## Setup

* + The Mesher Library is cloned from <https://github.com/LoRaMesher/LoRaMesher/tree/main>
  + The demo is based on the Counter example in ./examples.
  + We also need to set up PlatformIO in our VSCode editor. We add the existing project into the platformIO which builds the project by downloading required dependencies.
  + We set up the configuration of the platformio.ini file if needed (in case of Heltec boards). The Pin out values needs to be filled and the correct board is to be selected.
  + Then the code is compiled and uploaded into the board.

### Setting Up PlatformIO config:

* [board-heltec]
* board = heltec\_wifi\_lora\_32\_V3
* cs = 8
* rst = 12
* irq = 14
* io1 = 13
* [board-ttgo-beam]
* board = ttgo-t-beam
* cs = 18
* rst = 23
* irq = 26
* io1 = 33
* Define the pin configurations for the different board types – Heltec Lora Wifi 32 V3 and LilyGo T-Display
* [env:heltec\_wifi\_lora\_32\_V3-AE14]
* monitor\_port = COM12
* upload\_port = COM12
* platform = espressif32
* board = heltec\_wifi\_lora\_32\_V3
* framework = arduino
* monitor\_speed = 115200
* lib\_ldf\_mode = deep+
* build\_type = release
* build\_flags =
* -D CORE\_DEBUG\_LEVEL=5
* -D CS=${board-heltec.CS}
* -D RST=${board-heltec.RST}
* -D IRQ=${board-heltec.IRQ}
* -D IO1=${board-heltec.IO1}
* -D SHOULD\_USE\_LMIC=1
* -D HELTEC=1
* Defining the env for each board specifying the upload ports etc. We added the SHOULD\_USE\_LMIC=1 variable definition for the relay node. This definition is not present in other nodes. Also, the HELTEC=1 is added in all the Heltec boards for setting up their SPI configuration in the setup of LoraMesher.
* We removed the automatic library downloading from the `lib\_deps` variable and cloned the LoraMesher and MCCI\_LoraWAN\_LMIC\_library into the `lib` folder of the directory. This allows us to change the code of the library itself.
* We are using the Counter example of the library and the ttn-otaa example of the LMIC library.

### Changes In the Code

* Before starting LMIC we need to configure the project for our boards and frequency band. This can be done in lmic\_project\_config.h file in the ./lib directory where the LMIC library is downloaded.
* // project-specific definitions
* //#define CFG\_eu868 1
* // #define CFG\_us915 1
* //#define CFG\_au915 1
* //#define CFG\_as923 1
* // #define LMIC\_COUNTRY\_CODE LMIC\_COUNTRY\_CODE\_JP      /\* for as923-JP; also define CFG\_as923 \*/
* //#define CFG\_kr920 1
* #define CFG\_in866 1
* // #define CFG\_sx1276\_radio 1
* // #define CFG\_sx1261\_radio 1
* #define CFG\_sx1262\_radio 1
* #define ARDUINO\_heltec\_wifi\_lora\_32\_V3
* //#define LMIC\_USE\_INTERRUPTS
* In the setup function we add the setup of LMIC library when the node is a relay node.
* setupLoraMesher();
* static uint8\_t initData[] = "Hello";
* #ifdef SHOULD\_USE\_LMIC
* SPI.begin(9, 11, 10, CS); //Initialize SPI with the correct pins
* os\_init(); //Initialize the LMIC
* LMIC\_reset(); //Reset the LMIC
* do\_send(initData, sizeof(initData)); //Send the packet to LMIC
* #endif //SHOULD\_USE\_LMIC
* The LoraMesher setup needs to be accommodated for different types of boards and nodes.
* #ifdef HELTEC
* config.module = LoraMesher::LoraModules::SX1262\_MOD;
* #endif
* #ifndef HELTEC
* config.module = LoraMesher::LoraModules::SX1276\_MOD;
* #endif
* Separate SPI config is needed for the Heltec boards
* #ifdef HELTEC
* SPI.begin(9, 11, 10, CS); //Initialize SPI with the correct pins
* config.spi = &SPI;
* #endif
* And the LoraMesher radio need to start only if it’s not a relay n
* ode. We also set our relay node as a gateway node role.
* //Start LoRaMesher
* #ifndef SHOULD\_USE\_LMIC
* radio.start();
* #endif //SHOULD\_USE\_LMIC
* #ifdef SHOULD\_USE\_LMIC
* radio.addGatewayRole();
* #endif
* We copied the event handler of LMIC into the code.
* void onEvent (ev\_t ev) {
* Serial.print(os\_getTime());
* Serial.print(": ");
* switch(ev) {
* case EV\_SCAN\_TIMEOUT:
* Serial.println(F("EV\_SCAN\_TIMEOUT"));
* break;
* case EV\_BEACON\_FOUND:
* Serial.println(F("EV\_BEACON\_FOUND"));
* break;
* case EV\_BEACON\_MISSED:
* Serial.println(F("EV\_BEACON\_MISSED"));
* break;
* case EV\_BEACON\_TRACKED:
* Serial.println(F("EV\_BEACON\_TRACKED"));
* break;
* case EV\_JOINING:
* Serial.println(F("EV\_JOINING"));
* break;
* case EV\_JOINED:
* Serial.println(F("EV\_JOINED"));
* {
* u4\_t netid = 0;
* devaddr\_t devaddr = 0;
* u1\_t nwkKey[16];
* u1\_t artKey[16];
* LMIC\_getSessionKeys(&netid, &devaddr, nwkKey, artKey);
* Serial.print("netid: ");
* Serial.println(netid, DEC);
* Serial.print("devaddr: ");
* Serial.println(devaddr, HEX);
* Serial.print("AppSKey: ");
* for (size\_t i=0; i<sizeof(artKey); ++i) {
* if (i != 0)
* Serial.print("-");
* printHex2(artKey[i]);
* }
* Serial.println("");
* Serial.print("NwkSKey: ");
* for (size\_t i=0; i<sizeof(nwkKey); ++i) {
* if (i != 0)
* Serial.print("-");
* printHex2(nwkKey[i]);
* }
* Serial.println();
* }
* // Disable link check validation (automatically enabled
* // during join, but because slow data rates change max TX
* // size, we don't use it in this example.
* LMIC\_setLinkCheckMode(0);
* hasJoined = true;
* break;
* /\*
* || This event is defined but not used in the code. No
* || point in wasting codespace on it.
* ||
* || case EV\_RFU1:
* ||     Serial.println(F("EV\_RFU1"));
* ||     break;
* \*/
* case EV\_JOIN\_FAILED:
* Serial.println(F("EV\_JOIN\_FAILED"));
* break;
* case EV\_REJOIN\_FAILED:
* Serial.println(F("EV\_REJOIN\_FAILED"));
* break;
* case EV\_TXCOMPLETE:
* Serial.println(F("EV\_TXCOMPLETE (includes waiting for RX windows)"));
* if (LMIC.txrxFlags & TXRX\_ACK)
* Serial.println(F("Received ack"));
* if (LMIC.dataLen) {
* Serial.print(F("Received "));
* Serial.print(LMIC.dataLen);
* Serial.println(F(" bytes of payload"));
* }
* // Schedule next transmission
* os\_setTimedCallback(&sendjob, os\_getTime()+sec2osticks(TX\_INTERVAL), do\_send);
* break;
* case EV\_LOST\_TSYNC:
* Serial.println(F("EV\_LOST\_TSYNC"));
* break;
* case EV\_RESET:
* Serial.println(F("EV\_RESET"));
* break;
* case EV\_RXCOMPLETE:
* // data received in ping slot
* Serial.println(F("EV\_RXCOMPLETE"));
* break;
* case EV\_LINK\_DEAD:
* Serial.println(F("EV\_LINK\_DEAD"));
* break;
* case EV\_LINK\_ALIVE:
* Serial.println(F("EV\_LINK\_ALIVE"));
* break;
* /\*
* || This event is defined but not used in the code. No
* || point in wasting codespace on it.
* ||
* || case EV\_SCAN\_FOUND:
* ||    Serial.println(F("EV\_SCAN\_FOUND"));
* ||    break;
* \*/
* case EV\_TXSTART:
* Serial.println(F("EV\_TXSTART"));
* break;
* case EV\_TXCANCELED:
* Serial.println(F("EV\_TXCANCELED"));
* break;
* case EV\_RXSTART:
* /\* do not print anything -- it wrecks timing \*/
* break;
* case EV\_JOIN\_TXCOMPLETE:
* Serial.println(F("EV\_JOIN\_TXCOMPLETE: no JoinAccept"));
* break;
* default:
* Serial.print(F("Unknown event: "));
* Serial.println((unsigned) ev);
* break;
* }
* }
* For a SX1262 device we need to add certain configurations for the LMIC to work properly.
* class cHalConfiguration\_t: public Arduino\_LMIC::HalConfiguration\_t
* {
* public:
* virtual u1\_t queryBusyPin(void) override {return 13;};
* virtual bool queryUsingDcdc(void) override {return true;};
* virtual bool queryUsingDIO2AsRfSwitch(void) override {return true;};
* virtual bool queryUsingDIO3AsTCXOSwitch(void) override {return true;};
* };
* cHalConfiguration\_t myConfig;
* // Pin mapping
* const lmic\_pinmap lmic\_pins = {
* .nss = CS,
* .rxtx = LMIC\_UNUSED\_PIN,
* .rst = RST,
* .dio = {IRQ, LMIC\_UNUSED\_PIN, LMIC\_UNUSED\_PIN},
* .pConfig = &myConfig,
* };
* Then we added the sender job of the LMIC.
* void do\_send(osjob\_t\* j){
* // Check if there is not a current TX/RX job running
* if (LMIC.opmode & OP\_TXRXPEND) {
* Serial.println(F("OP\_TXRXPEND, not sending"));
* } else {
* // Prepare upstream data transmission at the next possible time.
* LMIC\_setTxData2(1, mydata, sizeof(mydata)-1, 0);
* Serial.println(F("Packet queued"));
* }
* // Next TX is scheduled after TX\_COMPLETE event.
* }
* Changes have been done to both of these codes. First, we don’t need the LMIC to create an OS job for sending data to the gateway. So, we remove the affecting osjob\_t\* j from the arguments. We also need it to send the received data and not the default data.
* void do\_send(uint8\_t mydata[], size\_t size) {
* // Send the payload as a byte array
* // Check if there is not a current TX/RX job running
* if (LMIC.opmode & OP\_TXRXPEND) {
* Serial.println(F("OP\_TXRXPEND, not sending"));
* } else {
* // Prepare upstream data transmission at the next possible time.
* int status = LMIC\_setTxData2(5, mydata, size - 1, 0);
* Serial.println(status);
* if (status == -1)
* {
* Serial.println("Adjusting TX Data Rate... Data Not Sent");
* hasFoundDR = false;
* }
* else if (status == 0)
* {
* Serial.println("Adjusted DR");
* hasFoundDR = true;
* }
* // LMIC\_setTxData(5, mydata, size - 1, 0);
* // LMIC\_setTxData2(1, jsonString, sizeof(mydata)-1, 0);
* Serial.println(F("Packet queued"));
* }
* // Next TX is scheduled after TX\_COMPLETE event.
* }
* The LMIC sender, tries to set up its data rate. If it does that we add a Boolean for this “hasFoundDR” and continuously send a dummy message to quickly set up its transmission DR.
* if (hasFoundDR)
* {
* Serial.println("Restarting Mesher");
* radio.start(); //Start the LoRaMesher to send the next packet
* }
* else{
* uint8\_t data[] = "hello";
* do\_send(data, sizeof(data));
* isUsingLmic = true;
* }
* This is added in the event of TX\_COMPLETE in the event handler. We are also using another Boolean “isUsingLmic” for switching between radios when the LMIC transmission is finished.
* Another Boolean is used to check if the relay node has joined the LoraWAN gateway “hasJoined”
* case EV\_JOINED:
* hasJoined = true;
* These Booleans are used to control the loop of the the LMIC. The lmic checks for events only when its os\_runloop\_once is called in the loop() function.
* #ifdef SHOULD\_USE\_LMIC
* if (!hasJoined || isUsingLmic || !hasFoundDR)
* {
* os\_runloop\_once();
* return;
* }
* #endif
* In the processReceivedPackets of the LoraMesher, we change it such that the relay node will stop the radio, and send the packet to gateway using LMIC
* void processReceivedPackets(void\*) {
* for (;;) {
* /\* Wait for the notification of processReceivedPackets and enter blocking \*/
* ulTaskNotifyTake(pdPASS, portMAX\_DELAY);
* led\_Flash(1, 100); //one quick LED flashes to indicate a packet has arrived
* //Iterate through all the packets inside the Received User Packets Queue
* while (radio.getReceivedQueueSize() > 0) {
* Serial.println("ReceivedUserData\_TaskHandle notify received");
* Serial.printf("Queue receiveUserData size: %d\n", radio.getReceivedQueueSize());
* //Get the first element inside the Received User Packets Queue
* AppPacket<dataPacket>\* packet = radio.getNextAppPacket<dataPacket>();
* //Print the data packet
* printDataPacket(packet);
* #ifdef SHOULD\_USE\_LMIC
* Serial.println("Sending packet to LMIC");
* radio.standby(); //Standby the LoraMesher
* isUsingLmic = true;
* dataPacket\* dPacket = packet->payload;
* uint32\_t x = dPacket->data.counter;
* uint8\_t bytes[8];
* bytes[0] = (x >> 24) & 0xFF;
* bytes[1] = (x >> 16) & 0xFF;
* bytes[2] = (x >> 8) & 0xFF;
* bytes[3] = (x) & 0xFF;
* bytes[4] = '\0';
* uint16\_t src = packet->src;
* bytes[5] = (src >> 8) & 0xFF;
* bytes[6] = (src) & 0xFF;
* bytes[7] = '\0';
* Serial.println(bytes[0]);
* Serial.println(bytes[1]);
* Serial.println(bytes[2]);
* Serial.println(bytes[3]);
* do\_send(bytes, sizeof(bytes));
* delay(1000);
* #endif //SHOULD\_USE\_LMIC
* //Delete the packet when used. It is very important to call this function to release the memory of the packet.
* radio.deletePacket(packet);
* }
* }
* }
* For receiving data from the gateway and sending back to a node we changed the dataPacket definition to include the type of message.
* struct dataPacket {
* int type = 0; // 0 for counter, 1 for wan packet
* union {
* uint32\_t counter = 0;
* uint8\_t data[4];
* } data;
* };
* When the relay node receives a TX\_ACK from the gateway, it creates a new packet and then adds the packet into its send queue.
* case EV\_TXCOMPLETE:
* Serial.println(F("EV\_TXCOMPLETE (includes waiting for RX windows)"));
* if (LMIC.txrxFlags & TXRX\_ACK)
* Serial.println(F("Received ack"));
* if (LMIC.dataLen) {
* Serial.print(F("Received "));
* Serial.print(LMIC.dataLen);
* Serial.println(F(" bytes of payload"));
* returnPacket->type = 1;
* for (int i = 0; i < LMIC.dataLen - 2; i++)
* {
* returnPacket->data.data[i] = (LMIC.frame[LMIC.dataBeg + i]);
* }
* uint16\_t dst = LMIC.frame[LMIC.dataBeg + LMIC.dataLen - 1] + 256 \* LMIC.frame[LMIC.dataBeg + LMIC.dataLen - 2];
* isUsingLmic = false;
* radio.start();
* radio.createPacketAndSend(dst, returnPacket, 1);
* break;
* }
* When a mesh node receives this packet, it checks what the type of the message is and prints it accordingly.
* void printPacket(dataPacket data) {
* if (data.type == 0)
* Serial.printf("Hello Counter received nº %d\n", data.data.counter);
* else{
* for(int i = 0; i < 4; i++)
* {
* Serial.println(data.data.data[i]);
* }
* }
* }
* In order to create a static mesh network for testing purposes, we need to change some variables in the library.
* // Define if is testing
* #define LM\_TESTING
* In BuildOptions.h we define the LM\_TESTING for this purpose.
* In LoraMesher.cpp we change the canReceivePackets function in a way that allows us to form a static routing table.
* #ifdef LM\_TESTING
* const int NUM\_NODES = 4;
* const uint16\_t label\_addresses[NUM\_NODES] = {
* 0xAE14,
* 0xB5B8,
* 0x121C,
* 0x1850,
* };
* const bool adjacency\_matrix[NUM\_NODES][NUM\_NODES] = {
* {0,1,0,0},
* {1,0,1,0},
* {0,1,0,1},
* {0,0,1,0},
* };
* bool LoraMesher::canReceivePacket(uint16\_t source) {
* int idx\_self = -1 ;
* int idx\_other = -1 ;
* uint16\_t this\_addr = getLocalAddress();
* for(int i = 0; i < NUM\_NODES; i++){
* if(label\_addresses[i] == this\_addr){ idx\_self = i; }
* if(label\_addresses[i] == source){ idx\_other = i; }
* }
* if (idx\_self == -1 || idx\_other == -1){return false;}
* return adjacency\_matrix[idx\_self][idx\_other];
* // return (source == NODE1 || source == NODE2);
* }
* #endif
* This checks if a node is allowed to receive a packet otherwise deletes the packet.