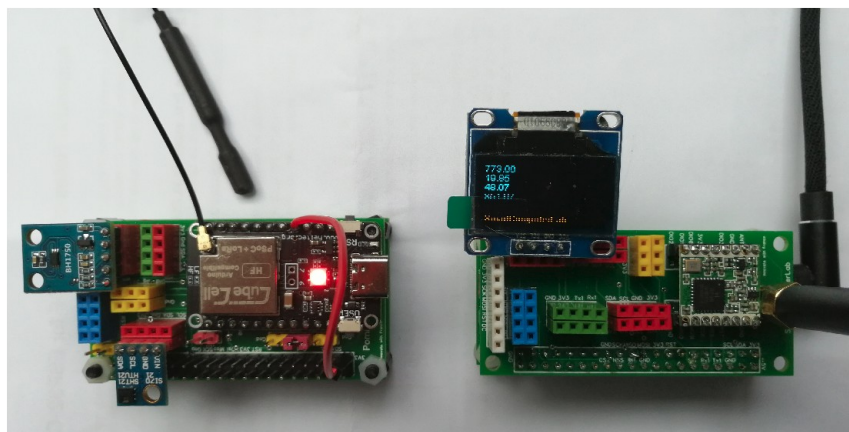# IoT Lab

Sending and receiving LoRa packets

In this exercise we are going to use two different IoT DevKits

- Pomme-Pi ZERO (CORE) with GPIO-LoRa extension card and
- CubleCell Pico: LoRa/LoRaWAN Devkit

as shown on the following figure:

Both boards are programmed with **Arduino IDE** or **PlatformIO**. Attention – the I2C "black" connector on this board (see arrow) has interchanged pins SDA <→SCL

Attention:
This exercise is the **starting point** for our 4 hours lab including the development of complete IoT architecture with low power terminal nodes and IoT gateway with Lora to WiFi relay. The gateway sends the data to MQTT and TS IoT servers.

Content

Attention:

You should install the libraries (tools) for ESP32 and CubeCell boards. Use preferences and board manager to do it.

Remember that you are using **ESP32-C3 (CORE) - `AirM2M_CORE_ESP32C3`** board and **CubeCell** board – **HTCC-AB01** board.
The **MCUs of these boards are completely different**:
ESP32-C3 uses **Extensa-Risc-V (RV32)** micro-processor while CubeCell integrates an **ARM-CORTEX-M0** microprocessor.

1.1 Receiver side on Pomme-Pi ZERO (CORE-RISC-V)

The following is the code for receiver on Pomme-Pi ZERO + GPIO-LoRa board.
Note the use of **`typedef union`** construct to send/receive LoRa packets in a simple but formatted way.

```
typedef union
{
  uint8_t frame[16]; // frames with bytes
  char    mess[16];   // text message
  float   sensor[4];    // 4 float type values for sensor data
} pack_t;
```

The same **`union`** should be used at the sender side.

```
#include <LoRa.h>
#include <Wire.h>
#include "SSD1306Wire.h"
// Initialize the OLED display using Arduino Wire:
SSD1306Wire display(0x3c, 4, 5); // ADDRESS, SDA, SCL

// with LoRa modem RFM95 on GPIO extension board

#define SS 7 // 26 // D0 – to NSS
#define RST 6 //16 // D4 – RST
#define DI0 11 // D8 – INTR

#define SCK 2 // D5 – CLK
#define MISO 10 // D6 – MISO
#define MOSI 3 // D7 – MOSI
#define BAND 8685E5

int sf=9;
long sbw=125E3;

typedef union
{
  uint8_t frame[16]; // frames with bytes
  char    mess[16];   // text message
  float   sensor[4];    // 4 float type values for sensor data
} pack_t;

void disp(char *l1, char *l2, char *l3, char *l4)
{
  display.init();
  //display.flipScreenVertically();
  display.setFont(ArialMT_Plain_10);
  display.setTextAlignment(TEXT_ALIGN_LEFT);
  display.setFont(ArialMT_Plain_10);
  display.drawString(0, 0, l1);
  display.drawString(0, 10, l2);
  display.drawString(0, 20, l3);
  display.drawString(0, 30, l4);
  display.drawString(0, 52, "SmartComputerLab");
  display.display();
}

int rssi=0;

void setup()
{
  Serial.begin(9600);
  SPI.begin(SCK, MISO, MOSI, SS); // SCK, MISO, MOSI, SS
  LoRa.setPins(SS, RST, DI0);
  delay(1000);
  Serial.println();Serial.println();
```

```
    Serial.println("Starting LoRa Receiver");
    if (!LoRa.begin(BAND))
    {
      Serial.println("Starting LoRa failed!"); while (1);
    }
    else
    {
      Serial.println("Starting LoRa ok!");
      LoRa.setSpreadingFactor(sf);
      LoRa.setSignalBandwidth(sbw);
      LoRa.setCodingRate4(5);
    }
}

void loop()
{
int i=0; char b1[32],b2[32],b3[32],b4[32];
pack_t rp;
  // try to parse packet
  int packetSize = LoRa.parsePacket();
  if (packetSize==16)
  {
    Serial.print("Received packet '");
    while (LoRa.available())
    {
      rp.frame[i] = LoRa.read();i++;
    }
    Serial.printf("%2.2f,%2.2f,%2.2f,%2.2f",rp.sensor[0],rp.sensor[1],rp.sensor[2],rp.sensor[3]);
    // print RSSI of packet
    rssi = LoRa.packetRssi();
    Serial.printf("\nRSSI: %d\n",rssi);
    sprintf(b1,"%2.2f",rp.sensor[0]);sprintf(b2,"%2.2f",rp.sensor[1]);
    sprintf(b3,"%2.2f",rp.sensor[2]);sprintf(b4,"%2.2f",rp.sensor[3]);
    disp(b1,b2,b3,b4);
}
```

1.2 Sender side on CubeCell board

```
#include "LoRaWan_APP.h"
#include "Arduino.h"

#include <BH1750.h>
BH1750 lightMeter;

#include <SHT21.h>  // include SHT21 library
SHT21 sht;

#ifndef LoraWan_RGB
#define LoraWan_RGB 0
#endif
#define RF_FREQUENCY                      868500000 // Hz
#define TX_OUTPUT_POWER                   14        // dBm
#define LORA_BANDWIDTH                    0         // [0: 125 kHz,
                                                    //  1: 250 kHz,
                                                    //  2: 500 kHz,
                                                    //  3: Reserved]
#define LORA_SPREADING_FACTOR             9         // [SF7..SF12]
#define LORA_CODINGRATE                   1         // [1: 4/5,
                                                    //  2: 4/6,
                                                    //  3: 4/7,
                                                    //  4: 4/8]
#define LORA_PREAMBLE_LENGTH              8         // Same for Tx and Rx
#define LORA_SYMBOL_TIMEOUT               0         // Symbols
#define LORA_FIX_LENGTH_PAYLOAD_ON        false
#define LORA_IQ_INVERSION_ON              false
#define RX_TIMEOUT_VALUE                  1000
#define BUFFER_SIZE                       128 // Define the payload size here

char txPacket[BUFFER_SIZE];
static RadioEvents_t RadioEvents;
void OnTxDone( void );
void OnTxTimeout( void );

typedef enum
{
```

```
        LOWPOWER,  ReadVTHL,  TX   // 3 states (1,2,3)
} States_t;

States_t state;
bool sleepMode = false;
int16_t rssi,rxSize;
uint16_t voltage;

float temperature, humidity, luminosity;

union pack
{
 uint8_t frame[16]; // trames avec octets
   float  data[4];   // 4 valeurs en virgule flottante
} sdp ;  // paquet d'émission

void setup()
{
    Serial.begin(9600); delay(200);
    pinMode(Vext, OUTPUT);
    digitalWrite(Vext, LOW); delay(100);
    Wire.begin();
    voltage = 0;
    rssi=0;
    RadioEvents.TxDone = OnTxDone;
    RadioEvents.TxTimeout = OnTxTimeout;
    Radio.Init( &RadioEvents );
    Radio.SetChannel( RF_FREQUENCY );
    Radio.SetTxConfig( MODEM_LORA, TX_OUTPUT_POWER, 0, LORA_BANDWIDTH,
                                   LORA_SPREADING_FACTOR, LORA_CODINGRATE,
                                   LORA_PREAMBLE_LENGTH, LORA_FIX_LENGTH_PAYLOAD_ON,
                                   true, 0, 0, LORA_IQ_INVERSION_ON, 3000 );
    state=ReadVTHL;   // read voltage , temperature and humidity
}


void loop()
{
  switch(state)
  {
    case TX:
    {
      sdp.data[0] = (float)voltage;
      sdp.data[1] = temperature;
      sdp.data[2] = humidity;
      sdp.data[3] = luminosity;
      if(voltage<(uint16_t)3680)turnOnRGB(COLOR_SEND,0);
      else turnOnRGB(COLOR_RECEIVED,200);
      Serial.printf("\r\nsending packet- mV:%d, T:%d, H:%d, L:%d\n",voltage,(int)temperature,
(int)humidity,(int)luminosity);
      Radio.Send(sdp.frame,16);
      Serial.println(strlen(txPacket));delay(100);
      state=LOWPOWER;
      break;
    }
    case LOWPOWER:
    {
      lowPowerHandler();delay(100);
      turnOffRGB();
      delay(2000);  //LowPower time
      state = ReadVTHL;
      break;
    }
    case ReadVTHL:
    {
      pinMode(VBAT_ADC_CTL,OUTPUT);
      digitalWrite(VBAT_ADC_CTL,LOW);
      voltage=analogRead(ADC)+550; //*2;
      pinMode(VBAT_ADC_CTL, INPUT);
      pinMode(Vext, OUTPUT);delay(40);
      digitalWrite(Vext, LOW); delay(40);
      Wire.begin();delay(40);
      temperature = sht.getTemperature();  // get temp from SHT
      humidity = sht.getHumidity(); // get temp from SHT
      Serial.print("Temp: ");      // print readings
      Serial.print(temperature);
```

```
            Serial.print("\t Humidity: ");
            Serial.println(humidity);delay(40);
            digitalWrite(Vext,LOW); // start power before activating Wire
            Wire.begin();delay(100);
            lightMeter.begin(); delay(200);    // 200
            luminosity = lightMeter.readLightLevel();
            Serial.print("Light: ");
            Serial.print(luminosity);
            Serial.println(" lux");
            delay(40);
            Wire.end();delay(40);
            digitalWrite(Vext, HIGH); delay(40);
            state = TX;
            break;
        }
        default:
              break;
    }
   Radio.IrqProcess();
}

void OnTxDone( void )
{
  Serial.print("TX done!");
  turnOnRGB(0,0);
}

void OnTxTimeout( void )
{
    Radio.Sleep( );
    Serial.print("TX Timeout......");
    state=ReadVTHL;
    Serial.print(state);
}
```

To do:

Analyze both codes:

- LoRa modem initialization
- LoRa radio parameters
- Lora packets structure and content

Modify the Lora radio parameters

```
void OnTxDone( void )
{
  Serial.print("TX done!");
  turnOnRGB(0,0);
}

void OnTxTimeout( void )
{
    Radio.Sleep( );
    Serial.print("TX Timeout......");
    state=ReadVTHL;
    Serial.print(state);
}
```

To do:

In order to shorten the high power period experiment with shorter `delay()` periods in `ReadVTHL` state.

1.4 Receiver/gateway side on Pomme-Pi ZERO CORE board

```
#include <WiFi.h>
#include "ThingSpeak.h"
const char* ssid     = "Livebox-08B0";          // put your credentials
const char* password = "G79ji6dtEptVTPWmZP";

#include <LoRa.h>
#include <Wire.h>
#include "SSD1306Wire.h"
SSD1306Wire display(0x3c, 4, 5); // ADDRESS, SDA, SCL

// with LoRa modem RFM95 and green RFM board – int and RST to solder

#define SS 7 // 26 // D0 – to NSS
#define RST 6 //16 // D4 – RST
#define DI0 11 // D8 – INTR

#define SCK 2 // D5 – CLK
#define MISO 10 // D6 – MISO
#define MOSI 3 // D7 – MOSI
#define BAND 8685E5

int sf=9;
long sbw=125E3;

typedef union
{
  uint8_t frame[16]; // trames avec octets
  char    mess[16];  // text message
  float   sensor[4];    // 4 valeurs en virgule flottante
} pack_t;


void disp(char *l1, char *l2, char *l3, char *l4)
{
  display.init();
  //display.flipScreenVertically();
  display.setFont(ArialMT_Plain_10);
  display.setTextAlignment(TEXT_ALIGN_LEFT);
  display.setFont(ArialMT_Plain_10);
  display.drawString(0, 0, l1);
  display.drawString(0, 10, l2);
  display.drawString(0, 20, l3);
  display.drawString(0, 30, l4);
  display.drawString(0, 52, "SmartComputerLab");
  display.display();
}

int rssi=0;

WiFiClient  client;

unsigned long myChannelNumber = 1697980;
const char * myWriteAPIKey = "4K897XNNHTW7I4NO";

void setup()
{
  Serial.begin(9600); delay(500);
  Serial.print("[WiFi] Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while(WiFi.status() != WL_CONNECTED)
    {
      Serial.print(".");
      delay(500);
    }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  delay(500);
  ThingSpeak.begin(client);  // Initialize ThingSpeak
  delay(1000);
  SPI.begin(SCK, MISO, MOSI, SS); // SCK, MISO, MOSI, SS
  LoRa.setPins(SS, RST, DI0);
```

```
    delay(1000);
    Serial.println();Serial.println();
    Serial.println("Starting LoRa Receiver");
    if (!LoRa.begin(BAND))
    {
        Serial.println("Starting LoRa failed!"); while (1);
    }
    else
    {
        Serial.println("Starting LoRa ok!");
        LoRa.setSpreadingFactor(sf);
        LoRa.setSignalBandwidth(sbw);
        LoRa.setCodingRate4(5);
    }
}

void loop()
{
int i=0; char b1[32],b2[32],b3[32],b4[32];
pack_t rp;
    // try to parse packet
    int packetSize = LoRa.parsePacket();
    if (packetSize==16)
    {
        Serial.print("Received packet '");
        while (LoRa.available())
        {
            rp.frame[i] = LoRa.read();i++;
        }
        Serial.printf("%2.2f,%2.2f,%2.2f,%2.2f",rp.sensor[0],rp.sensor[1],rp.sensor[2],rp.sensor[3]);
        // print RSSI of packet
        rssi = LoRa.packetRssi();
        Serial.printf("\nRSSI: %d\n",rssi);
        sprintf(b1,"%2.2f",rp.sensor[0]);sprintf(b2,"%2.2f",rp.sensor[1]);
        sprintf(b3,"%2.2f",rp.sensor[2]);sprintf(b4,"%2.2f",rp.sensor[3]);
        disp(b1,b2,b3,b4);
        ThingSpeak.setField(1, rp.sensor[0]);
        ThingSpeak.setField(2, rp.sensor[1]);
        ThingSpeak.setField(3, rp.sensor[2]);
        ThingSpeak.setField(4, rp.sensor[3]);
        ThingSpeak.setField(5, rssi);
        int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
        if(x == 200){
        Serial.println("Channel update successful.");
        }
        else{
            Serial.println("Problem updating channel. HTTP error code " + String(x));
        }
        delay(15000);   // delay necessary for "free" ThingSpeak account
    }
}
```
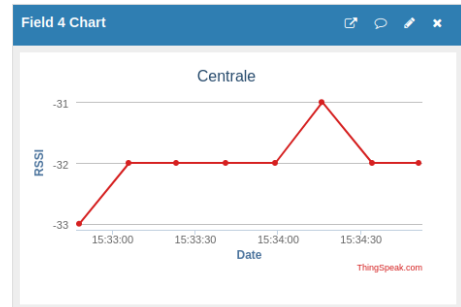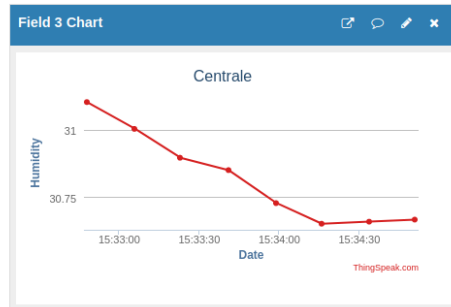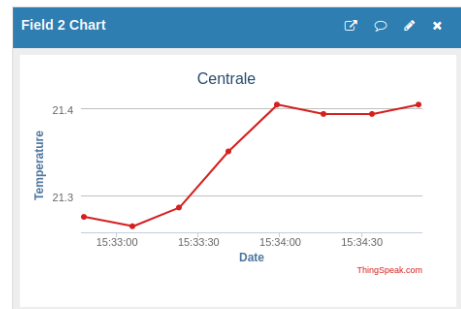
Running result:

```
192.168.1.50

Starting LoRa OK!
V:4069.00,T:21.35,H:30.84
RSSI=-32
Channel update successful.
V:4068.00,T:21.32,H:30.81
RSSI=-33
Channel update successful.
V:4068.00,T:21.34,H:30.71
RSSI=-32
Channel update successful.
V:4069.00,T:21.27,H:30.77
RSSI=-32
Channel update successful.
V:4069.00,T:21.30,H:30.77
RSSI=-33
Channel update successful.
V:4068.00,T
```
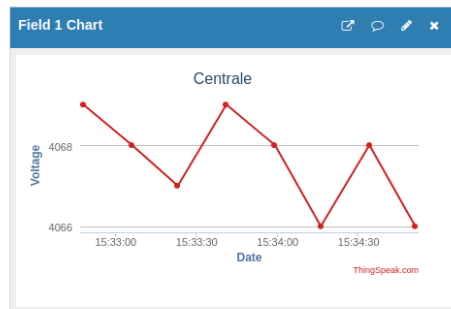
To do:

1. Instead of simple WiFi connection use `WiFiManager` to provide your credentials via local access point and simple web server at : 192.168.1.4
2. Use callback function onReceive to capture the arriving LoRa packets

1.5 Sending and receiving LoRa packets between two Pomme-Pi ZERO (CORE) boards

The following IoT Architecture includes two identical Pomme-Pi ZERO (CORE) boards with additional lora modem (Ra-01) operating on **434 MHz ISM** band.
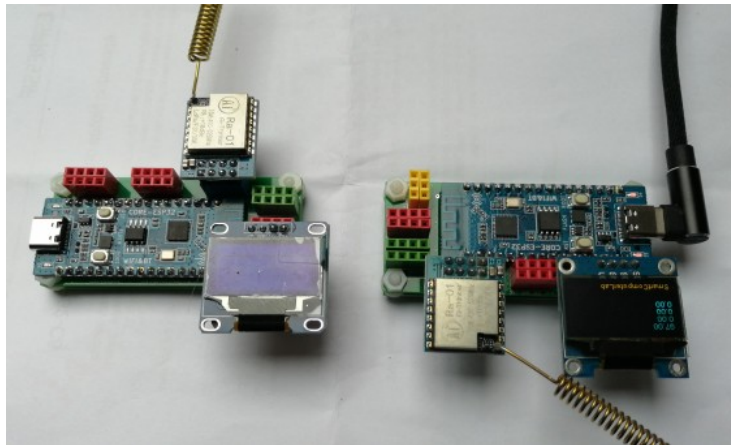
1.5.1 LoRa sender with Ra-01 modem

```
#include <LoRa.h>
#include <Wire.h> // Only needed for Arduino 1.6.5 and earlier
#include "SSD1306Wire.h" // legacy: #include "SSD1306.h"
SSD1306Wire display(0x3c, 4, 5); // ADDRESS, SDA, SCL

#define SS 7            //    NSS
#define RST 6           //    RST
#define DIO 11          //    INTR
#define SCK 2           //    CLK
#define MISO 10         //    MISO
#define MOSI 3          //    MOSI
#define BAND 434E6      // ISM band

int sf=7;               // spreading factor
long sbw=125E3;         // frequency bandwidth

typedef union
{
  uint8_t frame[16];  // frames with bytes
  char    mess[16];   // text message
  float   sensor[4];  // 4 flaot values for sensor data
} pack_t;


void disp(char *l1, char *l2, char *l3, char *l4)
{
  display.init();
  //display.flipScreenVertically();
  display.setFont(ArialMT_Plain_10);
  display.setTextAlignment(TEXT_ALIGN_LEFT);
  display.setFont(ArialMT_Plain_10);
  display.drawString(0, 0, l1);
  display.drawString(0, 10, l2);
  display.drawString(0, 20, l3);
  display.drawString(0, 30, l4);
  display.drawString(0, 52, "SmartComputerLab");
  display.display();
}
pack_t sp;

void setup()
{
  Serial.begin(9600);
  SPI.begin(SCK, MISO, MOSI, SS); // SCK, MISO, MOSI, SS
  LoRa.setPins(SS, RST, DIO);
```

```
  delay(1000);
  Serial.println();Serial.println();
  Serial.println("Starting LoRa Sender");
  if (!LoRa.begin(BAND))
  {
    Serial.println("Starting LoRa failed!"); while (1);
  }
  Serial.println("Starting LoRa ok!");
  LoRa.setSpreadingFactor(sf);
  LoRa.setSignalBandwidth(sbw);
  LoRa.setCodingRate4(5);
  sp.sensor[0]=1.0;sp.sensor[0]=2.0;sp.sensor[0]=3.0;sp.sensor[0]=4.0;
}


void loop()
{
int i=0; char b1[32],b2[32],b3[32],b4[32];
sprintf(b1,"%2.2f",sp.sensor[0]);sprintf(b2,"%2.2f",sp.sensor[1]);
sprintf(b3,"%2.2f",sp.sensor[2]);sprintf(b4,"%2.2f",sp.sensor[3]);
disp(b1,b2,b3,b4);
LoRa.beginPacket();
LoRa.write(sp.frame,16);
LoRa.endPacket();
sp.sensor[0]+=1.0;
delay(5000);
}
```

1.5.2 LoRa receiver with Ra-01 modem

```
#include <LoRa.h>
#include <Wire.h>
#include "SSD1306Wire.h"

SSD1306Wire display(0x3c, 4, 5); // ADDRESS, SDA, SCL
#define SS 7 // 26 // D0 – to NSS
#define RST 6 //16 // D4 – RST
#define DI0 11 // D8 – INTR
#define SCK 2 // D5 – CLK
#define MISO 10 // D6 – MISO
#define MOSI 3 // D7 – MOSI
#define BAND 434E6

int sf=7;
long sbw=125E3;

typedef union
{
  uint8_t frame[16]; // trames avec octets
  char    mess[16];   // text message
  float   sensor[4];    // 4 valeurs en virgule flottante
} pack_t;

void disp(char *l1, char *l2, char *l3, char *l4)
{
  display.init();
  //display.flipScreenVertically();
  display.setFont(ArialMT_Plain_10);
  display.setTextAlignment(TEXT_ALIGN_LEFT);
  display.setFont(ArialMT_Plain_10);
  display.drawString(0, 0, l1);
  display.drawString(0, 10, l2);
  display.drawString(0, 20, l3);
  display.drawString(0, 30, l4);
  display.drawString(0, 52, "SmartComputerLab");
  display.display();
}

int rssi=0;

void setup()
{
  Serial.begin(9600);
  SPI.begin(SCK, MISO, MOSI, SS); // SCK, MISO, MOSI, SS
  LoRa.setPins(SS, RST, DI0);
```

```
      delay(1000);
      Serial.println();Serial.println();
      Serial.println("Starting LoRa Receiver");
      if (!LoRa.begin(BAND))
      {
        Serial.println("Starting LoRa failed!"); while (1);
      }
      else
      {
        Serial.println("Starting LoRa ok!");
        LoRa.setSpreadingFactor(sf);
        LoRa.setSignalBandwidth(sbw);
        LoRa.setCodingRate4(5);
      }
}

void loop()
{
int i=0; char b1[32],b2[32],b3[32],b4[32];
pack_t rp;
  // try to parse packet
  int packetSize = LoRa.parsePacket();
  if (packetSize==16)
  {
    Serial.print("Received packet '");
    while (LoRa.available())
    {
      rp.frame[i] = LoRa.read();i++;
    }
    Serial.printf("%2.2f,%2.2f,%2.2f,%2.2f",rp.sensor[0],rp.sensor[1],rp.sensor[2],rp.sensor[3]);
    // print RSSI of packet
    rssi = LoRa.packetRssi();
    Serial.printf("\nRSSI: %d\n",rssi);
    sprintf(b1,"%2.2f",rp.sensor[0]);sprintf(b2,"%2.2f",rp.sensor[1]);
    sprintf(b3,"%2.2f",rp.sensor[2]);sprintf(b4,"%2.2f",rp.sensor[3]);
    disp(b1,b2,b3,b4);
  }
}
```

1.5.2 LoRa sender with `deep_sleep` state

```
#include <LoRa.h>
#include <Wire.h>
#include "SSD1306Wire.h"
SSD1306Wire display(0x3c, 4, 5); // ADDRESS, SDA, SCL

#define uS_TO_S_FACTOR 1000000ULL     // micro seconds to seconds convertion
#define TIME_TO_SLEEP  10             // time to sleep (in seconds)
RTC_DATA_ATTR float counter;          //  in EEPROM to count number of sleep cycles

#define SS 7                 //  NSS
#define RST 6                //  RST
#define DI0 11               //  INTR
#define SCK 2                //  CLK
#define MISO 10              //  MISO
#define MOSI 3               //  MOSI
#define BAND 434E6

int sf=7;
long sbw=125E3;

typedef union
{
  uint8_t frame[16]; // frames with bytes
  char    mess[16];   // text message
  float   sensor[4];    // 4 flaot values for sensor data
} pack_t;

void disp(char *l1, char *l2, char *l3, char *l4)
{
  display.init();
  //display.flipScreenVertically();
  display.setFont(ArialMT_Plain_10);
  display.setTextAlignment(TEXT_ALIGN_LEFT);
  display.setFont(ArialMT_Plain_10);
```

```
    display.drawString(0, 0, l1);
    display.drawString(0, 10, l2);
    display.drawString(0, 20, l3);
    display.drawString(0, 30, l4);
    display.drawString(0, 52, "SmartComputerLab");
    display.display();
}

pack_t sp;

void setup()
{
    char b1[32],b2[32],b3[32],b4[32];
    Serial.begin(9600);
    SPI.begin(SCK, MISO, MOSI, SS); // SCK, MISO, MOSI, SS
    LoRa.setPins(SS, RST, DI0);
    delay(1000);
    Serial.println();Serial.println();
    Serial.println("Starting LoRa Sender");
    if (!LoRa.begin(BAND))
    {
        Serial.println("Starting LoRa failed!"); while (1);
    }
    Serial.println("Starting LoRa ok!");
    LoRa.setSpreadingFactor(sf);
    LoRa.setSignalBandwidth(sbw);
    LoRa.setCodingRate4(5);
    sp.sensor[0]=sens0;sp.sensor[1]=sens1;sp.sensor[2]=sens2;sp.sensor[3]=sens3;
    sprintf(b1,"%6.2f",sp.sensor[0]);sprintf(b2,"%6.2f",sp.sensor[1]);
    sprintf(b3,"%6.2f",sp.sensor[2]);sprintf(b4,"%6.2f",sp.sensor[3]);
    disp(b1,b2,b3,b4);
    LoRa.beginPacket();
    LoRa.write(sp.frame,16);
    LoRa.endPacket();
    sens0+=1.0;sens1+=2.0;sens2+=3.0;sens3+=4.0;
    esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
    Serial.printf("Going to sleep now for %d seconds",TIME_TO_SLEEP);
    Serial.flush();
    esp_deep_sleep_start();
    Serial.println("This will never be printed");
}

void loop(){
    //This is not going to be called
}
```
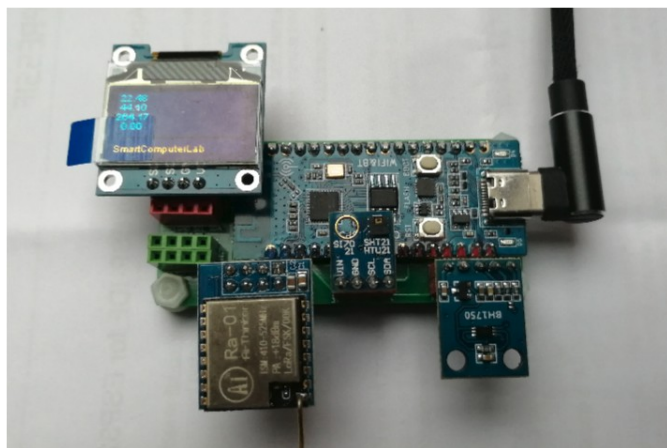
1.5.3 LoRa sender with `deep_sleep` state , OLED and 3 sensors (T/H/L)


```
#include <LoRa.h>
#include <Wire.h>
#include "SSD1306Wire.h"
#include "SHT21.h"
SHT21 sht;
#include <BH1750.h>
BH1750 lightMeter;

SSD1306Wire display(0x3c, 4, 5); // ADDRESS, SDA, SCL

#define uS_TO_S_FACTOR 1000000ULL  /* Conversion factor for micro seconds to seconds */
#define TIME_TO_SLEEP  10        /* Time ESP32 will go to sleep (in seconds) */

RTC_DATA_ATTR float counter;
```

```
// LoRa modem pins on SPI and ESP32C3 - CORE
#define SS 7                    //    NSS
#define RST 6                   //    RST
#define DI0 11                  //    INTR
#define SCK 2                   //    CLK
#define MISO 10                 //    MISO
#define MOSI 3                  //    MOSI
#define BAND 434E6
int sf=7;
long sbw=125E3;

typedef union
{
  uint8_t frame[16]; // frames with bytes
  char    mess[16];    // text message
  float   sensor[4];     // 4 flaot values for sensor data
} pack_t;

void disp(char *l1, char *l2, char *l3, char *l4)
{
  display.init();
  //display.flipScreenVertically();
  display.setFont(ArialMT_Plain_10);
  display.setTextAlignment(TEXT_ALIGN_LEFT);
  display.setFont(ArialMT_Plain_10);
  display.drawString(0, 0, l1);
  display.drawString(0, 10, l2);
  display.drawString(0, 20, l3);
  display.drawString(0, 30, l4);
  display.drawString(0, 52, "SmartComputerLab");
  display.display();
}

pack_t sp;

void setup()
{
  char b1[32],b2[32],b3[32],b4[32];
  Serial.begin(9600);delay(200);
  Wire.begin(4,5);delay(200);
  SPI.begin(SCK, MISO, MOSI, SS); // SCK, MISO, MOSI, SS
  LoRa.setPins(SS, RST, DI0);
  delay(1000);
  Serial.println();Serial.println();
  Serial.println("Starting LoRa Sender");
  if (!LoRa.begin(BAND))
  {
    Serial.println("Starting LoRa failed!"); while (1);
  }
  Serial.println("Starting LoRa ok!");
  LoRa.setSpreadingFactor(sf);
  LoRa.setSignalBandwidth(sbw);
  LoRa.setCodingRate4(5);
  delay(200);
  sp.sensor[1] = sht.getHumidity();     delay(200);
  sp.sensor[0] = sht.getTemperature(); delay(200);
  lightMeter.begin(); delay(200);    // 200
  sp.sensor[2] = lightMeter.readLightLevel();
  Serial.printf("T:%2.2f, H:%2.2f, L:%5.2f\n",sp.sensor[0], sp.sensor[1], sp.sensor[2]);
  //sp.sensor[0]=sens0;sp.sensor[1]=sens1;sp.sensor[2]=sens2;sp.sensor[3]=sens3;
  sprintf(b1,"%6.2f",sp.sensor[0]);sprintf(b2,"%6.2f",sp.sensor[1]);
  sprintf(b3,"%6.2f",sp.sensor[2]);sprintf(b4,"%6.2f",sp.sensor[3]);
  disp(b1,b2,b3,b4);
  LoRa.beginPacket();
  LoRa.write(sp.frame,16);
  LoRa.endPacket();
  esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
  Serial.printf("Going to sleep now for %d seconds",TIME_TO_SLEEP);
  Serial.flush();
  esp_deep_sleep_start();
  Serial.println("This will never be printed");
}

void loop(){
  //This is not going to be called
}
```

1.5.4 LoRa receiver/gateway -WiFi to TS with OLED display

```
#include <WiFi.h>
#include "ThingSpeak.h"
const char* ssid      = "Livebox-08B0";
const char* password = "G79ji6dtEptVTPWmZP";
#include <LoRa.h>
#include <Wire.h>
#include "SSD1306Wire.h"

SSD1306Wire display(0x3c, 4, 5); // ADDRESS, SDA, SCL

#define SS 7                      //   NSS
#define RST 6                     //   RST
#define DI0 11                    //   INTR
#define SCK 2                     //   CLK
#define MISO 10                   //   MISO
#define MOSI 3                    //   MOSI
#define BAND 434E6
int sf=7;
long sbw=125E3;

typedef union
{
  uint8_t frame[16]; // trames avec octets
  char    mess[16];    // text message
  float   sensor[4];     // 4 valeurs en virgule flottante
} pack_t;

void disp(char *l1, char *l2, char *l3, char *l4)
{
  display.init();
  //display.flipScreenVertically();
  display.setFont(ArialMT_Plain_10);
  display.setTextAlignment(TEXT_ALIGN_LEFT);
  display.setFont(ArialMT_Plain_10);
  display.drawString(0, 0, l1);
  display.drawString(0, 10, l2);
  display.drawString(0, 20, l3);
  display.drawString(0, 30, l4);
  display.drawString(0, 52, "SmartComputerLab");
  display.display();
}

int rssi=0;
WiFiClient  client;

unsigned long myChannelNumber = 1697980;
const char * myWriteAPIKey = "4K897XNNHTW7I4NO";

void setup()
{
  Serial.begin(9600); delay(500);
  Serial.print("[WiFi] Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while(WiFi.status() != WL_CONNECTED)
    { Serial.print(".");delay(500); }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  delay(500);
  ThingSpeak.begin(client);  // Initialize ThingSpeak
  delay(1000);
  SPI.begin(SCK, MISO, MOSI, SS); // SCK, MISO, MOSI, SS
  LoRa.setPins(SS, RST, DI0);
  delay(1000);
  Serial.println();Serial.println();
  Serial.println("Starting LoRa Receiver");
  if (!LoRa.begin(BAND))
  {
    Serial.println("Starting LoRa failed!"); while (1);
  }
  Serial.println("Starting LoRa ok!");
  LoRa.setSpreadingFactor(sf);
```

```
  LoRa.setSignalBandwidth(sbw);
  LoRa.setCodingRate4(5);
}

void loop()
{
int i=0; char b1[32],b2[32],b3[32],b4[32];
pack_t rp;
  // try to parse packet
  int packetSize = LoRa.parsePacket();
  if (packetSize==16)
  {
    Serial.print("Received packet '");
    while (LoRa.available())
    {
      rp.frame[i] = LoRa.read();i++;
    }
    Serial.printf("%2.2f,%2.2f,%2.2f,%2.2f",rp.sensor[0],rp.sensor[1],rp.sensor[2],rp.sensor[3]);
   rssi = LoRa.packetRssi();         // get RSSI of packet
   Serial.printf("\nRSSI: %d\n",rssi);
   sprintf(b1,"%2.2f",rp.sensor[0]);sprintf(b2,"%2.2f",rp.sensor[1]);
   sprintf(b3,"%2.2f",rp.sensor[2]);sprintf(b4,"%d",rssi);
   disp(b1,b2,b3,b4);
    ThingSpeak.setField(1, rp.sensor[0]);
    ThingSpeak.setField(2, rp.sensor[1]);
    ThingSpeak.setField(3, rp.sensor[2]);
    ThingSpeak.setField(4, rp.sensor[3]);
    ThingSpeak.setField(5, rssi);
    int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
    if(x == 200){
    Serial.println("Channel update successful.");
    }
    else{
      Serial.println("Problem updating channel. HTTP error code " + String(x));
    }
    delay(15000);
  }
}
```

1.5.5 LoRa receiver/gateway -WiFi to TS with OLED display and reception task

In this receiver example we are using an **independent task** to **parse** the reception buffer of the LoRa modem.
This allows us to free the main **loop()** to run other operations.

```
#include <WiFi.h>
#include "ThingSpeak.h"
const char* ssid     = "Livebox-08B0";
const char* password = "G79ji6dtEptVTPWmZP";
#include <LoRa.h>
#include <Wire.h>
#include "SSD1306Wire.h"
SSD1306Wire display(0x3c, 4, 5); // ADDRESS, SDA, SCL

#define SS 7                    //   NSS
#define RST 6                   //   RST
#define DI0 11                  //   INTR
#define SCK 2                   //   CLK
#define MISO 10                 //   MISO
#define MOSI 3                  //   MOSI
#define BAND 434E6
int sf=7;
long sbw=125E3;

typedef union
{
  uint8_t frame[16]; // trames avec octets
  char    mess[16];   // text message
  float   sensor[4];    // 4 valeurs en virgule flottante
} pack_t;

void disp(char *l1, char *l2, char *l3, char *l4)
{
  display.init();
  //display.flipScreenVertically();
  display.setFont(ArialMT_Plain_10);
```

```
    display.setTextAlignment(TEXT_ALIGN_LEFT);
    display.setFont(ArialMT_Plain_10);
    display.drawString(0, 0, l1);
    display.drawString(0, 10, l2);
    display.drawString(0, 20, l3);
    display.drawString(0, 30, l4);
    display.drawString(0, 52, "SmartComputerLab");
    display.display();
}

int rssi=0;
WiFiClient  client;

unsigned long myChannelNumber = 1697980;
const char * myWriteAPIKey = "4K897XNNHTW7I4NO";

static int taskCore = 0;

void LoRaTask( void * pvParameters )                    // LoRaTask replacing the basic loop() task
{
  while(true)                                           // the task must run in while() loop
  {
  int i=0; char b1[32],b2[32],b3[32],b4[32];
  pack_t rp;
  int packetSize = LoRa.parsePacket();
  if (packetSize==16)
    {
    Serial.print("Received packet '");
    while (LoRa.available())
    {
      rp.frame[i] = LoRa.read();i++;
    }
    Serial.printf("%2.2f,%2.2f,%2.2f,%2.2f",rp.sensor[0],rp.sensor[1],rp.sensor[2],rp.sensor[3]);
    rssi = LoRa.packetRssi();
    Serial.printf("\nRSSI: %d\n",rssi);
    sprintf(b1,"%2.2f",rp.sensor[0]);sprintf(b2,"%2.2f",rp.sensor[1]);
    sprintf(b3,"%2.2f",rp.sensor[2]);sprintf(b4,"%d",rssi);
    disp(b1,b2,b3,b4);
    ThingSpeak.setField(1, rp.sensor[0]);
    ThingSpeak.setField(2, rp.sensor[1]);
    ThingSpeak.setField(3, rp.sensor[2]);
    ThingSpeak.setField(4, rp.sensor[3]);
    ThingSpeak.setField(5, rssi);
    int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
    if(x == 200) Serial.println("Channel update successful.");
    else Serial.println("Problem updating channel. HTTP error code " + String(x));
    delay(15000);
    }
  }
}

void setup()
{
  Serial.begin(9600); delay(500);
  Serial.print("[WiFi] Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while(WiFi.status() != WL_CONNECTED)
    { Serial.print("."); delay(500);}
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  delay(500);
  ThingSpeak.begin(client);  // Initialize ThingSpeak
  delay(1000);
  SPI.begin(SCK, MISO, MOSI, SS); // SCK, MISO, MOSI, SS
  LoRa.setPins(SS, RST, DI0);
  delay(1000);
  Serial.println();Serial.println();
  Serial.println("Starting LoRa Receiver");
  if (!LoRa.begin(BAND))
  {
    Serial.println("Starting LoRa failed!"); while (1);
  }
  Serial.println("Starting LoRa ok!");
```

```
    LoRa.setSpreadingFactor(sf);
    LoRa.setSignalBandwidth(sbw);
    LoRa.setCodingRate4(5);
    Serial.print("Starting to create task on core ");
    Serial.println(taskCore);
    xTaskCreatePinnedToCore(
                    LoRaTask,    /* Function to implement the task */
                    "LoRaTask", /* Name of the task */
                    10000,       /* Stack size in words */
                    NULL,        /* Task input parameter */
                    0,           /* Priority of the task */
                    NULL,        /* Task handle. */
                    taskCore);   /* Core where the task should run */
    Serial.println("Task created...");
}

void loop()              // liberated basi task
{
    Serial.println("in the loop");
    delay(5000);
}
```

1.6 Sender – `deep_sleep` : power consumption analysis

In this section we are going to analyze the power consumption of the sender node operating in `deep_sleep` mode. We use Power Profiler Kit 2 to **source** the sender board and to analyze its power consumption (voltage*current). The regulated voltage is provided by the PPK2.
We start with the previously presented code , this time operating in deep_sleep mode. Note that the main `loop()` is not used, all functions are executed in `setup()` run that ends with a `deep_sleep` state..

USB – V/Gnd
(red/black)
connectors
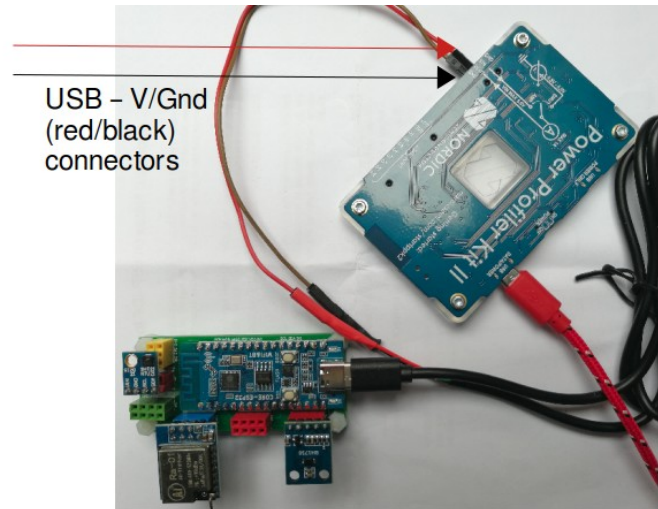
1.6.1 Sender without OLED display

```
#include <LoRa.h>
#include <Wire.h>
#include "SSD1306Wire.h"
#include "SHT21.h"
SHT21 sht;
#include <BH1750.h>
BH1750 lightMeter;
// Initialize the OLED display using Arduino Wire:
SSD1306Wire display(0x3c, 4, 5); // ADDRESS, SDA, SCL

#define uS_TO_S_FACTOR 1000000ULL  /* micro seconds to seconds */
#define TIME_TO_SLEEP  10          /* go to sleep (in seconds) */

RTC_DATA_ATTR float sens0=0.0, sens1=0.0, sens2=0.0, sens3=0.0;

#define SS 7 // 26 // D0 – to NSS
#define RST 6 //16 // D4 – RST
#define DI0 11 // D8 – INTR

#define SCK 2 // D5 – CLK
#define MISO 10 // D6 – MISO
#define MOSI 3 // D7 – MOSI
#define BAND 434E6

int sf=7;
long sbw=125E3;

typedef union
{
  uint8_t frame[16]; // frames with bytes
  char    mess[16];   // text message
  float   sensor[4];    // 4 flaot values for sensor data
} pack_t;

void disp(char *l1, char *l2, char *l3, char *l4)
{
  display.init();
  //display.flipScreenVertically();
  display.setFont(ArialMT_Plain_10);
  display.setTextAlignment(TEXT_ALIGN_LEFT);
  display.setFont(ArialMT_Plain_10);
  display.drawString(0, 0, l1);
  display.drawString(0, 10, l2);
  display.drawString(0, 20, l3);
  display.drawString(0, 30, l4);
  display.drawString(0, 52, "SmartComputerLab");
  display.display();
}

pack_t sp;
```

```
void setup()
{
  char b1[32],b2[32],b3[32],b4[32];
  Serial.begin(9600);delay(200);
  Wire.begin(4,5);delay(200);
  SPI.begin(SCK, MISO, MOSI, SS); // SCK, MISO, MOSI, SS
  LoRa.setPins(SS, RST, DI0);
  delay(1000);
  Serial.println();Serial.println();
  Serial.println("Starting LoRa Sender");
  if (!LoRa.begin(BAND))
  {
    Serial.println("Starting LoRa failed!"); while (1);
  }
  Serial.println("Starting LoRa ok!");
  LoRa.setSpreadingFactor(sf);
  LoRa.setSignalBandwidth(sbw);
  LoRa.setCodingRate4(5);
  delay(200);
  sp.sensor[1] = sht.getHumidity(); delay(200);
  sp.sensor[0] = sht.getTemperature();  // get temp/humi from SHT
  delay(200);
  lightMeter.begin(); delay(200);   // 200
  sp.sensor[2] = lightMeter.readLightLevel();
  Serial.printf("T:%2.2f, H:%2.2f, L:%5.2f\n",sp.sensor[0], sp.sensor[1], sp.sensor[2]);
  sprintf(b1,"%6.2f",sp.sensor[0]);sprintf(b2,"%6.2f",sp.sensor[1]);
  sprintf(b3,"%6.2f",sp.sensor[2]);sprintf(b4,"%6.2f",sp.sensor[3]);
  //disp(b1,b2,b3,b4);                  // not used
  LoRa.beginPacket();
  LoRa.write(sp.frame,16);
  LoRa.endPacket();
  esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
  Serial.printf("Going to sleep now for %d seconds",TIME_TO_SLEEP);
  Serial.flush();
  esp_deep_sleep_start();
  Serial.println("This will never be printed");
}

void loop(){
  //This is not going to be called
}
```


To do:

Add and activate the OLED display and analyze the power consumption.

Assignment (for students using Pomme-Pi ZERO (CORE)

After testing the presented IoT Architecture examples let us extend it with new features.

1. LoRa packets are not protected; so we have to add encryption to hide the payload. We can do it with **AES encryption available in hardware** with ESP32.
Note that AES system uses 16-byte segments to encrypt/decrypt the data with a 128-bit shared key.

2. The terminals should not communicate with other terminals. In order to separate the communication between the terminals and the gateway we may use **down-chirp**/**up-chirp** modes.
For example the terminal nodes send the "**up-chirp**" packets to the gateway and receive the "**down-chirp**" packets ACK from the gateway.

Appendix

A.1 AES encryption

A.1.1 AES "hardware" encryption for ESP32

The following example shows the use of AES encryption mechanism for embedded accelerator in ESP32 SoC. The encrypted/decrypted byte frame is 32-byte long; in any case it has to be multiple of 16 bytes.

```
#include "mbedtls/aes.h"

void encrypt(unsigned char *plainText,char *key,unsigned char *outputBuffer, int nblocks)
{
  mbedtls_aes_context aes;
  mbedtls_aes_init( &aes );
  mbedtls_aes_setkey_enc(&aes,(const unsigned char*)key,strlen(key)*8);
  for(int i=0;i<nblocks;i++)
    {
    mbedtls_aes_crypt_ecb(&aes,MBEDTLS_AES_ENCRYPT,
                      (const unsigned char*)(plainText+i*16), outputBuffer+i*16);
    }
  mbedtls_aes_free(&aes);
}

void decrypt(unsigned char *chipherText,char *key,unsigned char *outputBuffer, int nblocks)
{
  mbedtls_aes_context aes;
  mbedtls_aes_init( &aes );
  mbedtls_aes_setkey_dec( &aes, (const unsigned char*) key, strlen(key) * 8 );
    for(int i=0;i<nblocks;i++)
    {
    mbedtls_aes_crypt_ecb(&aes,MBEDTLS_AES_DECRYPT,
                      (const unsigned char*)(chipherText+i*16), outputBuffer+i*16);
    }
  mbedtls_aes_free(&aes );
}

void setup() {

  mbedtls_aes_context aes;
  Serial.begin(9600);delay(400);
  Serial.println(); Serial.println();

  //char key[16]= { 0x00, 0x01, 0x02, 0x03,0x00, 0x01, 0x02, 0x03,0x00, 0x01, 0x02, 0x03,0x00, 0x01,
0x02, 0x03};
  char *key = "abcdefghijklmnop";
  unsigned char input[16] = { 0xAB, 0xCD, 0xAC, 0xB8,0x8A, 0x77, 0xA6, 0xA6,0x8B, 0xC1, 0xD2,
0xF3,0xBB, 0xF1, 0xF2, 0xF3};
  //unsigned char *input = (unsigned char *)"SmartComputerLabSmartComputerLab";
  unsigned char crypte[16], decrypte[16];

  Serial.println();  Serial.println();delay(400);
  for(int i=0; i<16;i++) Serial.print(input[i],HEX);
  Serial.println();
  encrypt(input, key, crypte, 1);
  for(int i=0; i<16;i++) Serial.print(crypte[i],HEX);
  Serial.println();delay(400);
  Serial.println();
  decrypt(crypte, key, decrypte,1);
  for(int i=0; i<16;i++) Serial.print(decrypte[i],HEX);
  Serial.println();delay(400);
}

void loop() { }
```

Execution result:

```
ABCDACB88A77A6A68BC1D2F3BBF1F2F3
E78C3A1E3D356A116AF49CC77EC773C

ABCDACB88A77A6A68BC1D2F3BBF1F2F3
```

A.1.1 AES "software" encryption/decryption for CubeCell

The encryption for CubeCell (ARM) must be done via software. The following are two necessary files to add (include) to your sketch: **aes.h** and **aes.c**

```
https://github.com/HelTecAutomation/CubeCell-Arduino/tree/master/cores/asr650x/
lora/system/crypto
```

A.1.1.1 aes.h

```c
/*
 ---------------------------------------------------------------------------
 Copyright (c) 1998-2008, Brian Gladman, Worcester, UK. All rights reserved.

 LICENSE TERMS

 The redistribution and use of this software (with or without changes)
 is allowed without the payment of fees or royalties provided that:

  1. source code distributions include the above copyright notice, this
     list of conditions and the following disclaimer;

  2. binary distributions include the above copyright notice, this list
     of conditions and the following disclaimer in their documentation;

  3. the name of the copyright holder is not used to endorse products
     built using this software without specific written permission.

 DISCLAIMER

 This software is provided 'as is' with no explicit or implied warranties
 in respect of its properties, including, but not limited to, correctness
 and/or fitness for purpose.
 ---------------------------------------------------------------------------
 Issue 09/09/2006

 This is an AES implementation that uses only 8-bit byte operations on the
 cipher state.
*/

#ifndef AES_H
#define AES_H

#if 1
#  define AES_ENC_PREKEYED  /* AES encryption with a precomputed key schedule  */
#endif
#if 1
#  define AES_DEC_PREKEYED  /* AES decryption with a precomputed key schedule  */
#endif
#if 0
#  define AES_ENC_128_OTFK  /* AES encryption with 'on the fly' 128 bit keying */
#endif
#if 0
#  define AES_DEC_128_OTFK  /* AES decryption with 'on the fly' 128 bit keying */
#endif
#if 0
#  define AES_ENC_256_OTFK  /* AES encryption with 'on the fly' 256 bit keying */
#endif
#if 0
#  define AES_DEC_256_OTFK  /* AES decryption with 'on the fly' 256 bit keying */
#endif

#define N_ROW                   4
#define N_COL                   4
#define N_BLOCK    (N_ROW * N_COL)
#define N_MAX_ROUNDS           14

typedef uint8_t return_type;

/*  Warning: The key length for 256 bit keys overflows a byte
    (see comment below)
*/

typedef uint8_t length_type;

typedef struct
```

```
{   uint8_t ksch[(N_MAX_ROUNDS + 1) * N_BLOCK];
    uint8_t rnd;
} aes_context;

/*  The following calls are for a precomputed key schedule

    NOTE: If the length_type used for the key length is an
    unsigned 8-bit character, a key length of 256 bits must
    be entered as a length in bytes (valid inputs are hence
    128, 192, 16, 24 and 32).
*/

#if defined( AES_ENC_PREKEYED ) || defined( AES_DEC_PREKEYED )

return_type aes_set_key( const uint8_t key[],
                         length_type keylen,
                         aes_context ctx[1] );
#endif

#if defined( AES_ENC_PREKEYED )

return_type aes_encrypt( const uint8_t in[N_BLOCK],
                         uint8_t out[N_BLOCK],
                         const aes_context ctx[1] );

return_type aes_cbc_encrypt( const uint8_t *in,
                         uint8_t *out,
                         int32_t n_block,
                         uint8_t iv[N_BLOCK],
                         const aes_context ctx[1] );
#endif

#if defined( AES_DEC_PREKEYED )

return_type aes_decrypt( const uint8_t in[N_BLOCK],
                         uint8_t out[N_BLOCK],
                         const aes_context ctx[1] );

return_type aes_cbc_decrypt( const uint8_t *in,
                         uint8_t *out,
                         int32_t n_block,
                         uint8_t iv[N_BLOCK],
                         const aes_context ctx[1] );
#endif

/*  The following calls are for 'on the fly' keying.  In this case the
    encryption and decryption keys are different.

    The encryption subroutines take a key in an array of bytes in
    key[L] where L is 16, 24 or 32 bytes for key lengths of 128,
    192, and 256 bits respectively.  They then encrypts the input
    data, in[] with this key and put the reult in the output array
    out[].  In addition, the second key array, o_key[L], is used
    to output the key that is needed by the decryption subroutine
    to reverse the encryption operation.  The two key arrays can
    be the same array but in this case the original key will be
    overwritten.

    In the same way, the decryption subroutines output keys that
    can be used to reverse their effect when used for encryption.

    Only 128 and 256 bit keys are supported in these 'on the fly'
    modes.
*/

#if defined( AES_ENC_128_OTFK )
void aes_encrypt_128( const uint8_t in[N_BLOCK],
                      uint8_t out[N_BLOCK],
                      const uint8_t key[N_BLOCK],
                      uint8_t o_key[N_BLOCK] );
#endif

#if defined( AES_DEC_128_OTFK )
void aes_decrypt_128( const uint8_t in[N_BLOCK],
                      uint8_t out[N_BLOCK],
                      const uint8_t key[N_BLOCK],
```

```
                                      uint8_t o_key[N_BLOCK] );
#endif


#if defined( AES_ENC_256_OTFK )
void aes_encrypt_256( const uint8_t in[N_BLOCK],
                            uint8_t out[N_BLOCK],
                            const uint8_t key[2 * N_BLOCK],
                            uint8_t o_key[2 * N_BLOCK] );
#endif


#if defined( AES_DEC_256_OTFK )
void aes_decrypt_256( const uint8_t in[N_BLOCK],
                            uint8_t out[N_BLOCK],
                            const uint8_t key[2 * N_BLOCK],
                            uint8_t o_key[2 * N_BLOCK] );
#endif


#endif
```

A.1.1.2 `aes.c`

```
/*
 ---------------------------------------------------------------------------
 Copyright (c) 1998-2008, Brian Gladman, Worcester, UK. All rights reserved.

 LICENSE TERMS

 The redistribution and use of this software (with or without changes)
 is allowed without the payment of fees or royalties provided that:

  1. source code distributions include the above copyright notice, this
     list of conditions and the following disclaimer;

  2. binary distributions include the above copyright notice, this list
     of conditions and the following disclaimer in their documentation;

  3. the name of the copyright holder is not used to endorse products
     built using this software without specific written permission.

 DISCLAIMER

 This software is provided 'as is' with no explicit or implied warranties
 in respect of its properties, including, but not limited to, correctness
 and/or fitness for purpose.
 ---------------------------------------------------------------------------
 Issue 09/09/2006

 This is an AES implementation that uses only 8-bit byte operations on the
 cipher state (there are options to use 32-bit types if available).

 The combination of mix columns and byte substitution used here is based on
 that developed by Karl Malbrain. His contribution is acknowledged.
 */

/* define if you have a fast memcpy function on your system */
#if 0
#  define HAVE_MEMCPY
#  include <string.h>
#  if defined( _MSC_VER )
#    include <intrin.h>
#    pragma intrinsic( memcpy )
#  endif
#endif


#include <stdlib.h>
#include <stdint.h>

/* define if you have fast 32-bit types on your system */
#if 0
#  define HAVE_UINT_32T
#endif

/* define if you don't want any tables */
#if 1
```

```c
#   define USE_TABLES
#endif

/*  On Intel Core 2 duo VERSION_1 is faster */

/* alternative versions (test for performance on your system) */
#if 1
#   define VERSION_1
#endif

#include "aes.h"

//#if defined( HAVE_UINT_32T )
//  typedef unsigned long uint32_t;
//#endif

/* functions for finite field multiplication in the AES Galois field    */

#define WPOLY   0x011b
#define BPOLY    0x1b
#define DPOLY   0x008d

#define f1(x)    (x)
#define f2(x)    ((x << 1) ^ (((x >> 7) & 1) * WPOLY))
#define f4(x)    ((x << 2) ^ (((x >> 6) & 1) * WPOLY) ^ (((x >> 6) & 2) * WPOLY))
#define f8(x)    ((x << 3) ^ (((x >> 5) & 1) * WPOLY) ^ (((x >> 5) & 2) * WPOLY) \
                          ^ (((x >> 5) & 4) * WPOLY))
#define d2(x)    (((x) >> 1) ^ ((x) & 1 ? DPOLY : 0))

#define f3(x)    (f2(x) ^ x)
#define f9(x)    (f8(x) ^ x)
#define fb(x)    (f8(x) ^ f2(x) ^ x)
#define fd(x)    (f8(x) ^ f4(x) ^ x)
#define fe(x)    (f8(x) ^ f4(x) ^ f2(x))

#if defined( USE_TABLES )

#define sb_data(w) {    /* S Box data values */                          \
    w(0x63), w(0x7c), w(0x77), w(0x7b), w(0xf2), w(0x6b), w(0x6f), w(0xc5),\
    w(0x30), w(0x01), w(0x67), w(0x2b), w(0xfe), w(0xd7), w(0xab), w(0x76),\
    w(0xca), w(0x82), w(0xc9), w(0x7d), w(0xfa), w(0x59), w(0x47), w(0xf0),\
    w(0xad), w(0xd4), w(0xa2), w(0xaf), w(0x9c), w(0xa4), w(0x72), w(0xc0),\
    w(0xb7), w(0xfd), w(0x93), w(0x26), w(0x36), w(0x3f), w(0xf7), w(0xcc),\
    w(0x34), w(0xa5), w(0xe5), w(0xf1), w(0x71), w(0xd8), w(0x31), w(0x15),\
    w(0x04), w(0xc7), w(0x23), w(0xc3), w(0x18), w(0x96), w(0x05), w(0x9a),\
    w(0x07), w(0x12), w(0x80), w(0xe2), w(0xeb), w(0x27), w(0xb2), w(0x75),\
    w(0x09), w(0x83), w(0x2c), w(0x1a), w(0x1b), w(0x6e), w(0x5a), w(0xa0),\
    w(0x52), w(0x3b), w(0xd6), w(0xb3), w(0x29), w(0xe3), w(0x2f), w(0x84),\
    w(0x53), w(0xd1), w(0x00), w(0xed), w(0x20), w(0xfc), w(0xb1), w(0x5b),\
    w(0x6a), w(0xcb), w(0xbe), w(0x39), w(0x4a), w(0x4c), w(0x58), w(0xcf),\
    w(0xd0), w(0xef), w(0xaa), w(0xfb), w(0x43), w(0x4d), w(0x33), w(0x85),\
    w(0x45), w(0xf9), w(0x02), w(0x7f), w(0x50), w(0x3c), w(0x9f), w(0xa8),\
    w(0x51), w(0xa3), w(0x40), w(0x8f), w(0x92), w(0x9d), w(0x38), w(0xf5),\
    w(0xbc), w(0xb6), w(0xda), w(0x21), w(0x10), w(0xff), w(0xf3), w(0xd2),\
    w(0xcd), w(0x0c), w(0x13), w(0xec), w(0x5f), w(0x97), w(0x44), w(0x17),\
    w(0xc4), w(0xa7), w(0x7e), w(0x3d), w(0x64), w(0x5d), w(0x19), w(0x73),\
    w(0x60), w(0x81), w(0x4f), w(0xdc), w(0x22), w(0x2a), w(0x90), w(0x88),\
    w(0x46), w(0xee), w(0xb8), w(0x14), w(0xde), w(0x5e), w(0x0b), w(0xdb),\
    w(0xe0), w(0x32), w(0x3a), w(0x0a), w(0x49), w(0x06), w(0x24), w(0x5c),\
    w(0xc2), w(0xd3), w(0xac), w(0x62), w(0x91), w(0x95), w(0xe4), w(0x79),\
    w(0xe7), w(0xc8), w(0x37), w(0x6d), w(0x8d), w(0xd5), w(0x4e), w(0xa9),\
    w(0x6c), w(0x56), w(0xf4), w(0xea), w(0x65), w(0x7a), w(0xae), w(0x08),\
    w(0xba), w(0x78), w(0x25), w(0x2e), w(0x1c), w(0xa6), w(0xb4), w(0xc6),\
    w(0xe8), w(0xdd), w(0x74), w(0x1f), w(0x4b), w(0xbd), w(0x8b), w(0x8a),\
    w(0x70), w(0x3e), w(0xb5), w(0x66), w(0x48), w(0x03), w(0xf6), w(0x0e),\
    w(0x61), w(0x35), w(0x57), w(0xb9), w(0x86), w(0xc1), w(0x1d), w(0x9e),\
    w(0xe1), w(0xf8), w(0x98), w(0x11), w(0x69), w(0xd9), w(0x8e), w(0x94),\
    w(0x9b), w(0x1e), w(0x87), w(0xe9), w(0xce), w(0x55), w(0x28), w(0xdf),\
    w(0x8c), w(0xa1), w(0x89), w(0x0d), w(0xbf), w(0xe6), w(0x42), w(0x68),\
    w(0x41), w(0x99), w(0x2d), w(0x0f), w(0xb0), w(0x54), w(0xbb), w(0x16) }

#define isb_data(w) {   /* inverse S Box data values */                      \
    w(0x52), w(0x09), w(0x6a), w(0xd5), w(0x30), w(0x36), w(0xa5), w(0x38),\
    w(0xbf), w(0x40), w(0xa3), w(0x9e), w(0x81), w(0xf3), w(0xd7), w(0xfb),\
    w(0x7c), w(0xe3), w(0x39), w(0x82), w(0x9b), w(0x2f), w(0xff), w(0x87),\
    w(0x34), w(0x8e), w(0x43), w(0x44), w(0xc4), w(0xde), w(0xe9), w(0xcb),\
```

```
    w(0x54), w(0x7b), w(0x94), w(0x32), w(0xa6), w(0xc2), w(0x23), w(0x3d),\
    w(0xee), w(0x4c), w(0x95), w(0x0b), w(0x42), w(0xfa), w(0xc3), w(0x4e),\
    w(0x08), w(0x2e), w(0xa1), w(0x66), w(0x28), w(0xd9), w(0x24), w(0xb2),\
    w(0x76), w(0x5b), w(0xa2), w(0x49), w(0x6d), w(0x8b), w(0xd1), w(0x25),\
    w(0x72), w(0xf8), w(0xf6), w(0x64), w(0x86), w(0x68), w(0x98), w(0x16),\
    w(0xd4), w(0xa4), w(0x5c), w(0xcc), w(0x5d), w(0x65), w(0xb6), w(0x92),\
    w(0x6c), w(0x70), w(0x48), w(0x50), w(0xfd), w(0xed), w(0xb9), w(0xda),\
    w(0x5e), w(0x15), w(0x46), w(0x57), w(0xa7), w(0x8d), w(0x9d), w(0x84),\
    w(0x90), w(0xd8), w(0xab), w(0x00), w(0x8c), w(0xbc), w(0xd3), w(0x0a),\
    w(0xf7), w(0xe4), w(0x58), w(0x05), w(0xb8), w(0xb3), w(0x45), w(0x06),\
    w(0xd0), w(0x2c), w(0x1e), w(0x8f), w(0xca), w(0x3f), w(0x0f), w(0x02),\
    w(0xc1), w(0xaf), w(0xbd), w(0x03), w(0x01), w(0x13), w(0x8a), w(0x6b),\
    w(0x3a), w(0x91), w(0x11), w(0x41), w(0x4f), w(0x67), w(0xdc), w(0xea),\
    w(0x97), w(0xf2), w(0xcf), w(0xce), w(0xf0), w(0xb4), w(0xe6), w(0x73),\
    w(0x96), w(0xac), w(0x74), w(0x22), w(0xe7), w(0xad), w(0x35), w(0x85),\
    w(0xe2), w(0xf9), w(0x37), w(0xe8), w(0x1c), w(0x75), w(0xdf), w(0x6e),\
    w(0x47), w(0xf1), w(0x1a), w(0x71), w(0x1d), w(0x29), w(0xc5), w(0x89),\
    w(0x6f), w(0xb7), w(0x62), w(0x0e), w(0xaa), w(0x18), w(0xbe), w(0x1b),\
    w(0xfc), w(0x56), w(0x3e), w(0x4b), w(0xc6), w(0xd2), w(0x79), w(0x20),\
    w(0x9a), w(0xdb), w(0xc0), w(0xfe), w(0x78), w(0xcd), w(0x5a), w(0xf4),\
    w(0x1f), w(0xdd), w(0xa8), w(0x33), w(0x88), w(0x07), w(0xc7), w(0x31),\
    w(0xb1), w(0x12), w(0x10), w(0x59), w(0x27), w(0x80), w(0xec), w(0x5f),\
    w(0x60), w(0x51), w(0x7f), w(0xa9), w(0x19), w(0xb5), w(0x4a), w(0x0d),\
    w(0x2d), w(0xe5), w(0x7a), w(0x9f), w(0x93), w(0xc9), w(0x9c), w(0xef),\
    w(0xa0), w(0xe0), w(0x3b), w(0x4d), w(0xae), w(0x2a), w(0xf5), w(0xb0),\
    w(0xc8), w(0xeb), w(0xbb), w(0x3c), w(0x83), w(0x53), w(0x99), w(0x61),\
    w(0x17), w(0x2b), w(0x04), w(0x7e), w(0xba), w(0x77), w(0xd6), w(0x26),\
    w(0xe1), w(0x69), w(0x14), w(0x63), w(0x55), w(0x21), w(0x0c), w(0x7d) }

#define mm_data(w) {     /* basic data for forming finite field tables */   \
    w(0x00), w(0x01), w(0x02), w(0x03), w(0x04), w(0x05), w(0x06), w(0x07),\
    w(0x08), w(0x09), w(0x0a), w(0x0b), w(0x0c), w(0x0d), w(0x0e), w(0x0f),\
    w(0x10), w(0x11), w(0x12), w(0x13), w(0x14), w(0x15), w(0x16), w(0x17),\
    w(0x18), w(0x19), w(0x1a), w(0x1b), w(0x1c), w(0x1d), w(0x1e), w(0x1f),\
    w(0x20), w(0x21), w(0x22), w(0x23), w(0x24), w(0x25), w(0x26), w(0x27),\
    w(0x28), w(0x29), w(0x2a), w(0x2b), w(0x2c), w(0x2d), w(0x2e), w(0x2f),\
    w(0x30), w(0x31), w(0x32), w(0x33), w(0x34), w(0x35), w(0x36), w(0x37),\
    w(0x38), w(0x39), w(0x3a), w(0x3b), w(0x3c), w(0x3d), w(0x3e), w(0x3f),\
    w(0x40), w(0x41), w(0x42), w(0x43), w(0x44), w(0x45), w(0x46), w(0x47),\
    w(0x48), w(0x49), w(0x4a), w(0x4b), w(0x4c), w(0x4d), w(0x4e), w(0x4f),\
    w(0x50), w(0x51), w(0x52), w(0x53), w(0x54), w(0x55), w(0x56), w(0x57),\
    w(0x58), w(0x59), w(0x5a), w(0x5b), w(0x5c), w(0x5d), w(0x5e), w(0x5f),\
    w(0x60), w(0x61), w(0x62), w(0x63), w(0x64), w(0x65), w(0x66), w(0x67),\
    w(0x68), w(0x69), w(0x6a), w(0x6b), w(0x6c), w(0x6d), w(0x6e), w(0x6f),\
    w(0x70), w(0x71), w(0x72), w(0x73), w(0x74), w(0x75), w(0x76), w(0x77),\
    w(0x78), w(0x79), w(0x7a), w(0x7b), w(0x7c), w(0x7d), w(0x7e), w(0x7f),\
    w(0x80), w(0x81), w(0x82), w(0x83), w(0x84), w(0x85), w(0x86), w(0x87),\
    w(0x88), w(0x89), w(0x8a), w(0x8b), w(0x8c), w(0x8d), w(0x8e), w(0x8f),\
    w(0x90), w(0x91), w(0x92), w(0x93), w(0x94), w(0x95), w(0x96), w(0x97),\
    w(0x98), w(0x99), w(0x9a), w(0x9b), w(0x9c), w(0x9d), w(0x9e), w(0x9f),\
    w(0xa0), w(0xa1), w(0xa2), w(0xa3), w(0xa4), w(0xa5), w(0xa6), w(0xa7),\
    w(0xa8), w(0xa9), w(0xaa), w(0xab), w(0xac), w(0xad), w(0xae), w(0xaf),\
    w(0xb0), w(0xb1), w(0xb2), w(0xb3), w(0xb4), w(0xb5), w(0xb6), w(0xb7),\
    w(0xb8), w(0xb9), w(0xba), w(0xbb), w(0xbc), w(0xbd), w(0xbe), w(0xbf),\
    w(0xc0), w(0xc1), w(0xc2), w(0xc3), w(0xc4), w(0xc5), w(0xc6), w(0xc7),\
    w(0xc8), w(0xc9), w(0xca), w(0xcb), w(0xcc), w(0xcd), w(0xce), w(0xcf),\
    w(0xd0), w(0xd1), w(0xd2), w(0xd3), w(0xd4), w(0xd5), w(0xd6), w(0xd7),\
    w(0xd8), w(0xd9), w(0xda), w(0xdb), w(0xdc), w(0xdd), w(0xde), w(0xdf),\
    w(0xe0), w(0xe1), w(0xe2), w(0xe3), w(0xe4), w(0xe5), w(0xe6), w(0xe7),\
    w(0xe8), w(0xe9), w(0xea), w(0xeb), w(0xec), w(0xed), w(0xee), w(0xef),\
    w(0xf0), w(0xf1), w(0xf2), w(0xf3), w(0xf4), w(0xf5), w(0xf6), w(0xf7),\
    w(0xf8), w(0xf9), w(0xfa), w(0xfb), w(0xfc), w(0xfd), w(0xfe), w(0xff) }

static const uint8_t sbox[256]  =  sb_data(f1);

#if defined( AES_DEC_PREKEYED )
static const uint8_t isbox[256] = isb_data(f1);
#endif

static const uint8_t gfm2_sbox[256] = sb_data(f2);
static const uint8_t gfm3_sbox[256] = sb_data(f3);

#if defined( AES_DEC_PREKEYED )
static const uint8_t gfmul_9[256] = mm_data(f9);
static const uint8_t gfmul_b[256] = mm_data(fb);
static const uint8_t gfmul_d[256] = mm_data(fd);
```

```
static const uint8_t gfmul_e[256] = mm_data(fe);
#endif

#define s_box(x)        sbox[(x)]
#if defined( AES_DEC_PREKEYED )
#define is_box(x)       isbox[(x)]
#endif
#define gfm2_sb(x)      gfm2_sbox[(x)]
#define gfm3_sb(x)      gfm3_sbox[(x)]
#if defined( AES_DEC_PREKEYED )
#define gfm_9(x)        gfmul_9[(x)]
#define gfm_b(x)        gfmul_b[(x)]
#define gfm_d(x)        gfmul_d[(x)]
#define gfm_e(x)        gfmul_e[(x)]
#endif
#else

/* this is the high bit of x right shifted by 1 */
/* position. Since the starting polynomial has  */
/* 9 bits (0x11b), this right shift keeps the    */
/* values of all top bits within a byte          */

static uint8_t hibit(const uint8_t x)
{   uint8_t r = (uint8_t)((x >> 1) | (x >> 2));

    r |= (r >> 2);
    r |= (r >> 4);
    return (r + 1) >> 1;
}

/* return the inverse of the finite field element x */

static uint8_t gf_inv(const uint8_t x)
{   uint8_t p1 = x, p2 = BPOLY, n1 = hibit(x), n2 = 0x80, v1 = 1, v2 = 0;

    if(x < 2)
        return x;

    for( ; ; )
    {
        if(n1)
            while(n2 >= n1)                  /* divide polynomial p2 by p1    */
            {
                n2 /= n1;                     /* shift smaller polynomial left */
                p2 ^= (p1 * n2) & 0xff; /* and remove from larger one     */
                v2 ^= (v1 * n2);              /* shift accumulated value and   */
                n2 = hibit(p2);               /* add into result               */
            }
        else
            return v1;

        if(n2)                               /* repeat with values swapped     */
            while(n1 >= n2)
            {
                n1 /= n2;
                p1 ^= p2 * n1;
                v1 ^= v2 * n1;
                n1 = hibit(p1);
            }
        else
            return v2;
    }
}

/* The forward and inverse affine transformations used in the S-box */
uint8_t fwd_affine(const uint8_t x)
{
#if defined( HAVE_UINT_32T )
    uint32_t w = x;
    w ^= (w << 1) ^ (w << 2) ^ (w << 3) ^ (w << 4);
    return 0x63 ^ ((w ^ (w >> 8)) & 0xff);
#else
    return 0x63 ^ x ^ (x << 1) ^ (x << 2) ^ (x << 3) ^ (x << 4)
                    ^ (x >> 7) ^ (x >> 6) ^ (x >> 5) ^ (x >> 4);
#endif
}
```

```c
uint8_t inv_affine(const uint8_t x)
{
#if defined( HAVE_UINT_32T )
    uint32_t w = x;
    w = (w << 1) ^ (w << 3) ^ (w << 6);
    return 0x05 ^ ((w ^ (w >> 8)) & 0xff);
#else
    return 0x05 ^ (x << 1) ^ (x << 3) ^ (x << 6)
                ^ (x >> 7) ^ (x >> 5) ^ (x >> 2);
#endif
}

#define s_box(x)    fwd_affine(gf_inv(x))
#define is_box(x)   gf_inv(inv_affine(x))
#define gfm2_sb(x)  f2(s_box(x))
#define gfm3_sb(x)  f3(s_box(x))
#define gfm_9(x)    f9(x)
#define gfm_b(x)    fb(x)
#define gfm_d(x)    fd(x)
#define gfm_e(x)    fe(x)

#endif

#if defined( HAVE_MEMCPY )
#  define block_copy_nn(d, s, l)    memcpy(d, s, l)
#  define block_copy(d, s)          memcpy(d, s, N_BLOCK)
#else
#  define block_copy_nn(d, s, l)    copy_block_nn(d, s, l)
#  define block_copy(d, s)          copy_block(d, s)
#endif

static void copy_block( void *d, const void *s )
{
#if defined( HAVE_UINT_32T )
    ((uint32_t*)d)[ 0] = ((uint32_t*)s)[ 0];
    ((uint32_t*)d)[ 1] = ((uint32_t*)s)[ 1];
    ((uint32_t*)d)[ 2] = ((uint32_t*)s)[ 2];
    ((uint32_t*)d)[ 3] = ((uint32_t*)s)[ 3];
#else
    ((uint8_t*)d)[ 0] = ((uint8_t*)s)[ 0];
    ((uint8_t*)d)[ 1] = ((uint8_t*)s)[ 1];
    ((uint8_t*)d)[ 2] = ((uint8_t*)s)[ 2];
    ((uint8_t*)d)[ 3] = ((uint8_t*)s)[ 3];
    ((uint8_t*)d)[ 4] = ((uint8_t*)s)[ 4];
    ((uint8_t*)d)[ 5] = ((uint8_t*)s)[ 5];
    ((uint8_t*)d)[ 6] = ((uint8_t*)s)[ 6];
    ((uint8_t*)d)[ 7] = ((uint8_t*)s)[ 7];
    ((uint8_t*)d)[ 8] = ((uint8_t*)s)[ 8];
    ((uint8_t*)d)[ 9] = ((uint8_t*)s)[ 9];
    ((uint8_t*)d)[10] = ((uint8_t*)s)[10];
    ((uint8_t*)d)[11] = ((uint8_t*)s)[11];
    ((uint8_t*)d)[12] = ((uint8_t*)s)[12];
    ((uint8_t*)d)[13] = ((uint8_t*)s)[13];
    ((uint8_t*)d)[14] = ((uint8_t*)s)[14];
    ((uint8_t*)d)[15] = ((uint8_t*)s)[15];
#endif
}

static void copy_block_nn( uint8_t * d, const uint8_t *s, uint8_t nn )
{
    while( nn-- )
        //*((uint8_t*)d)++ = *((uint8_t*)s)++;
        *d++ = *s++;
}

static void xor_block( void *d, const void *s )
{
#if defined( HAVE_UINT_32T )
    ((uint32_t*)d)[ 0] ^= ((uint32_t*)s)[ 0];
    ((uint32_t*)d)[ 1] ^= ((uint32_t*)s)[ 1];
    ((uint32_t*)d)[ 2] ^= ((uint32_t*)s)[ 2];
    ((uint32_t*)d)[ 3] ^= ((uint32_t*)s)[ 3];
#else
    ((uint8_t*)d)[ 0] ^= ((uint8_t*)s)[ 0];
    ((uint8_t*)d)[ 1] ^= ((uint8_t*)s)[ 1];
```

```c
    ((uint8_t*)d)[ 2] ^= ((uint8_t*)s)[ 2];
    ((uint8_t*)d)[ 3] ^= ((uint8_t*)s)[ 3];
    ((uint8_t*)d)[ 4] ^= ((uint8_t*)s)[ 4];
    ((uint8_t*)d)[ 5] ^= ((uint8_t*)s)[ 5];
    ((uint8_t*)d)[ 6] ^= ((uint8_t*)s)[ 6];
    ((uint8_t*)d)[ 7] ^= ((uint8_t*)s)[ 7];
    ((uint8_t*)d)[ 8] ^= ((uint8_t*)s)[ 8];
    ((uint8_t*)d)[ 9] ^= ((uint8_t*)s)[ 9];
    ((uint8_t*)d)[10] ^= ((uint8_t*)s)[10];
    ((uint8_t*)d)[11] ^= ((uint8_t*)s)[11];
    ((uint8_t*)d)[12] ^= ((uint8_t*)s)[12];
    ((uint8_t*)d)[13] ^= ((uint8_t*)s)[13];
    ((uint8_t*)d)[14] ^= ((uint8_t*)s)[14];
    ((uint8_t*)d)[15] ^= ((uint8_t*)s)[15];
#endif
}

static void copy_and_key( void *d, const void *s, const void *k )
{
#if defined( HAVE_UINT_32T )
    ((uint32_t*)d)[ 0] = ((uint32_t*)s)[ 0] ^ ((uint32_t*)k)[ 0];
    ((uint32_t*)d)[ 1] = ((uint32_t*)s)[ 1] ^ ((uint32_t*)k)[ 1];
    ((uint32_t*)d)[ 2] = ((uint32_t*)s)[ 2] ^ ((uint32_t*)k)[ 2];
    ((uint32_t*)d)[ 3] = ((uint32_t*)s)[ 3] ^ ((uint32_t*)k)[ 3];
#elif 1
    ((uint8_t*)d)[ 0] = ((uint8_t*)s)[ 0] ^ ((uint8_t*)k)[ 0];
    ((uint8_t*)d)[ 1] = ((uint8_t*)s)[ 1] ^ ((uint8_t*)k)[ 1];
    ((uint8_t*)d)[ 2] = ((uint8_t*)s)[ 2] ^ ((uint8_t*)k)[ 2];
    ((uint8_t*)d)[ 3] = ((uint8_t*)s)[ 3] ^ ((uint8_t*)k)[ 3];
    ((uint8_t*)d)[ 4] = ((uint8_t*)s)[ 4] ^ ((uint8_t*)k)[ 4];
    ((uint8_t*)d)[ 5] = ((uint8_t*)s)[ 5] ^ ((uint8_t*)k)[ 5];
    ((uint8_t*)d)[ 6] = ((uint8_t*)s)[ 6] ^ ((uint8_t*)k)[ 6];
    ((uint8_t*)d)[ 7] = ((uint8_t*)s)[ 7] ^ ((uint8_t*)k)[ 7];
    ((uint8_t*)d)[ 8] = ((uint8_t*)s)[ 8] ^ ((uint8_t*)k)[ 8];
    ((uint8_t*)d)[ 9] = ((uint8_t*)s)[ 9] ^ ((uint8_t*)k)[ 9];
    ((uint8_t*)d)[10] = ((uint8_t*)s)[10] ^ ((uint8_t*)k)[10];
    ((uint8_t*)d)[11] = ((uint8_t*)s)[11] ^ ((uint8_t*)k)[11];
    ((uint8_t*)d)[12] = ((uint8_t*)s)[12] ^ ((uint8_t*)k)[12];
    ((uint8_t*)d)[13] = ((uint8_t*)s)[13] ^ ((uint8_t*)k)[13];
    ((uint8_t*)d)[14] = ((uint8_t*)s)[14] ^ ((uint8_t*)k)[14];
    ((uint8_t*)d)[15] = ((uint8_t*)s)[15] ^ ((uint8_t*)k)[15];
#else
    block_copy(d, s);
    xor_block(d, k);
#endif
}

static void add_round_key( uint8_t d[N_BLOCK], const uint8_t k[N_BLOCK] )
{
    xor_block(d, k);
}

static void shift_sub_rows( uint8_t st[N_BLOCK] )
{   uint8_t tt;

    st[ 0] = s_box(st[ 0]); st[ 4] = s_box(st[ 4]);
    st[ 8] = s_box(st[ 8]); st[12] = s_box(st[12]);

    tt = st[1]; st[ 1] = s_box(st[ 5]); st[ 5] = s_box(st[ 9]);
    st[ 9] = s_box(st[13]); st[13] = s_box( tt );

    tt = st[2]; st[ 2] = s_box(st[10]); st[10] = s_box( tt );
    tt = st[6]; st[ 6] = s_box(st[14]); st[14] = s_box( tt );

    tt = st[15]; st[15] = s_box(st[11]); st[11] = s_box(st[ 7]);
    st[ 7] = s_box(st[ 3]); st[ 3] = s_box( tt );
}

#if defined( AES_DEC_PREKEYED )

static void inv_shift_sub_rows( uint8_t st[N_BLOCK] )
{   uint8_t tt;

    st[ 0] = is_box(st[ 0]); st[ 4] = is_box(st[ 4]);
    st[ 8] = is_box(st[ 8]); st[12] = is_box(st[12]);
```

```
      tt = st[13]; st[13] = is_box(st[9]); st[ 9] = is_box(st[5]);
      st[ 5] = is_box(st[1]); st[ 1] = is_box( tt );

      tt = st[2]; st[ 2] = is_box(st[10]); st[10] = is_box( tt );
      tt = st[6]; st[ 6] = is_box(st[14]); st[14] = is_box( tt );

      tt = st[3]; st[ 3] = is_box(st[ 7]); st[ 7] = is_box(st[11]);
      st[11] = is_box(st[15]); st[15] = is_box( tt );
}

#endif


#if defined( VERSION_1 )
  static void mix_sub_columns( uint8_t dt[N_BLOCK] )
  { uint8_t st[N_BLOCK];
    block_copy(st, dt);
#else
  static void mix_sub_columns( uint8_t dt[N_BLOCK], uint8_t st[N_BLOCK] )
  {
#endif
    dt[ 0] = gfm2_sb(st[0]) ^ gfm3_sb(st[5]) ^ s_box(st[10]) ^ s_box(st[15]);
    dt[ 1] = s_box(st[0]) ^ gfm2_sb(st[5]) ^ gfm3_sb(st[10]) ^ s_box(st[15]);
    dt[ 2] = s_box(st[0]) ^ s_box(st[5]) ^ gfm2_sb(st[10]) ^ gfm3_sb(st[15]);
    dt[ 3] = gfm3_sb(st[0]) ^ s_box(st[5]) ^ s_box(st[10]) ^ gfm2_sb(st[15]);

    dt[ 4] = gfm2_sb(st[4]) ^ gfm3_sb(st[9]) ^ s_box(st[14]) ^ s_box(st[3]);
    dt[ 5] = s_box(st[4]) ^ gfm2_sb(st[9]) ^ gfm3_sb(st[14]) ^ s_box(st[3]);
    dt[ 6] = s_box(st[4]) ^ s_box(st[9]) ^ gfm2_sb(st[14]) ^ gfm3_sb(st[3]);
    dt[ 7] = gfm3_sb(st[4]) ^ s_box(st[9]) ^ s_box(st[14]) ^ gfm2_sb(st[3]);

    dt[ 8] = gfm2_sb(st[8]) ^ gfm3_sb(st[13]) ^ s_box(st[2]) ^ s_box(st[7]);
    dt[ 9] = s_box(st[8]) ^ gfm2_sb(st[13]) ^ gfm3_sb(st[2]) ^ s_box(st[7]);
    dt[10] = s_box(st[8]) ^ s_box(st[13]) ^ gfm2_sb(st[2]) ^ gfm3_sb(st[7]);
    dt[11] = gfm3_sb(st[8]) ^ s_box(st[13]) ^ s_box(st[2]) ^ gfm2_sb(st[7]);

    dt[12] = gfm2_sb(st[12]) ^ gfm3_sb(st[1]) ^ s_box(st[6]) ^ s_box(st[11]);
    dt[13] = s_box(st[12]) ^ gfm2_sb(st[1]) ^ gfm3_sb(st[6]) ^ s_box(st[11]);
    dt[14] = s_box(st[12]) ^ s_box(st[1]) ^ gfm2_sb(st[6]) ^ gfm3_sb(st[11]);
    dt[15] = gfm3_sb(st[12]) ^ s_box(st[1]) ^ s_box(st[6]) ^ gfm2_sb(st[11]);
  }


#if defined( AES_DEC_PREKEYED )

#if defined( VERSION_1 )
  static void inv_mix_sub_columns( uint8_t dt[N_BLOCK] )
  { uint8_t st[N_BLOCK];
    block_copy(st, dt);
#else
  static void inv_mix_sub_columns( uint8_t dt[N_BLOCK], uint8_t st[N_BLOCK] )
  {
#endif
    dt[ 0] = is_box(gfm_e(st[ 0]) ^ gfm_b(st[ 1]) ^ gfm_d(st[ 2]) ^ gfm_9(st[ 3]));
    dt[ 5] = is_box(gfm_9(st[ 0]) ^ gfm_e(st[ 1]) ^ gfm_b(st[ 2]) ^ gfm_d(st[ 3]));
    dt[10] = is_box(gfm_d(st[ 0]) ^ gfm_9(st[ 1]) ^ gfm_e(st[ 2]) ^ gfm_b(st[ 3]));
    dt[15] = is_box(gfm_b(st[ 0]) ^ gfm_d(st[ 1]) ^ gfm_9(st[ 2]) ^ gfm_e(st[ 3]));

    dt[ 4] = is_box(gfm_e(st[ 4]) ^ gfm_b(st[ 5]) ^ gfm_d(st[ 6]) ^ gfm_9(st[ 7]));
    dt[ 9] = is_box(gfm_9(st[ 4]) ^ gfm_e(st[ 5]) ^ gfm_b(st[ 6]) ^ gfm_d(st[ 7]));
    dt[14] = is_box(gfm_d(st[ 4]) ^ gfm_9(st[ 5]) ^ gfm_e(st[ 6]) ^ gfm_b(st[ 7]));
    dt[ 3] = is_box(gfm_b(st[ 4]) ^ gfm_d(st[ 5]) ^ gfm_9(st[ 6]) ^ gfm_e(st[ 7]));

    dt[ 8] = is_box(gfm_e(st[ 8]) ^ gfm_b(st[ 9]) ^ gfm_d(st[10]) ^ gfm_9(st[11]));
    dt[13] = is_box(gfm_9(st[ 8]) ^ gfm_e(st[ 9]) ^ gfm_b(st[10]) ^ gfm_d(st[11]));
    dt[ 2] = is_box(gfm_d(st[ 8]) ^ gfm_9(st[ 9]) ^ gfm_e(st[10]) ^ gfm_b(st[11]));
    dt[ 7] = is_box(gfm_b(st[ 8]) ^ gfm_d(st[ 9]) ^ gfm_9(st[10]) ^ gfm_e(st[11]));

    dt[12] = is_box(gfm_e(st[12]) ^ gfm_b(st[13]) ^ gfm_d(st[14]) ^ gfm_9(st[15]));
    dt[ 1] = is_box(gfm_9(st[12]) ^ gfm_e(st[13]) ^ gfm_b(st[14]) ^ gfm_d(st[15]));
    dt[ 6] = is_box(gfm_d(st[12]) ^ gfm_9(st[13]) ^ gfm_e(st[14]) ^ gfm_b(st[15]));
    dt[11] = is_box(gfm_b(st[12]) ^ gfm_d(st[13]) ^ gfm_9(st[14]) ^ gfm_e(st[15]));
  }

#endif

#if defined( AES_ENC_PREKEYED ) || defined( AES_DEC_PREKEYED )
```

```c
/*  Set the cipher key for the pre-keyed version */

return_type aes_set_key( const uint8_t key[], length_type keylen, aes_context ctx[1] )
{
    uint8_t cc, rc, hi;

    switch( keylen )
    {
    case 16:
    case 24:
    case 32:
        break;
    default:
        ctx->rnd = 0;
        return ( uint8_t )-1;
    }
    block_copy_nn(ctx->ksch, key, keylen);
    hi = (keylen + 28) << 2;
    ctx->rnd = (hi >> 4) - 1;
    for( cc = keylen, rc = 1; cc < hi; cc += 4 )
    {   uint8_t tt, t0, t1, t2, t3;

        t0 = ctx->ksch[cc - 4];
        t1 = ctx->ksch[cc - 3];
        t2 = ctx->ksch[cc - 2];
        t3 = ctx->ksch[cc - 1];
        if( cc % keylen == 0 )
        {
            tt = t0;
            t0 = s_box(t1) ^ rc;
            t1 = s_box(t2);
            t2 = s_box(t3);
            t3 = s_box(tt);
            rc = f2(rc);
        }
        else if( keylen > 24 && cc % keylen == 16 )
        {
            t0 = s_box(t0);
            t1 = s_box(t1);
            t2 = s_box(t2);
            t3 = s_box(t3);
        }
        tt = cc - keylen;
        ctx->ksch[cc + 0] = ctx->ksch[tt + 0] ^ t0;
        ctx->ksch[cc + 1] = ctx->ksch[tt + 1] ^ t1;
        ctx->ksch[cc + 2] = ctx->ksch[tt + 2] ^ t2;
        ctx->ksch[cc + 3] = ctx->ksch[tt + 3] ^ t3;
    }
    return 0;
}

#endif

#if defined( AES_ENC_PREKEYED )

/*  Encrypt a single block of 16 bytes */

return_type aes_encrypt( const uint8_t in[N_BLOCK], uint8_t  out[N_BLOCK], const aes_context
ctx[1] )
{
    if( ctx->rnd )
    {
        uint8_t s1[N_BLOCK], r;
        copy_and_key( s1, in, ctx->ksch );

        for( r = 1 ; r < ctx->rnd ; ++r )
#if defined( VERSION_1 )
        {
            mix_sub_columns( s1 );
            add_round_key( s1, ctx->ksch + r * N_BLOCK);
        }
#else
        {   uint8_t s2[N_BLOCK];
            mix_sub_columns( s2, s1 );
            copy_and_key( s1, s2, ctx->ksch + r * N_BLOCK);
        }
```

```
#endif
        shift_sub_rows( s1 );
        copy_and_key( out, s1, ctx->ksch + r * N_BLOCK );
    }
    else
        return ( uint8_t )-1;
    return 0;
}

/* CBC encrypt a number of blocks (input and return an IV) */

return_type aes_cbc_encrypt( const uint8_t *in, uint8_t *out,
                    int32_t n_block, uint8_t iv[N_BLOCK], const aes_context ctx[1] )
{

    while(n_block--)
    {
        xor_block(iv, in);
        if(aes_encrypt(iv, iv, ctx) != EXIT_SUCCESS)
            return EXIT_FAILURE;
        //memcpy(out, iv, N_BLOCK);
        block_copy(out, iv);
        in += N_BLOCK;
        out += N_BLOCK;
    }
    return EXIT_SUCCESS;
}

#endif

#if defined( AES_DEC_PREKEYED )

/*  Decrypt a single block of 16 bytes */

return_type aes_decrypt( const uint8_t in[N_BLOCK], uint8_t out[N_BLOCK], const aes_context ctx[1] )
{
    if( ctx->rnd )
    {
        uint8_t s1[N_BLOCK], r;
        copy_and_key( s1, in, ctx->ksch + ctx->rnd * N_BLOCK );
        inv_shift_sub_rows( s1 );

        for( r = ctx->rnd ; --r ; )
#if defined( VERSION_1 )
        {
            add_round_key( s1, ctx->ksch + r * N_BLOCK );
            inv_mix_sub_columns( s1 );
        }
#else
        {   uint8_t s2[N_BLOCK];
            copy_and_key( s2, s1, ctx->ksch + r * N_BLOCK );
            inv_mix_sub_columns( s1, s2 );
        }
#endif
        copy_and_key( out, s1, ctx->ksch );
    }
    else
        return -1;
    return 0;
}

/* CBC decrypt a number of blocks (input and return an IV) */

return_type aes_cbc_decrypt( const uint8_t *in, uint8_t *out,
                    int32_t n_block, uint8_t iv[N_BLOCK], const aes_context ctx[1] )
{
    while(n_block--)
    {   uint8_t tmp[N_BLOCK];

        //memcpy(tmp, in, N_BLOCK);
        block_copy(tmp, in);
        if(aes_decrypt(in, out, ctx) != EXIT_SUCCESS)
            return EXIT_FAILURE;
        xor_block(out, iv);
        //memcpy(iv, tmp, N_BLOCK);
        block_copy(iv, tmp);
```

```
        in += N_BLOCK;
        out += N_BLOCK;
    }
    return EXIT_SUCCESS;
}

#endif

#if defined( AES_ENC_128_OTFK )

/*  The 'on the fly' encryption key update for for 128 bit keys */

static void update_encrypt_key_128( uint8_t k[N_BLOCK], uint8_t *rc )
{   uint8_t cc;

    k[0] ^= s_box(k[13]) ^ *rc;
    k[1] ^= s_box(k[14]);
    k[2] ^= s_box(k[15]);
    k[3] ^= s_box(k[12]);
    *rc = f2( *rc );

    for(cc = 4; cc < 16; cc += 4 )
    {
        k[cc + 0] ^= k[cc - 4];
        k[cc + 1] ^= k[cc - 3];
        k[cc + 2] ^= k[cc - 2];
        k[cc + 3] ^= k[cc - 1];
    }
}

/*  Encrypt a single block of 16 bytes with 'on the fly' 128 bit keying */

void aes_encrypt_128( const uint8_t in[N_BLOCK], uint8_t out[N_BLOCK],
                      const uint8_t key[N_BLOCK], uint8_t o_key[N_BLOCK] )
{   uint8_t s1[N_BLOCK], r, rc = 1;

    if(o_key != key)
        block_copy( o_key, key );
    copy_and_key( s1, in, o_key );

    for( r = 1 ; r < 10 ; ++r )
#if defined( VERSION_1 )
    {
        mix_sub_columns( s1 );
        update_encrypt_key_128( o_key, &rc );
        add_round_key( s1, o_key );
    }
#else
    {   uint8_t s2[N_BLOCK];
        mix_sub_columns( s2, s1 );
        update_encrypt_key_128( o_key, &rc );
        copy_and_key( s1, s2, o_key );
    }
#endif

    shift_sub_rows( s1 );
    update_encrypt_key_128( o_key, &rc );
    copy_and_key( out, s1, o_key );
}

#endif

#if defined( AES_DEC_128_OTFK )

/*  The 'on the fly' decryption key update for for 128 bit keys */

static void update_decrypt_key_128( uint8_t k[N_BLOCK], uint8_t *rc )
{   uint8_t cc;

    for( cc = 12; cc > 0; cc -= 4 )
    {
        k[cc + 0] ^= k[cc - 4];
        k[cc + 1] ^= k[cc - 3];
        k[cc + 2] ^= k[cc - 2];
        k[cc + 3] ^= k[cc - 1];
    }
```

```
        *rc = d2(*rc);
        k[0] ^= s_box(k[13]) ^ *rc;
        k[1] ^= s_box(k[14]);
        k[2] ^= s_box(k[15]);
        k[3] ^= s_box(k[12]);
    }

/*  Decrypt a single block of 16 bytes with 'on the fly' 128 bit keying */

void aes_decrypt_128( const uint8_t in[N_BLOCK], uint8_t out[N_BLOCK],
                        const uint8_t key[N_BLOCK], uint8_t o_key[N_BLOCK] )
{
    uint8_t s1[N_BLOCK], r, rc = 0x6c;
    if(o_key != key)
        block_copy( o_key, key );

    copy_and_key( s1, in, o_key );
    inv_shift_sub_rows( s1 );

    for( r = 10 ; --r ; )
#if defined( VERSION_1 )
    {
        update_decrypt_key_128( o_key, &rc );
        add_round_key( s1, o_key );
        inv_mix_sub_columns( s1 );
    }
#else
    {   uint8_t s2[N_BLOCK];
        update_decrypt_key_128( o_key, &rc );
        copy_and_key( s2, s1, o_key );
        inv_mix_sub_columns( s1, s2 );
    }
#endif
    update_decrypt_key_128( o_key, &rc );
    copy_and_key( out, s1, o_key );
}

#endif

#if defined( AES_ENC_256_OTFK )

/*  The 'on the fly' encryption key update for for 256 bit keys */

static void update_encrypt_key_256( uint8_t k[2 * N_BLOCK], uint8_t *rc )
{   uint8_t cc;

    k[0] ^= s_box(k[29]) ^ *rc;
    k[1] ^= s_box(k[30]);
    k[2] ^= s_box(k[31]);
    k[3] ^= s_box(k[28]);
    *rc = f2( *rc );

    for(cc = 4; cc < 16; cc += 4)
    {
        k[cc + 0] ^= k[cc - 4];
        k[cc + 1] ^= k[cc - 3];
        k[cc + 2] ^= k[cc - 2];
        k[cc + 3] ^= k[cc - 1];
    }

    k[16] ^= s_box(k[12]);
    k[17] ^= s_box(k[13]);
    k[18] ^= s_box(k[14]);
    k[19] ^= s_box(k[15]);

    for( cc = 20; cc < 32; cc += 4 )
    {
        k[cc + 0] ^= k[cc - 4];
        k[cc + 1] ^= k[cc - 3];
        k[cc + 2] ^= k[cc - 2];
        k[cc + 3] ^= k[cc - 1];
    }
}

/*  Encrypt a single block of 16 bytes with 'on the fly' 256 bit keying */
```

```c
void aes_encrypt_256( const uint8_t in[N_BLOCK], uint8_t out[N_BLOCK],
                      const uint8_t key[2 * N_BLOCK], uint8_t o_key[2 * N_BLOCK] )
{
    uint8_t s1[N_BLOCK], r, rc = 1;
    if(o_key != key)
    {
        block_copy( o_key, key );
        block_copy( o_key + 16, key + 16 );
    }
    copy_and_key( s1, in, o_key );

    for( r = 1 ; r < 14 ; ++r )
#if defined( VERSION_1 )
    {
        mix_sub_columns(s1);
        if( r & 1 )
            add_round_key( s1, o_key + 16 );
        else
        {
            update_encrypt_key_256( o_key, &rc );
            add_round_key( s1, o_key );
        }
    }
#else
    {   uint8_t s2[N_BLOCK];
        mix_sub_columns( s2, s1 );
        if( r & 1 )
            copy_and_key( s1, s2, o_key + 16 );
        else
        {
            update_encrypt_key_256( o_key, &rc );
            copy_and_key( s1, s2, o_key );
        }
    }
#endif

    shift_sub_rows( s1 );
    update_encrypt_key_256( o_key, &rc );
    copy_and_key( out, s1, o_key );
}

#endif

#if defined( AES_DEC_256_OTFK )

/*  The 'on the fly' encryption key update for for 256 bit keys */

static void update_decrypt_key_256( uint8_t k[2 * N_BLOCK], uint8_t *rc )
{   uint8_t cc;

    for(cc = 28; cc > 16; cc -= 4)
    {
        k[cc + 0] ^= k[cc - 4];
        k[cc + 1] ^= k[cc - 3];
        k[cc + 2] ^= k[cc - 2];
        k[cc + 3] ^= k[cc - 1];
    }

    k[16] ^= s_box(k[12]);
    k[17] ^= s_box(k[13]);
    k[18] ^= s_box(k[14]);
    k[19] ^= s_box(k[15]);

    for(cc = 12; cc > 0; cc -= 4)
    {
        k[cc + 0] ^= k[cc - 4];
        k[cc + 1] ^= k[cc - 3];
        k[cc + 2] ^= k[cc - 2];
        k[cc + 3] ^= k[cc - 1];
    }

    *rc = d2(*rc);
    k[0] ^= s_box(k[29]) ^ *rc;
    k[1] ^= s_box(k[30]);
    k[2] ^= s_box(k[31]);
    k[3] ^= s_box(k[28]);
```

```
}

/*  Decrypt a single block of 16 bytes with 'on the fly'
    256 bit keying
*/
void aes_decrypt_256( const uint8_t in[N_BLOCK], uint8_t out[N_BLOCK],
                      const uint8_t key[2 * N_BLOCK], uint8_t o_key[2 * N_BLOCK] )
{
    uint8_t s1[N_BLOCK], r, rc = 0x80;

    if(o_key != key)
    {
        block_copy( o_key, key );
        block_copy( o_key + 16, key + 16 );
    }

    copy_and_key( s1, in, o_key );
    inv_shift_sub_rows( s1 );

    for( r = 14 ; --r ; )
#if defined( VERSION_1 )
    {
        if( ( r & 1 ) )
        {
            update_decrypt_key_256( o_key, &rc );
            add_round_key( s1, o_key + 16 );
        }
        else
            add_round_key( s1, o_key );
        inv_mix_sub_columns( s1 );
    }
#else
    {   uint8_t s2[N_BLOCK];
        if( ( r & 1 ) )
        {
            update_decrypt_key_256( o_key, &rc );
            copy_and_key( s2, s1, o_key + 16 );
        }
        else
            copy_and_key( s2, s1, o_key );
        inv_mix_sub_columns( s1, s2 );
    }
#endif
    copy_and_key( out, s1, o_key );
}

#endif
```

A.1.2 Our AES - test example on CubeCell board

```
#include "aes.h"
aes_context ctx[2];
const uint8_t in[16]={ 0xAB, 0xCD, 0xAC, 0xB8,0x8A, 0x77, 0xA6, 0xA6,0x8B, 0xC1, 0xD2, 0xF3,0xBB,
0xF1, 0xF2, 0xF3};
uint8_t out[16], out1[16];
const uint8_t key[16]={ 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68,0x69, 0x6A, 0x6B, 0x6C,0x6D,
0x6E, 0x6F, 0x70};

void setup(){
  Serial.begin(9600);delay(300);
  Serial.println("startcrypt");delay(300);
  aes_set_key(key,
              16,            // len in bytes - length_type
              ctx );
  for(int i=0; i<16;i++) Serial.print(in[i],HEX);
  Serial.println();delay(300);
  aes_encrypt(in,
              out1,
              ctx );
  for(int i=0; i<16;i++) Serial.print(out1[i],HEX);
  Serial.println();delay(300);
  aes_decrypt(out1,
              out,
              ctx );
for(int i=0; i<16;i++) Serial.print(out[i],HEX);
Serial.println();
}

void loop(){ delay(15000);}
```

Final remarks

Below you can see the execution results of the above presented examples hardware – ESP32 /software – CubeCell (ARM) coded

Execution result for CubeCell – software:

```
startcrypt
ABCDACB88A77A6A68BC1D2F3BBF1F2F3
E78C3A1E3D356A116AF49CC77EC773C
ABCDACB88A77A6A68BC1D2F3BBF1F2F3
```

Remark – the execution result for ESP32 (hardware accelerated):

```
ABCDACB88A77A6A68BC1D2F3BBF1F2F3
E78C3A1E3D356A116AF49CC77EC773C

ABCDACB88A77A6A68BC1D2F3BBF1F2F3
```

```
AES.CubeCell.   aes.cpp   aes.h

#include "aes.h"
aes_context ctx[2];
const uint8_t in[16]={ 0xAB, 0xCD, 0xAC, 0xB8,0x8A, 0x77, 0xA6, 0xA6,0x8B, 0xC1, 0xD2, 0xF3,0xBB, 0xF1, 0xF2, 0xF3};
uint8_t out[16], out1[16];
const uint8_t key[16]={ 0x00, 0x01, 0x02, 0x03,0x00, 0x01, 0x02, 0x03,0x00, 0x01, 0x02, 0x03,0x00, 0x01, 0x02, 0x03};

void setup(){
  Serial.begin(9600);delay(300);
  Serial.println("startcrypt");delay(300);
  aes_set_key(key,
              16,           // len in bytes - length_type
              ctx );
  for(int i=0; i<16;i++) Serial.print(in[i],HEX);
  Serial.println();delay(300);
  aes_encrypt(in,
              out1,
              ctx );
  for(int i=0; i<16;i++) Serial.print(out1[i],HEX);
  Serial.println();delay(300);
  aes_decrypt(out1,
              out,
              ctx );
for(int i=0; i<16;i++) Serial.print(out[i],HEX);
Serial.println();
}
```

Table of Contents