Nessa Etapa vamos usar a mesma ideia do código anterior, no entanto, todas aquelas configurações feitas via comando AT serão automatizadas nas próprias linhas de código usando os recursos da biblioteca. Mas usando parte do código exemplo SimpleP2P_ESP32, por enquanto retirando todas as confirmações de endereços, chaves, modo e etc, Apenas a confirmação da configuração do Handler

1. Transmissor: **(08/01/25)**

C/C++

```
#if !defined(ARDUINO_ESP32_DEV) // Verificação para garantir que o ESP32 está sendo usado
#error Use este exemplo com o ESP32
#endif

// ----------------------------------------------------
// Bibliotecas

#include <RoboCore_SMW_SX1276M0.h>

// ----------------------------------------------------
// Variáveis

#include <HardwareSerial.h> // Biblioteca para comunicação serial por hardware
HardwareSerial LoRaSerial(2); // Define o canal serial 2 para comunicação LoRa
#define RXD2 16 // Define o pino RX para a comunicação serial LoRa
#define TXD2 17 // Define o pino TX para a comunicação serial LoRa

SMW_SX1276M0 lorawan(LoRaSerial); // Cria o objeto LoRaWAN para o módulo SMW_SX1276M0

CommandResponse response; // Variável para armazenar a resposta dos comandos LoRa

// Endereços e chaves para configuração do dispositivo e comunicação
const char DEVADDR[] = "00000001"; // Endereço do dispositivo atual
const char DEVADDR_P2P[] = "00000002"; // Endereço de destino para comunicação P2P
const char APPSKEY[] = "0000000000000000000000000000000c"; // Chave de sessão do aplicativo
const char NWKSKEY[] = "0000000000000000000000000000000b"; // Chave de sessão da rede

const unsigned long PAUSE_TIME = 15000; // Pausa de 10 segundos entre transmissões (em ms)
unsigned long timeout; // Variável para controle de tempo
int count = 0; // Contador para número de transmissões

// ----------------------------------------------------
// Protótipos

void event_handler(Event); // Função para gerenciar eventos LoRa

// ----------------------------------------------------
// ----------------------------------------------------

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

  // definicao do pino de reset do modulo
  lorawan.setPinReset(5);
  lorawan.reset(); // realiza um reset no modulo

  // Inicializa a UART para o módulo LoRa
  LoRaSerial.begin(115200, SERIAL_8N1, RXD2, TXD2);

  // Define a função de tratamento de eventos
  lorawan.event_listener = &event_handler;
  Serial.println(F("Handler configurado"));
}
// ----------------------------------------------------
// ----------------------------------------------------


void loop(){
  // Escuta dados recebidos do módulo
  lorawan.listen();

  // Envia uma mensagem
  if(timeout < millis()){
    // Atualiza o contador
```

```
    count++;
    if(count > 255){
      count = 0; // Reinicia o contador
    }
    // Converte o contador para HEX
    //for(int i =0; i >= 0; i +=1){
    char data[] = "FPP@pontes";
    //}

    // Envia a mensagem
    Serial.print(F("Dados: "));
    Serial.println(data);
    response = lorawan.sendT(10, data); // NÃO ENVIE "05" na porta 1

    // Atualiza o timeout
    timeout = millis() + PAUSE_TIME;
  }
}

// ------------------------------------------------
// ------------------------------------------------

// Função para tratar os eventos do módulo
//  @param (type) : o tipo do evento [Event]
void event_handler(Event type){
  // Verifica se o evento é de conexão
  if(type == Event::JOINED){
    Serial.println(F("Conectado"));
  }
  // Verifica se um texto foi recebido
  else if(type == Event::RECEIVED){
    Serial.println(F("Mensagem de texto recebida"));

    // Aguarda um tempo para limpar dados restantes do evento
    delay(50);
    lorawan.flush();

    // Lê a mensagem
    uint8_t port;
    Buffer buffer;
    response = lorawan.readT(port, buffer);
    if(response == CommandResponse::OK){
      Serial.print(F("Mensagem: "));
      while(buffer.available()){
        Serial.write(buffer.read());
      }
      Serial.print(F(" na porta "));
      Serial.println(port);
    }
  }
  // Verifica se uma mensagem em hexadecimal foi recebida
  else if(type == Event::RECEIVED_X){
    Serial.println(F("Mensagem HEX recebida"));

    // Aguarda um tempo para limpar dados restantes do evento
    delay(50);
    lorawan.flush();

    // Lê a mensagem
    uint8_t port;
    Buffer buffer;
    response = lorawan.readX(port, buffer);
    if(response == CommandResponse::OK){
      Serial.print(F("Mensagem: "));
      while(buffer.available()){
        Serial.write(buffer.read());
      }
      Serial.print(F(" na porta "));
      Serial.println(port);
    }
  }
}

// ------------------------------------------------
```

2. Receptor: **(08/01/25)**

C/C++

```
#if !defined(ARDUINO_ESP32_DEV)  // Verificação para garantir que o ESP32 está sendo usado
#error Use este exemplo com o ESP32
#endif

// ----------------------------------------------------
// Bibliotecas

#include <RoboCore_SMW_SX1276M0.h>
#include <string.h>

// ----------------------------------------------------
// Variáveis

#include <HardwareSerial.h>    // Biblioteca para comunicação serial por hardware
HardwareSerial LoRaSerial(2);  // Define o canal serial 2 para comunicação LoRa
#define RXD2 16                // Define o pino RX para a comunicação serial LoRa
#define TXD2 17                // Define o pino TX para a comunicação serial LoRa

SMW_SX1276M0 lorawan(LoRaSerial);  // Cria o objeto LoRaWAN para o módulo SMW_SX1276M0

CommandResponse response;  // Variável para armazenar a resposta dos comandos LoRa

// Endereços e chaves para configuração do dispositivo e comunicação
const char DEVADDR[] = "00000002";                          // Endereço do dispositivo atual
const char DEVADDR_P2P[] = "00000001";                      // Endereço de destino para comunicação P2P
const char APPSKEY[] = "0000000000000000000000000000000c";  // Chave de sessão do aplicativo
const char NWKSKEY[] = "0000000000000000000000000000000b";  // Chave de sessão da rede

const unsigned long PAUSE_TIME = 30000;  // Pausa de 30 segundos entre transmissões (em ms)
unsigned long timeout;                   // Variável para controle de tempo
int count = 0;                           // Contador para número de transmissões

// ----------------------------------------------------
// Protótipos

void event_handler(Event);  // Função para gerenciar eventos LoRa

// ----------------------------------------------------
// ----------------------------------------------------

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

  // Definição do pino de reset do módulo
  lorawan.setPinReset(5);
  lorawan.reset();  // Realiza um reset no módulo

  // Inicializa a UART para o módulo LoRa
  LoRaSerial.begin(115200, SERIAL_8N1, RXD2, TXD2);

  // Define a função de tratamento de eventos
  lorawan.event_listener = &event_handler;
  Serial.println(F("Handler configurado"));
}
// ----------------------------------------------------
// ----------------------------------------------------


void loop() {
  // Escuta dados recebidos do módulo
  lorawan.listen();
}

// ----------------------------------------------------
// ----------------------------------------------------

// Função para tratar os eventos do módulo
//  @param (type) : o tipo do evento [Event]
void event_handler(Event type) {
  // Lê a mensagem
  uint8_t port;
  Buffer buffer;
  char lido;
  // Verifica se o evento é de conexão
  if (type == Event::JOINED) {
    Serial.println(F("Conectado"));
  }
  // Verifica se um texto foi recebido
  else if (type == Event::RECEIVED) {
    Serial.println(F("Mensagem de texto recebida"));
```

```
    // Aguarda um tempo para limpar dados restantes do evento
    delay(50);
    lorawan.flush();

    response = lorawan.readT(port, buffer);
    if (response == CommandResponse::OK) {
      Serial.print(F("Mensagem: "));
      while (buffer.available()) {
        //lido = buffer.read();
        //Serial.println(lido);
        //Serial.println('\n');
      }
      Serial.print(F("na porta:"));
      //Serial.println(port);
    }
  }

  else if (type == Event::RECEIVED_X) {
    Serial.println(F("Mensagem TEXTO recebida"));

    // Aguarda um tempo para limpar dados restantes do evento
    delay(50);
    lorawan.flush();

    // Lê a mensagem
    uint8_t port;
    Buffer buffer;
    response = lorawan.readX(port, buffer);
    if (response == CommandResponse::OK) {
      Serial.print(F("Mensagem: "));
      String mensagemHex = "";  // Para armazenar a string hexadecimal

      // Processa cada byte recebido no buffer
      while (buffer.available()) {
        uint8_t byteHex = buffer.read();  // Lê um byte (em hexadecimal)

        // DEBUG: Exibe o valor bruto recebido USEI PARA VERIFICAR
        //Serial.print(F("Byte recebido (HEX): "));
        //Serial.println(byteHex, HEX);

        // Adiciona o caractere correspondente à string hexadecimal
        mensagemHex += (char)byteHex;
      }

      // Converte a string hexadecimal para texto ASCII
      String mensagemConvertida = "";  // Para armazenar o texto final
      for (int i = 0; i < mensagemHex.length(); i += 2) {
        // Lê dois caracteres (ex.: "49") e converte para um byte
        String hexByte = mensagemHex.substring(i, i + 2);
        char caractere = (char)strtol(hexByte.c_str(), NULL, 16);  // Converte de hex para ASCII
        mensagemConvertida += caractere;                           // Adiciona o caractere à string final
      }

      // Exibe a mensagem convertida
      Serial.println(mensagemConvertida);  // Exibe o texto convertido
      Serial.print(F("Na porta: "));
      Serial.println(port);
    }
  }
}

// ------------------------------------------------
```

O Único tipo de Confirmação de configuração usado foi Confirmação de Handler:

O que é o Handler e por que configurá-lo?

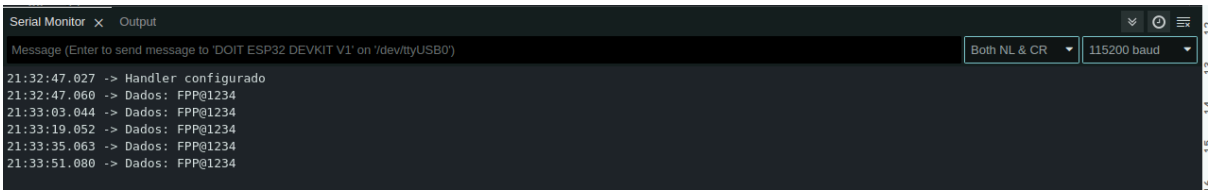
- O **handler** é uma função de callback, ou seja, uma função que será automaticamente chamada pelo sistema sempre que um **evento** específico for detectado no módulo LoRa.
- Essa configuração permite que o programa reaja a eventos como:
    - **JOINED**: Quando o dispositivo se conecta com sucesso à rede ou ao modo P2P.
    - **RECEIVED**: Quando uma mensagem de texto é recebida.
    - **RECEIVED_X**: Quando uma mensagem em formato hexadecimal é recebida.

A configuração do handler garante que o módulo LoRa execute automaticamente a função `event_handler()` assim que algum desses eventos ocorrer.

Confirmação no código

No código fornecido, após configurar o handler, há uma mensagem de depuração para confirmar que o handler foi configurado corretamente:

```
Serial.println(F("Handler configurado"));
```



Essa mensagem é exibida no monitor serial logo após a linha:

```
lorawan.event_listener = &event_handler;
```

→ Isso indica que o sistema está pronto para tratar eventos gerados pelo módulo LoRa.

PORÉM PODEMOS USAR O EXEMPLO DA SimpleP2P_ESP32 PARA CRIAR UMA SÉRIE DE VERIFICAÇÕES:

- DevEUI
- Device Address
- P2P Address
- Chaves

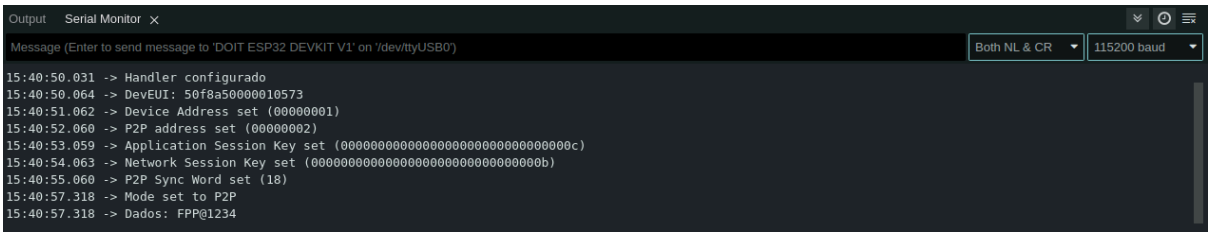
- Modo p2p (on/off)

Para fazer essas modificações é necessário apenas adicionar esses verificações na void setup(), para ficar dessa forma:

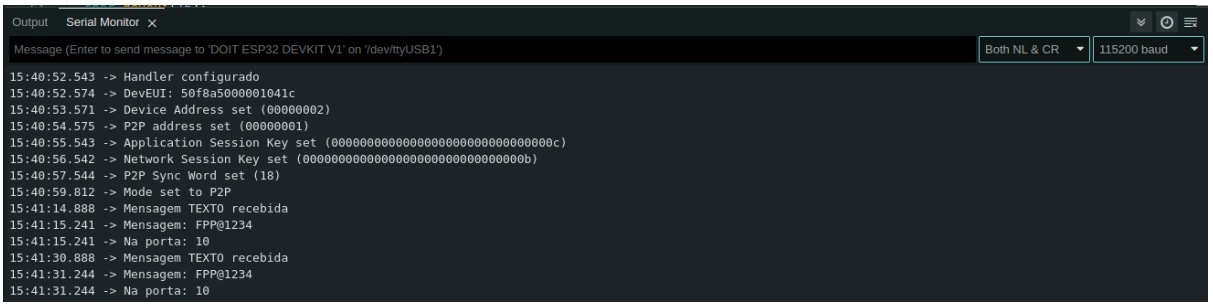
C/C++

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

  // Definição do pino de reset do módulo
  lorawan.setPinReset(5);       // Configura o pino responsável pelo reset do módulo LoRa
  lorawan.reset();              // Executa o reset no módulo LoRa

  // Inicializa a UART para comunicação com o módulo LoRa
  LoRaSerial.begin(115200, SERIAL_8N1, RXD2, TXD2);

  // Configura o handler de eventos para gerenciar respostas e mensagens do módulo
  lorawan.event_listener = &event_handler;
  Serial.println(F("Handler configurado"));

  //------------------------------------------ Testes de configuração ------------------------------------------------

  // Lê o Device EUI do módulo
  char deveui[16];
  response = lorawan.get_DevEUI(deveui);
  if (response == CommandResponse::OK) {
    Serial.print(F("DevEUI: "));               // Exibe o identificador único do dispositivo
    Serial.write((uint8_t *)deveui, 16);       // Escreve o Device EUI na saída serial
    Serial.println();
  } else {
    Serial.println(F("Error getting the Device EUI")); // Exibe mensagem de erro caso não seja possível obter o Device EUI
  }

  // Configura o endereço do dispositivo
  response = lorawan.set_DevAddr(DEVADDR);
  if (response == CommandResponse::OK) {
    Serial.print(F("Device Address set ("));    // Confirma a configuração do endereço do dispositivo
    Serial.write((uint8_t *)DEVADDR, 8);        // Escreve o endereço configurado na saída serial
    Serial.println(')');
  } else {
    Serial.println(F("Error setting the Device Address")); // Exibe mensagem de erro caso a configuração falhe
  }

  // Configura o endereço de destino P2P
  response = lorawan.set_P2P_DevAddr(DEVADDR_P2P);
  if (response == CommandResponse::OK) {
    Serial.print(F("P2P address set ("));       // Confirma a configuração do endereço P2P
    Serial.write((uint8_t *)DEVADDR_P2P, 8);    // Escreve o endereço P2P configurado na saída serial
    Serial.println(')');
  } else {
    Serial.println(F("Error setting the P2P address")); // Exibe mensagem de erro caso a configuração falhe
  }

  // Configura a chave de sessão do aplicativo (AppSKey)
  response = lorawan.set_AppSKey(APPSKEY);
  if (response == CommandResponse::OK) {
    Serial.print(F("Application Session Key set ")); // Confirma a configuração da AppSKey
    Serial.write((uint8_t *)APPSKEY, 32);            // Escreve a AppSKey configurada na saída serial
    Serial.println(')');
  } else {
    Serial.println(F("Error setting the Application Session Key")); // Exibe mensagem de erro caso a configuração falhe
  }

  // Configura a chave de sessão da rede (NwkSKey)
  response = lorawan.set_NwkSKey(NWKSKEY);
  if (response == CommandResponse::OK) {
    Serial.print(F("Network Session Key set (")); // Confirma a configuração da NwkSKey
    Serial.write((uint8_t *)NWKSKEY, 32);         // Escreve a NwkSKey configurada na saída serial
    Serial.println(')');
  } else {
    Serial.println(F("Error setting the Network Session Key")); // Exibe mensagem de erro caso a configuração falhe
  }

  // Configura a palavra de sincronização P2P (opcional)
  const uint8_t SYNC_WORD = 18;  // Define o valor da palavra de sincronização
  response = lorawan.set_P2P_SyncWord(SYNC_WORD);
  if (response == CommandResponse::OK) {
    Serial.print(F("P2P Sync Word set ("));    // Confirma a configuração da palavra de sincronização
    Serial.print(SYNC_WORD);                   // Exibe o valor configurado
    Serial.println(')');
  } else {
    Serial.println(F("Error setting the P2P Sync Word")); // Exibe mensagem de erro caso a configuração falhe
  }

  // Configura o modo de operação para P2P
  response = lorawan.set_JoinMode(SMW_SX1276M0_JOIN_MODE_P2P);
  if (response == CommandResponse::OK) {
    Serial.println(F("Mode set to P2P"));      // Confirma que o modo P2P foi configurado com sucesso
  } else {
    Serial.println(F("Error setting the join mode")); // Exibe mensagem de erro caso a configuração falhe
  }
}
```

Com esse mesma void Setup() aplicada no transmissor e no receptor e os códigos usados no início desse arquivo teremos as seguintes saídas nos monitores do transmissor e do receptor:



Todas as verificações são feitas de maneira automática para o transmissor.



Todas as verificações são feitas de maneira automática para o Receptor.

★ Definições:

1. **Handler** → É uma função configurada (`event_handler`) para gerenciar eventos recebidos do módulo LoRa (ex.: mensagens ou status). No seu caso, ele processa mensagens recebidas via P2P.
2. **DevEUI** → Identificador exclusivo do dispositivo LoRa, usado para identificar de forma única o módulo no ambiente de comunicação.
3. **Device Address set** → Endereço do dispositivo configurado (`DEVADDR`) no módulo LoRa. É necessário para a comunicação P2P e para identificar o remetente em uma transmissão.
4. **Application Session Key** → Chave usada para criptografar e autenticar os dados enviados pela aplicação. Garante que a comunicação seja segura e confiável.
5. **Network Session Key** → Chave usada para proteger os dados de controle e autenticar o dispositivo na rede. Complementa a segurança da comunicação.

6. **P2P Sync Word set** → Palavra de sincronização usada no modo P2P para garantir que apenas dispositivos configurados com o mesmo valor possam se comunicar.
7. **Modo Set P2P** → Configuração do módulo para operar no modo ponto-a-ponto (P2P), permitindo comunicação direta entre dispositivos sem passar por uma rede LoRaWAN.

Pontos Importantes:

1. **conversão de hexadecimal para Texto:**
2.