\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <lmic.h>

#include <hal/hal.h>

#include <SPI.h>

unsigned long counter = 0;

//

// For normal use, we require that you edit the sketch to replace FILLMEIN

// with values assigned by the TTN console. However, for regression tests,

// we want to be able to compile these scripts. The regression tests define

// COMPILE\_REGRESSION\_TEST, and in that case we define FILLMEIN to a non-

// working but innocuous value.

//

#ifdef COMPILE\_REGRESSION\_TEST

# define FILLMEIN 0

#else

# warning "You must replace the values marked FILLMEIN with real values from the TTN control panel!"

# define FILLMEIN (#dont edit this, edit the lines that use FILLMEIN)

#endif

// This EUI must be in little-endian format, so least-significant-byte

// first. When copying an EUI from ttnctl output, this means to reverse

// the bytes. For TTN issued EUIs the last bytes should be 0xD5, 0xB3,

// 0x70.

static const u1\_t PROGMEM APPEUI[8] = { 0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X00 };

void os\_getArtEui (u1\_t\* buf) {

 memcpy\_P(buf, APPEUI, 8);

}

// This should also be in little endian format, see above.

static const u1\_t PROGMEM DEVEUI[8] = { 0X03, 0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X00 };

void os\_getDevEui (u1\_t\* buf) {

 memcpy\_P(buf, DEVEUI, 8);

}

// This key should be in big endian format (or, since it is not really a

// number but a block of memory, endianness does not really apply). In

// practice, a key taken from ttnctl can be copied as-is.

static const u1\_t PROGMEM APPKEY[16] = { 0x9c, 0xad, 0x10, 0xd4, 0xbc, 0xff, 0x25, 0x1e, 0xbb, 0x8f, 0x9c, 0x49, 0xed, 0x9d, 0xa5, 0xbc };

void os\_getDevKey (u1\_t\* buf) {

 memcpy\_P(buf, APPKEY, 16);

}

static uint8\_t mydata[1] = {0X01};

static osjob\_t sendjob;

// Schedule TX every this many seconds (might become longer due to duty

// cycle limitations).

const unsigned TX\_INTERVAL = 60;

// Pin mapping

const lmic\_pinmap lmic\_pins = {

 .nss = 10,

 .rxtx = LMIC\_UNUSED\_PIN,

 .rst = 5,

 .dio = {7, 8, LMIC\_UNUSED\_PIN},

};

void printHex2(unsigned v) {

 v &= 0xff;

 if (v < 16)

 Serial.print('0');

 Serial.print(v, HEX);

}

void onEvent (ev\_t ev) {

 Serial.print(os\_getTime());

 Serial.print(F(": "));

 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(F("netid: "));

 Serial.println(netid, DEC);

 Serial.print(F("devaddr: "));

 Serial.println(devaddr, HEX);

 Serial.print(F("AppSKey: "));

 for (size\_t i = 0; i < sizeof(artKey); ++i) {

 if (i != 0)

 Serial.print("-");

 printHex2(artKey[i]);

 }

 Serial.println("");

 Serial.print(F("NwkSKey: "));

 for (size\_t i = 0; i < sizeof(nwkKey); ++i) {

 if (i != 0)

 Serial.print(F("-"));

 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);

 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;

 }

}

void do\_send(osjob\_t\* j) {

/\*uncomment below to see network parameter\*/

 Serial.print("Frequency: "); Serial.print(LMIC.freq / 1000000);

 Serial.print("."); Serial.print((LMIC.freq / 100000) % 10);

 Serial.print("MHz");

 Serial.print(" LMIC.datarate: "); Serial.print(LMIC.datarate);

 Serial.print(" LMIC.txpow: "); Serial.println(LMIC.txpow);

 // Check if there is not a current TX/RX job running

 if (LMIC.opmode & OP\_TXRXPEND) {

 Serial.println(F("OP\_TXRXPEND, not sending"));

 } else {

 byte payload[4];

 //nicu send counter - comment to use the old method

 payload[0] = (byte) ((counter & 0xFF000000) >> 24 );

 payload[1] = (byte) ((counter & 0x00FF0000) >> 16 );

 payload[2] = (byte) ((counter & 0x0000FF00) >> 8 );

 payload[3] = (byte) ((counter & 0X000000FF) );

 // Prepare upstream data transmission at the next possible time.

 LMIC\_setTxData2(1, payload, sizeof(payload), 0);

 Serial.println(F("Packet queued"));

 }

 // Next TX is scheduled after TX\_COMPLETE event.

}

void setup() {

 Serial.begin(9600);

 Serial.println(F("Starting"));

 //------ Added ----------------

 attachInterrupt(0, onPulse, RISING);

#ifdef VCC\_ENABLE

 // For Pinoccio Scout boards

 pinMode(VCC\_ENABLE, OUTPUT);

 digitalWrite(VCC\_ENABLE, HIGH);

 delay(1000);

#endif

 // LMIC init

 os\_init();

 // Reset the MAC state. Session and pending data transfers will be discarded.

 LMIC\_reset();

// Use with Arduino Pro Mini ATmega328P 3.3V 8 MHz

 // Let LMIC compensate for +/- 1% clock error

 LMIC\_setClockError(MAX\_CLOCK\_ERROR \* 1 / 100);

 // Start job (sending automatically starts OTAA too)

 do\_send(&sendjob);

}

void loop() {

 os\_runloop\_once();

}

void onPulse()

{

 counter++;

 Serial.println();

 Serial.print(F("counter: "));

 Serial.println(counter);

}