorawan.reset(); // Reinicializa o módulo LoRa

          LoRaSerial.begin(115200, SERIAL_8N1, RXD2, TXD2); // Configura a comunicação UART para LoRa

          lorawan.event_listener = &event_handler; // Define o handler de eventos
          Serial.println(F("Handler configurado"));

          pinMode(15, OUTPUT); // Pino usado pelo LDR
          pinMode(12, OUTPUT); // Pino usado pelo DHT11

          // Inicializamos nosso sensor DHT11
          dht.begin();
}

// ------------------------------------------------
void loop() {
```

```
          lorawan.listen(); // Aguarda mensagens recebidas
}

// -------------------------------------------------
// Função para tratar eventos do LoRa
void event_handler(Event type) {
          uint8_t port;
          Buffer buffer;

          if (type == Event::JOINED) { // Evento de conexão
          Serial.println(F("Conectado"));
          }
          else if (type == Event::RECEIVED) { // Texto recebido
          Serial.println(F("Mensagem de texto recebida"));
          delay(50);
          lorawan.flush(); // Limpa buffer de comunicação

          response = lorawan.readT(port, buffer);
          if (response == CommandResponse::OK) {
          Serial.print(F("Mensagem: "));
          while (buffer.available()) {
                    // Lê os dados recebidos
          }
          Serial.print(F("na porta:"));
          }
          }
          else if (type == Event::RECEIVED_X) { // Mensagem recebida em formato hexadecimal
          delay(50);
          lorawan.flush();
          response = lorawan.readX(port, buffer);
          if (response == CommandResponse::OK) {
          String mensagemHex = "";
          while (buffer.available()) {
                    uint8_t byteHex = buffer.read();
                    mensagemHex += (char)byteHex;
          }

          // Converte mensagem hexadecimal para string
          String mensagemConvertida = "";
          for (int i = 0; i < mensagemHex.length(); i += 2) {
                    String hexByte = mensagemHex.substring(i, i + 2);
                    char caractere = (char)strtol(hexByte.c_str(), NULL, 16);
                    mensagemConvertida += caractere;
          }

          Serial.println("================================================");
          Serial.print(F("Mensagem Recebida do Master: "));
          Serial.print(mensagemConvertida);
          Serial.print(F(" | Na porta: "));
          Serial.println(port);
          Serial.println("================================================");

          // Responde com "ACESSO LIBERADO!" caso a mensagem seja "SENHA"
          if (mensagemConvertida.equals("envie os dados")) {
                    Serial.println(F("!Mensagem Hexadecimal recebida e convertida! Enviando resposta para A..."));

                    // NESSE PONTO VAMOS ENVIAR OS DADOS DA PLACA ESCRAVA PARA O MASTER/COMPUTADOR
                    int luminosidade = analogRead(15); // Lê o valor do LDR
                    float temperatura = dht.readTemperature(); // Lê a temperatura do DHT11
                    float umidade = dht.readHumidity(); // Lê a umidade do DHT11

                    char resposta[30];
                    sprintf(resposta, "L:%d T:%.1f U:%.1f", luminosidade, temperatura, umidade); // Converte o
valor para string

                    response = lorawan.sendT(port, resposta);

                    if (response == CommandResponse::OK) {
                    Serial.println(F("Resposta enviada com sucesso!"));
                    } else {
                    Serial.println(F("Erro ao enviar a resposta!"));
                    }
          }
          }
          }
}
```