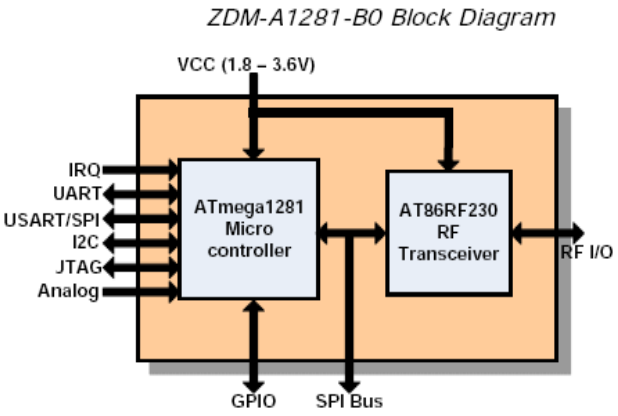
Travaux dirigés

Les réseaux sans fil - LRWPAN - 802.15.4 - ZigBee

Table des matières

1 - Caractéristiques d'un module ZigBee

On donne les caractéristiques d'un module ZigBee du marché.



Key features

- Ultra compact size (24 x 13.5 mm for ZDM-A1281-A2 module and 18,8 x 13.5 mm for ZDM-A1281-B0 module)
- Innovative (patent-pending) balanced chip antenna design with antenna gain of approximately 0 dBi (for ZDM-A1281-A2 version)
- High RX sensitivity (-101 dBm)
- Outperforming link budget (104 dB)
- Up to 3 dBm output power
- Very low power consumption (<6 µA in deep sleep mode)
- Ample memory resources (128 kBytes of flash memory, 8 kBytes RAM, 4 kBytes EEPROM)
- Wide range of interfaces (both analog and digital):
  - 10 spare GPIO, 2 spare IRQ lines
  - 4 ADC lines
  - UART with CTS/RTS control
  - I^2C, USART/SPI
- Up to 30 lines can be configured as GPIO
- Capability to write own MAC address into the EEPROM
- Optional antenna reference designs
- IEEE 802.15.4 compliant
- 2.4 GHz ISM band
- eZeeNet embedded software, including UART bootloader and AT command set

Benefits

- Less physical space constraints
- Best-in-class RF link range
- Longer battery life
- Easy prototyping with 2-layer PCB
- More memory for user software application
- Mesh networking capability
- Easy-to-use low cost Evaluation Kit
- Single source of support for HW and SW
- Worldwide license-free operation

RF Characteristics			
Parameters	**Range**	**Unit**	**Condition**
Frequency Band	2.400 to 2.4835	GHz	
Number of Channels	16		
Channel Spacing	5	MHz	
Transmitter Output Power	-17 to +3	dBm	Adjusted in 16 steps
Receiver Sensitivity	- 101	dBm	PER = 1%
On-Air Data Rate	250	kbps	
TX Output / Rx Input Nominal Impedance	100	Ohms	For balanced output

ATmega1281V Microcontroller Characteristics			
Parameters	**Range**	**Unit**	**Condition**
On-Chip Flash Memory Size	128	kBytes	
On-Chip RAM Size	8	kBytes	
On-Chip EEPROM Size	4	kBytes	
Operation Frequency	4	MHz	

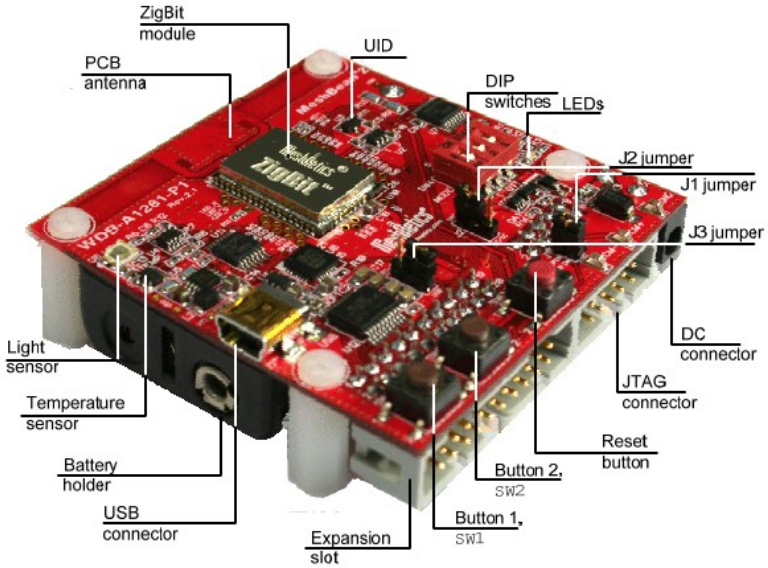
Module Interfaces Characteristics			
Parameters	**Range**	**Unit**	**Condition**
UART Maximum Baud Rate	38.4	kbps	
ADC Resolution / Conversion Time	10 / 200	Bits / µs	In the single conversion mode
ADC Input Resistance	100	MOhm	
ADC Reference Voltage (Vref)	1.0 to V_{cc} - 0.3	V	
ADC Input Voltage	0 ÷ Vref	V	
I^2C Maximum Clock	222	kHz	
GPIO Output Voltage (High/Low)	2.3 / 0.5	V	(-10 / 5 mA)
Real Time Oscillator Frequency	32.768	kHz	

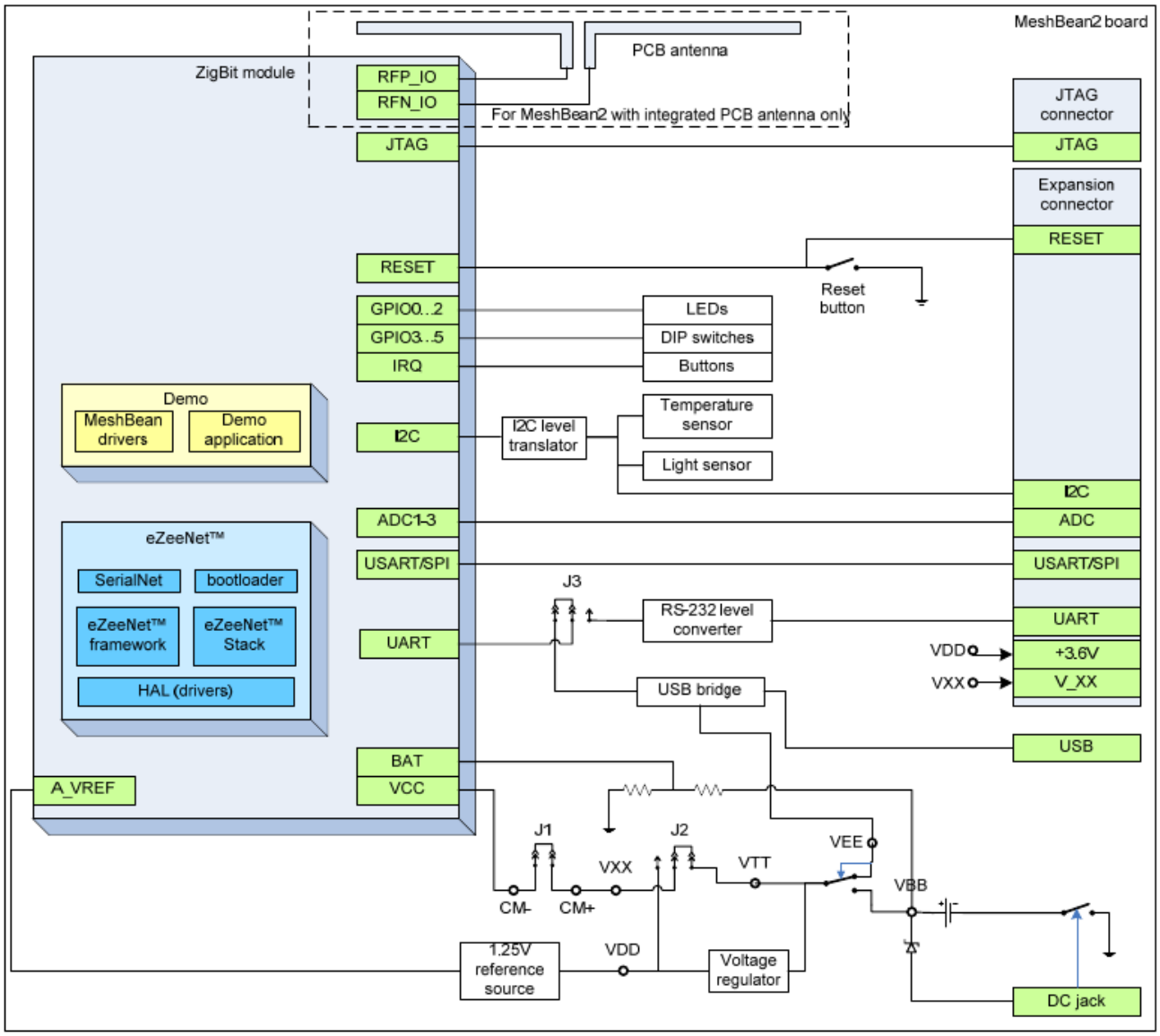
Question 1 : Complétez le tableau suivant :

Question	**Réponse**
Bande de fréquence utilisée	
Nombre de canaux RF	
Puissance d'émission maxi (en dBm et en mW)	
Vitesse de transmission entre module en RF	

Vitesse de transmission filaire maximum via RS232	
Topologies possibles	
Entrées/Sorties disponibles	
Sensibilité de la réception (dBm et mW)	

2 - Exemple de kit de développement





Question 2 : Repérez sur les figures précédentes les principales caractéristiques du kit de développement ZigBee (E/S, Liaisons, etc.).

3 - Exemple d'application : Lowpower

3.1 - Description de l'application

This sample shows how to collect data transmitted from low-power devices, employing the simplest power management strategy. At least 2 nodes are participating, but up to 7 end devices can be engaged. There must be one and only coordinator, and the rest of nodes must be configured as end devices.

For simple node identification, define unique logical addresses by DIP switches. When setting 8 possible DIP combinations, keep in mind that:

- ON position corresponds to logical 1;
- Third DIP switch defines the most significant bit for logical address;
- Zero logical address corresponds to the coordinator, which must be unique.

To start the application and to initiate the network push SW1 button on each node, starting with coordinator. Green LED is switched ON if the network is started successfully.

Coordinator organizes the network with its own 'unique' PAN ID which is determined by its MAC address (considering the 16 least significant bits). Besides, user can set PAN ID in flash memory or EEPROM. In order to join, end devices are scanning the network.

End device measures temperature each 10 seconds and sends data to coordinator if the absolute increment of the measured value exceeds 0.5°C. Flashing yellow LED is indicating that data are transmitting. After the transmission is completed, the end device falls

asleep. Unconditionally, the current temperature value is also sent if SW2 button is pressed on an end device, regardless the current node mode (sleeping or active). Do not abuse the device with quick series of multiple clicks, since this may cause unstable work.

Coordinator never sleeps; it keeps sending the received temperature data to UART via US

Question 3 : Que fait cette application ?

Question 4 : Quelle topologie réseau est mise en place dans cet exemple ?

3.2 - Etude des Makefiles

Extrait du Makefile de l'enddevice :

```
PROJNAME = enddevice

## path to BitCloud stack
STACK_PATH = ../../BitCloud/Components

include $(STACK_PATH)/../lib/MakerulesBcAll
include ./Makerules

CFLAGS += -DCHANGES_THRESHOLD=0
CFLAGS += -DAPP_USE_APS_ACK=1

#-------------------------------------
# Stack parameters being set to Config Server
#-------------------------------------
CFLAGS += -DCS_NWK_UNIQUE_ADDR=true
CFLAGS += -DCS_DEVICE_TYPE=DEVICE_TYPE_END_DEVICE
CFLAGS += -DCS_EXT_PANID=0xAAAAAAAAAAAAAAAALL
CFLAGS += -DCS_RX_ON_WHEN_IDLE=false

CFLAGS += -DCS_NEIB_TABLE_SIZE=7
CFLAGS += -DCS_MAX_CHILDREN_AMOUNT=6
CFLAGS += -DCS_MAX_CHILDREN_ROUTER_AMOUNT=2
CFLAGS += -DCS_ROUTE_TABLE_SIZE=30
CFLAGS += -DCS_END_DEVICE_SLEEP_PERIOD=10000

ifneq (, $(findstring -DAT86RF230B, $(CFLAGS)))
CFLAGS += -DCS_CHANNEL_MASK=(1L << 0x18)
else
  ifneq (, $(findstring -DAT86RF230, $(CFLAGS)))
  CFLAGS += -DCS_CHANNEL_MASK=(1L << 0x18)
  endif
endif

CFLAGS += -g

## app include dirs
INCLUDES += -I.

## app objects to build
OBJ =
## path to app objects
VPATH += .:

## objects to build with -O0
DBG_OBJ = enddevice.o

#-------------------------------------
# Build
#-------------------------------------
all: $(PROJNAME).elf $(PROJNAME).srec $(PROJNAME).hex size

$(OBJ) $(STACK_OBJ): %.o: %.c
      $(CC) -c $(CFLAGS) $(INCLUDES) $^ -o $@

$(DBG_OBJ): %.o: %.c
      $(CC) -c $(CFLAGS) -O0 $(INCLUDES) $^ -o $@
```

```
#-------------------------------------
# Link
#-------------------------------------
$(PROJNAME).elf: $(OBJ) $(DBG_OBJ) $(STACK_OBJ) Makefile
	$(CC)  $(CFLAGS)  $(OBJ)  $(DBG_OBJ)  $(STACK_OBJ)  $(LIB_PATH)/WdtInitatmega1281.o  -o  $
(PROJNAME).elf -L$(LIB_PATH) -L$(BSP_PATH)/lib -lBSPMeshBean -l$(STACK_LIB)
	rm -f *.o

%.srec: %.elf
	avr-objcopy -O srec --srec-len 128 $< $@

%.hex: %.elf
	avr-objcopy -O ihex $(HEX_FLASH_FLAGS)  $< $@

size:
	avr-size -td $(PROJNAME).elf

clean:
	rm -rf $(PROJNAME).elf $(PROJNAME).hex $(PROJNAME).srec $(PROJNAME).o

flash:
	jtagiceii -d ATmega1281 -f 0x9F62 -e -pf -if $(PROJNAME).hex


# eof Makefile
```

Question 5 : Complétez le tableau suivant :

Question	**Réponse**
Type d'adressage dans le réseau ?	
Donnez l'identifiant PAN.	
Numéro de canal ? Calculez la fréquence utilisée pour ce canal.	
Temps endormissement de l'enddevice ?	
Que représente enddevice.o ?	
Repérez la commande de compilation.	
Repérez la commande de génération du fichier hexa à flasher.	
Quel sera le nom du fichier à flasher dans l'avr ?	

On donne un extrait du Makefile du coordinateur :

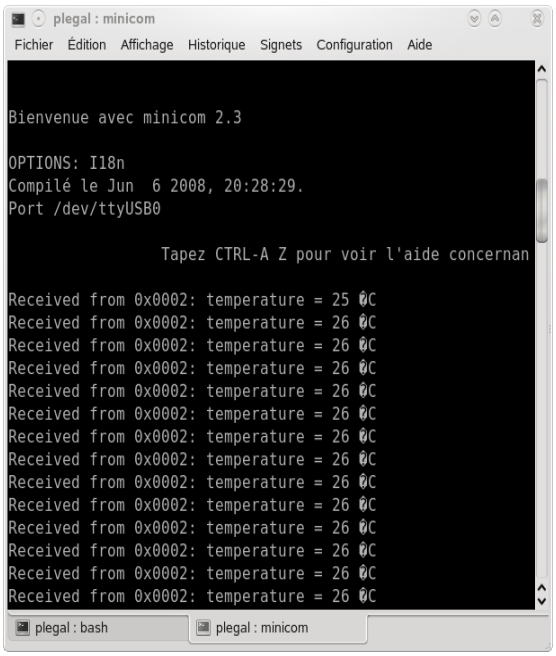
```
PROJNAME = coordinator

## path to BitCloud stack
STACK_PATH = ../../BitCloud/Components

include $(STACK_PATH)/../lib/MakerulesBcAll
include ./Makerules

# UART channel number
CFLAGS += -DAPP_UART_CHANNEL=UART_CHANNEL_1

CFLAGS += -DCHANGES_THRESHOLD=0
```

```
CFLAGS += -DAPP_USE_APS_ACK=1

#-------------------------------------
# Stack parameters being set to Config Server
#-------------------------------------
CFLAGS += -DCS_NWK_UNIQUE_ADDR=true
CFLAGS += -DCS_NWK_ADDR=0x0000
CFLAGS += -DCS_DEVICE_TYPE=DEVICE_TYPE_COORDINATOR
CFLAGS += -DCS_EXT_PANID=0xAAAAAAAAAAAAAAAALL
CFLAGS += -DCS_RX_ON_WHEN_IDLE=true

CFLAGS += -DCS_NEIB_TABLE_SIZE=8
CFLAGS += -DCS_MAX_CHILDREN_AMOUNT=7
CFLAGS += -DCS_MAX_CHILDREN_ROUTER_AMOUNT=0
CFLAGS += -DCS_ROUTE_TABLE_SIZE=8
CFLAGS += -DCS_END_DEVICE_SLEEP_PERIOD=10000

ifneq (, $(findstring -DAT86RF230B, $(CFLAGS)))
CFLAGS += -DCS_CHANNEL_MASK=(1L << 0x18)
else
  ifneq (, $(findstring -DAT86RF230, $(CFLAGS)))
  CFLAGS += -DCS_CHANNEL_MASK=(1L << 0x18)
  endif
endif
```

Question 6 : Complétez le tableau suivant :

Question	**Réponse**
Type d'adressage dans le réseau ?	
Donnez l'identifiant PAN.	
Numéro de canal ?	
Combien de enddevices peuvent se connecter au coordinateur ?	

Capture d'écran de la sortie du Coordinateur :



3.3 - Etude des codes sources

On donne le fichier source de l'enddevice:

```
/*********************************************************************
  enddevice.C

  ZigBeeNet Low-Power: Enddevice part of application implementation.

  Written by V.Marchenko
*********************************************************************/

//#include <lowpower.h>
#include <configServer.h>                           // Config Server header
#include <appFramework.h>                           // Main stack types
#include <zdo.h>                                    // Main ZDO header
#include <aps.h>                                    // Main APS header
#include <appTimer.h>                               // Application timer header
#include <sliders.h>                                // BSP sliders (DIP-switches) header
#include <buttons.h>                                // BSP buttons header
#include <sensors.h>                                // BSP sensors header
#include <leds.h>                                   // BSL LEDs header


#ifndef COORDINATOR_NETWORK_ADDRESS
#define COORDINATOR_NETWORK_ADDRESS         0x0000      // Default coordinator short (NWK) address
#endif

#ifndef APP_JOINING_INDICATION_PERIOD
#define APP_JOINING_INDICATION_PERIOD       500L        // Period of blinking during starting network
#endif

#define APP_ENDPOINT                        2           // Endpoint will be used
#define APP_PROFILE_ID                      1           // Profile Id will be used
#define APP_CLUSTER_ID                      1           // Cluster Id will be used

// Leds aliases definition hardware platfor supported if
#define APP_NETWORK_STATUS_LED            LED_GREEN     // Network status LED
#define APP_RECEIVING_STATUS_LED          LED_YELLOW    // Data receiving status LED
#define APP_SENDING_STATUS_LED            LED_RED       // Data transmission status LED


typedef enum
{
    APP_BUTTON_RELEASED_STATE,
    APP_BUTTON_PRESSED_STATE
} AppButtonState_t;

typedef enum
{
    APP_INITIAL_STATE,                                  // Application initial state (after Power On or Reset)
    APP_START_WAIT_STATE,                               // Waiting while the Button0 was not pressed
    APP_NETWORK_JOINING_STATE,                          // Joining network state
    APP_NETWORK_JOINED_STATE,                           // Network available
    APP_ERROR_STATE                                     // Error state
} AppState_t;


// Enddevice state
typedef enum
{
    DEVICE_ACTIVE_IDLE_STATE,                           // Device is not in sleep state and the temperature must be measured
```

```
    DEVICE_MEASURING_STATE,                             // Temperature measuring
    DEVICE_MESSAGE_SENDING_STATE,                       // Current temperature sending to the coordinator
    DEVICE_SLEEP_PREPARE_STATE,                         // Message was sent successfully. Node ready to sleep.
    DEVICE_SLEEP_STATE,                                 // Actually sleep state
    DEVICE_AWAKENING_STATE                              // Node was interrupted. Awakening.
} AppDeviceState_t;


typedef enum                                           // Data transmission feature state
{
    APP_DATA_TRANSMISSION_IDLE_STATE,                   // Data transmission was finished or(and) not started yet
    APP_DATA_TRANSMISSION_BUSY_STATE                    // Data transmission in progress
} AppDataTransmissionState_t;



typedef struct                                         // Application message
{
    uint16_t temperature;                               // Current temperature
} PACK AppMessage_t;

typedef struct                                         // Application message buffer
{
    uint8_t        header[APS_ASDU_OFFSET];             // Auxilliary header (stack required)
    AppMessage_t   message;                             // Actually application message
    uint8_t        footer[APS_AFFIX_LENGTH - APS_ASDU_OFFSET]; // Auxilliary footer (stack required)
} PACK AppMessageBuffer_t;



/******************************************************************************
  Global variables
 ******************************************************************************/
/******************************************************************************
  Local variables
 ******************************************************************************/
static AppState_t   appState = APP_INITIAL_STATE;                    // Current application state
static AppDeviceState_t appDeviceState = DEVICE_ACTIVE_IDLE_STATE;    // Current device state

// Application data transmission entity state
static AppDataTransmissionState_t appDataTtransmissionState = APP_DATA_TRANSMISSION_IDLE_STATE;
static ShortAddr_t  nwkAddr;                                         // Node NWK address

static HAL_AppTimer_t networkTimer;                                  // Timer indicating network start

// Endpoint simple descriptor (ZDO endpoint descriptor)
static SimpleDescriptor_t        simpleDescriptor = {APP_ENDPOINT, APP_PROFILE_ID, 1, 1, 0, 0 , NULL, 0, NULL};
static APS_RegisterEndpointReq_t apsRegisterEndpointReq;    // APS Register Endpoint Request primitive (APS endpoint descriptor)
static APS_DataReq_t apsDataReq;                            // APS Data Request primitive (for application message sending)


// ZDO primitives
static ZDO_StartNetworkReq_t  zdoStartNetworkReq;           // Request parameters for network start
static ZDO_SleepReq_t zdoSleepReq;                          // Request parameters for stack sleep
static ZDO_WakeUpReq_t zdoWakeUpReq;                        // Request parameters for stack awakening
static AppMessageBuffer_t appMessageBuffer;                 // Application message buffer

static AppButtonState_t key1State = APP_BUTTON_RELEASED_STATE;  // KEY1 (SW2) current state


/******************************************************************************
  Static functions
******************************************************************************/
static void buttonReleased(uint8_t button);                 // Button released handler
static void buttonPressed(uint8_t button);                  // Button pressed handler
// Network start/join confirmation handler
static void ZDO_StartNetworkConf(ZDO_StartNetworkConf_t *conf);
static void APS_DataConf(APS_DataConf_t *conf);             // Data transmission confirmation handler
// Temperature measured handler
static void temperaturesSensorHandler(bool result, int16_t temperature);
static void ZDO_SleepConf(ZDO_SleepConf_t *conf);           // Sleep confirmation handler
static void ZDO_WakeUpConf(ZDO_WakeUpConf_t *conf);         // Wake up confirmation handler
static void initApp(void);                                  // Common application initial function
static void deviceTaskHandler(void);                        // Common device task handler in network state
static void openPeriphery(void);                            // Open LEDs and Temperature Sensor
static void closePeriphery(void);                           // Close LEDs and Temperature Sensor
static void sendMessage(void);                              // Send the application message
static void startNetwork(void);                             // Start Network

/*****************************************************************
  Description: Starting network timer has fired. Toggle LED for blink
  Parameters: none.
  Returns:    none
*****************************************************************/
void startingNetworkTimerFired()
{
    BSP_ToggleLed(APP_NETWORK_STATUS_LED);    // Network Status LED toggling
}

/*****************************************************************
  Description: Application task handler
  Parameters: none.
  Returns:    none
*****************************************************************/
void APL_TaskHandler()
{
    switch (appState)
    {
        // node is in initial state
```

```
    case APP_INITIAL_STATE:                // Initial (after RESET) state
        initApp();                           // Init application as a whole
        SYS_PostTask(APL_TASK_ID);           // Application task posting
        break;

    case APP_NETWORK_JOINING_STATE:        // Network is in the joining stage
        startNetwork();                      // Start/joing network
        break;

    case APP_NETWORK_JOINED_STATE:         // Network was successfully started
        deviceTaskHandler();                 // Normal device operation when one joined network
        break;

    default:
        break;
    }
}

/*************************************************************************
  Description:  Open LEDs and Sensor
  Parameters:   none
  Returns:      none
*************************************************************************/
static void openPeriphery(void)
{
    BSP_OpenLeds();                // LEDs opening
    BSP_OpenTemperatureSensor();   // Temperature Sensor opening
}

/*************************************************************************
  Description:  Close LEDs and Sensor
  Parameters:   none
  Returns:      none
*************************************************************************/
static void closePeriphery(void)
{
    BSP_OffLed(APP_NETWORK_STATUS_LED);
    BSP_OffLed(APP_SENDING_STATUS_LED);
    BSP_OffLed(APP_RECEIVING_STATUS_LED);
    BSP_CloseLeds();               // LEDs closing
    BSP_CloseTemperatureSensor();  // Temperature Sensor closing
}


/*************************************************************************
  Description:  application and stack parameters init
  Parameters:   none
  Returns:      none
*************************************************************************/
static void initApp(void)
{
    // Read NWK address as dipswitch's state.
    nwkAddr = BSP_ReadSliders();
    // In this application end device cannot have network address the COORDINATOR_NETWORK_ADDRESS
    if (COORDINATOR_NETWORK_ADDRESS != nwkAddr)
    {
        // Set valid network address to Config Server
        CS_WriteParameter(CS_NWK_ADDR_ID, &nwkAddr);
        openPeriphery();                                // Periphery opening
        // Buttons opening with button released handler defining
        BSP_OpenButtons(buttonPressed, buttonReleased);
        appState = APP_START_WAIT_STATE;             // Wait for Button0 was released (pressed one time) state switching
    }
    else
    {
        appState = APP_ERROR_STATE;                  // Network address isn't valid. Error state switching to.
    }
}

/*************************************************************************
  Description:  Data sent handler
  Parameters:   conf - APS Data Confirm primitive
  Returns:      none
*************************************************************************/
static void APS_DataConf(APS_DataConf_t *conf)
{
    appDataTtransmissionState = APP_DATA_TRANSMISSION_IDLE_STATE; // Data transmission entity is idle
    BSP_OffLed(APP_SENDING_STATUS_LED);
    if (APS_SUCCESS_STATUS == conf->status)         // Data transmission was successfully performed
    {
        appDeviceState = DEVICE_SLEEP_PREPARE_STATE;                 // Switch device state to prepare for asleep
    }
    else
    {
        appDeviceState = DEVICE_ACTIVE_IDLE_STATE;    // Data transmission wasn't successfully finished. Retry.
    }
    SYS_PostTask(APL_TASK_ID);                      // Application task posting
}


/*************************************************************************
  Description:  Temperature measured handler
  Parameters:   result - measurement status (true - success, 0 - fail)
                temperature - value measured
  Returns:      none
*************************************************************************/
static void temperaturesSensorHandler(bool result, int16_t temperature)
{
    if (true == result)
```

```
    {
        appDeviceState = DEVICE_MESSAGE_SENDING_STATE;               //Switch device state to application message sending
         appMessageBuffer.message.temperature = temperature;            // Temperature measured will be sent as an application
message
    }
    //else
    // still in measuring
    SYS_PostTask(APL_TASK_ID);                                      // Application task posting
}

/*************************************************************
  Description:  Send the application message
  Parameters:   none
  Returns:      none
*************************************************************/
static void sendMessage(void)
{
    if (APP_DATA_TRANSMISSION_IDLE_STATE == appDataTtransmissionState)  // If previous data transmission was finished
    {
         appDataTtransmissionState = APP_DATA_TRANSMISSION_BUSY_STATE; // Data transmission entity is busy while sending not
finished
        // prepare and send APS Data Request
        apsDataReq.dstAddrMode             = APS_SHORT_ADDRESS;                         // Short addressing mode
        apsDataReq.dstAddress.shortAddress = COORDINATOR_NETWORK_ADDRESS;               // Destination node short address
        apsDataReq.dstEndpoint             = APP_ENDPOINT;                              // Destination endpoint
        apsDataReq.profileId               = simpleDescriptor.AppProfileId;             // Profile ID
        apsDataReq.clusterId               = APP_CLUSTER_ID;                            // Destination cluster ID
        apsDataReq.srcEndpoint             = APP_ENDPOINT;                              // Source endpoint
        apsDataReq.asduLength              = sizeof (AppMessage_t);                     // ASDU size
        apsDataReq.asdu                     = (uint8_t *) &appMessageBuffer.message;;        // ASDU pointer as an application
message
        apsDataReq.txOptions.acknowledgedTransmission = 1;                              // Acknowledged transmission enabled
        apsDataReq.radius                  = 0;                                         // Default radius
        apsDataReq.APS_DataConf            = APS_DataConf;                              // Confirm handler
        APS_DataReq(&apsDataReq);                                                       // Data Request sending
        BSP_OnLed(APP_SENDING_STATUS_LED);
    }
}


/*************************************************************
  Description:  ZDO Sleep Confirm handler
  Parameters:   conf - ZDO Sleep Confirm primitive
  Returns:      none
*************************************************************/
static void ZDO_SleepConf(ZDO_SleepConf_t *conf)
{
    if (ZDO_SUCCESS_STATUS == conf->status)   // Stack was slept successfully
    {
        closePeriphery();                       // LEDs and Temperature Sensor closing
        appDeviceState = DEVICE_SLEEP_STATE;    // Device actually slept
    }
    else
        SYS_PostTask(APL_TASK_ID);              // Still in current state.
    // Application task posting for attempt repeat.
}

/*************************************************************
  Description:  Prepare to sleep
  Parameters:   none
  Returns:      none
*************************************************************/
static void prepareToSleep(void)
{
    zdoSleepReq.ZDO_SleepConf = ZDO_SleepConf;  // Sleep Confirm handler defining
    ZDO_SleepReq(&zdoSleepReq);                 // Sleep Request sending
}

/*************************************************************
  Description:  Device common task handler
  Parameters:   none
  Returns:      none
*************************************************************/
static void deviceTaskHandler(void)
{
    switch (appDeviceState)                     // Actual device state when one joined network
    {
    case DEVICE_ACTIVE_IDLE_STATE:            // Device ready to temperature measuring
        BSP_ReadTemperatureData(temperaturesSensorHandler); // Temperature measuring
        break;

    case DEVICE_MESSAGE_SENDING_STATE:        // Message sending state
        sendMessage();                          // Application message sending
        break;

    case DEVICE_SLEEP_PREPARE_STATE:          // Prepare to sleep state
        if (APP_BUTTON_RELEASED_STATE == key1State)
            prepareToSleep();                   // Prepare to sleep
        else
            SYS_PostTask(APL_TASK_ID);          // Still in current state.
        break;

    case DEVICE_AWAKENING_STATE:              // Awakening state
        zdoWakeUpReq.ZDO_WakeUpConf = ZDO_WakeUpConf; // ZDO WakeUp confirm handler defining
        ZDO_WakeUpReq(&zdoWakeUpReq);               // ZDO WakeUp Request sending
        break;

    default:
        break;
```

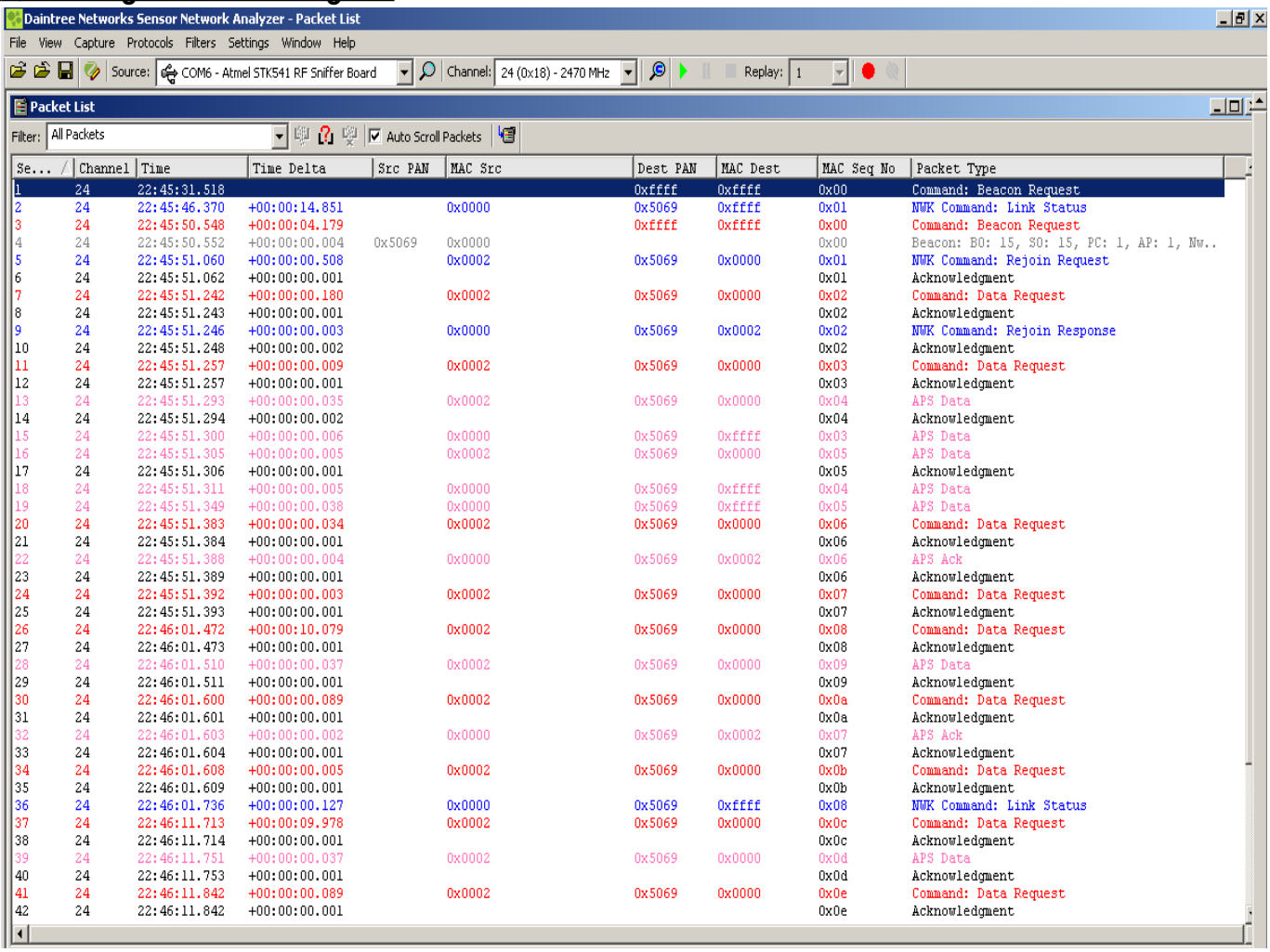
```
    }
}


/*************************************************************
  Description:  Application endpoint indication handler
  Parameters:   ind - APS Data Indication primitive
  Returns:      none
*************************************************************/
static void APS_DataInd(APS_DataInd_t *ind)
{
    ind = ind;  //unused parameter warning prevention
    BSP_ToggleLed(APP_RECEIVING_STATUS_LED);
}


/*************************************************************
  Description: ZDO_StartNetwork primitive confirmation was received.
  Parameters: confirmInfo - confirmation information
  Returns:    none
*************************************************************/
void ZDO_StartNetworkConf(ZDO_StartNetworkConf_t *confInfo)
{
    // Joined network successfully
    if ((ZDO_SUCCESS_STATUS == confInfo->status))   // NEtwork was started successfully
    {
        appState = APP_NETWORK_JOINED_STATE;          // Application state switching
        appDeviceState = DEVICE_ACTIVE_IDLE_STATE;    // Device state setting
        HAL_StopAppTimer(&networkTimer);              // Network join state indication timer stopping

        // Turn network indication on
        BSP_OnLed(APP_NETWORK_STATUS_LED);
        // Set application endpoint properties
        apsRegisterEndpointReq.simpleDescriptor = &simpleDescriptor;
        apsRegisterEndpointReq.APS_DataInd = APS_DataInd;
        // Register endpoint
        APS_RegisterEndpointReq(&apsRegisterEndpointReq);
    }
    SYS_PostTask(APL_TASK_ID);                     // Application task posting
}


/*************************************************************
  Description: start network
  Parameters: none.
  Returns:    none
*************************************************************/
static void startNetwork(void)
{
    // Configure timer for LED blinking during network start
    networkTimer.interval = APP_JOINING_INDICATION_PERIOD;
    networkTimer.mode     = TIMER_REPEAT_MODE;
    networkTimer.callback = startingNetworkTimerFired;
    HAL_StartAppTimer(&networkTimer);
    // Network started confirm handler
    zdoStartNetworkReq.ZDO_StartNetworkConf = ZDO_StartNetworkConf;
    // start network
    ZDO_StartNetworkReq(&zdoStartNetworkReq);
}


/*************************************************************
  Description: Button pressed handler
  Parameters: button - number of button was released
              (KEY1 as BSP_KEY0 - Join network,
              KEY2 as BSK_KEY1 - Wake up and send temperature value)
  Returns:    none
*************************************************************/
static void buttonPressed(uint8_t button)
{
    switch (button)
    {
    case BSP_KEY1:                              // Device wake up button
        key1State = APP_BUTTON_PRESSED_STATE;  // Button1 state changing
        break;

    default:
        break;
    }
}

/*************************************************************
  Description: Button released handler
  Parameters: button - number of button was released
              (KEY1 as BSP_KEY0 - Join network,
              KEY2 as BSK_KEY1 - Wake up and send temperature value)
  Returns:    none
*************************************************************/
static void buttonReleased(uint8_t button)
{
    switch (button)
    {
    case BSP_KEY0:                             // Network start button
        if (APP_START_WAIT_STATE == appState)     // If application wait this event
        {
            appState = APP_NETWORK_JOINING_STATE;   // Application state to join network switching
            SYS_PostTask(APL_TASK_ID);              // Application task posting
        }
        break;
```

```
    case BSP_KEY1:                              // Device wake up button
        key1State = APP_BUTTON_RELEASED_STATE;     // Button1 state changing
        if (DEVICE_SLEEP_STATE == appDeviceState)     // If device has slept
        {
            appDeviceState = DEVICE_AWAKENING_STATE;    // Device state switching
            SYS_PostTask(APL_TASK_ID);                  // Application task posting
        }
        break;

    default:
        break;
    }
}


/**************************************************************************
  Description: Device wakeup handler. Initialize
  Parameters: button - number of button was released
              (KEY1 as BSP_KEY0 - Join network,
              KEY2 as BSK_KEY1 - Wake up and send temperature value)
  Returns:    none
**************************************************************************/
static void wakeUpHandler(void)
{
    appState = APP_NETWORK_JOINED_STATE;
    appDeviceState = DEVICE_ACTIVE_IDLE_STATE;

    openPeriphery();

    // Turn network indication on
    BSP_OnLed(APP_NETWORK_STATUS_LED);

    SYS_PostTask(APL_TASK_ID);
}

/**************************************************************************
  Description: End device wake up indication

  Parameters: none.

  Returns: nothing.
**************************************************************************/
void ZDO_WakeUpInd(void)
{
    if (DEVICE_SLEEP_STATE == appDeviceState)
        wakeUpHandler();
}

/**************************************************************************
  Description: Network update notification

  Parameters: ZDO_MgmtNwkUpdateNotf_t *nwkParams - update notification

  Returns: nothing.
**************************************************************************/
void ZDO_MgmtNwkUpdateNotf(ZDO_MgmtNwkUpdateNotf_t *nwkParams)
{
    ZDO_StartNetworkConf_t conf;
    if (ZDO_NETWORK_STARTED_STATUS == nwkParams->status)
    {
        conf.status = ZDO_SUCCESS_STATUS;
        ZDO_StartNetworkConf(&conf);
    }
    else if (ZDO_NETWORK_LEFT_STATUS == nwkParams->status)
    {
        appState =  APP_NETWORK_JOINING_STATE;
        SYS_PostTask(APL_TASK_ID);
    }
}

/**************************************************************************
  Description: Wake up confirmation handler

  Parameters: conf - confirmation parameters

  Returns: nothing.
**************************************************************************/
void ZDO_WakeUpConf(ZDO_WakeUpConf_t *conf)
{
    if (ZDO_SUCCESS_STATUS == conf->status)
        wakeUpHandler();
    else
        SYS_PostTask(APL_TASK_ID);
}

//eof enddevice.c
```

Question 7 : Complétez le tableau suivant :

Question	Réponse
Quel est le langage utilisé ?	
Quelle est la différence entre APP_NETWORK_JOINING_STATE, et APP_NETWORK_JOINED_STATE ?	
Repérer la structure d'émission. Combien d'octets sont envoyés ?	
Quelle fonction est appelée au Reset ?	
Quelle fonction est appelée lorsque la jonction au réseau est effective ?	
Quelle fonction est appelée systématiquement lorsque l'enddevice est connecté au réseau ?	
Quelles sont les principales activités de l'enddevice lorsque qu'il est connecté au réseau ?	
Dans quelle fonction se passe l'émission sur le réseau ?	
Quelle est l'adresse du destinataire du message ?	
Quelle fonction sera automatiquement appelée par la stack ZigBee après l'émission ?	
Que se passe-t'il si l'émission à échouée ?	
Que se passe-t'il si l'émission est réussie ?	
A quoi sert la fonction SYS_PostTask(APL_TASK_ID); ?	

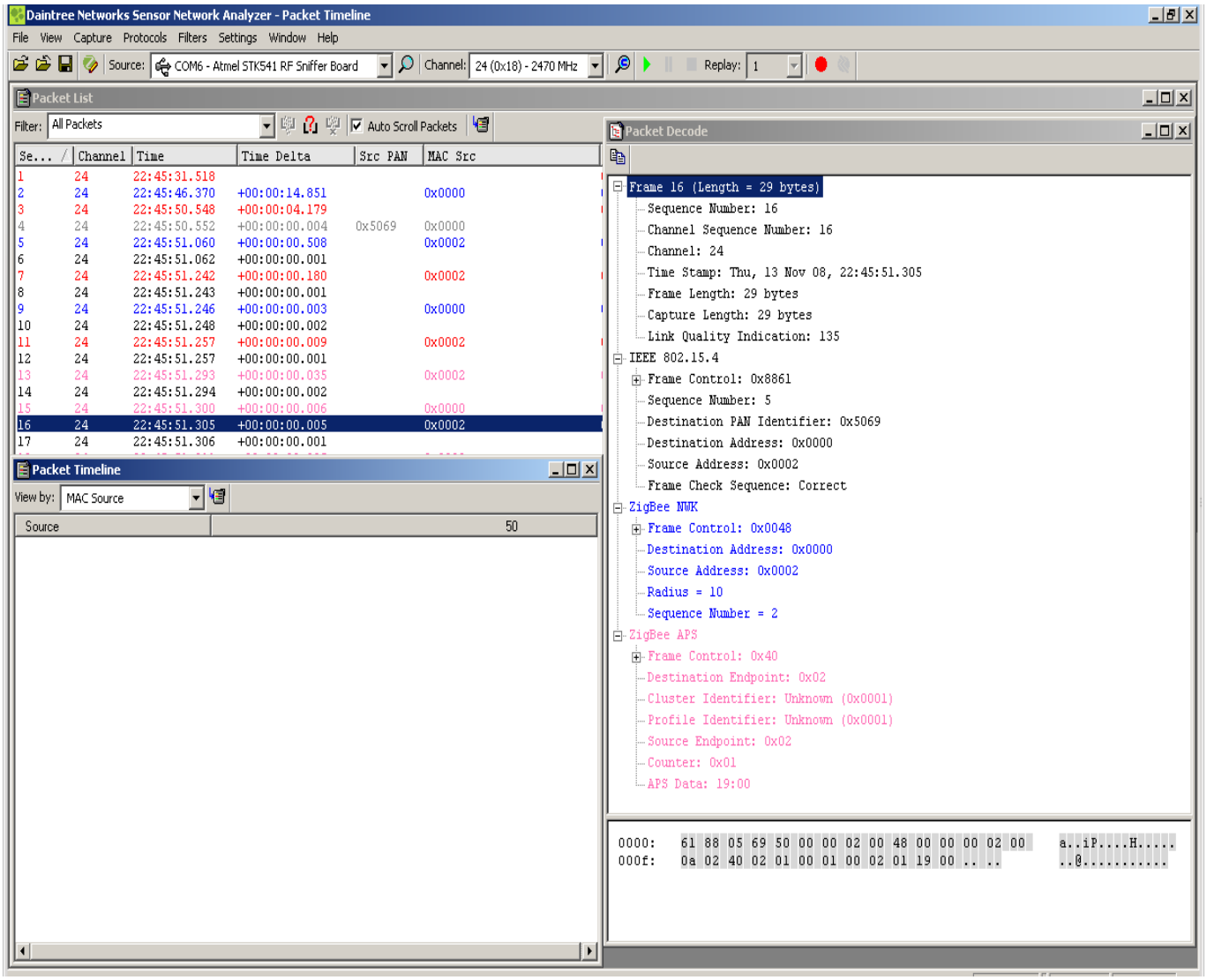
4 - Décodage de trames ZigBee



Se...	Channel	Time	Time Delta	Src PAN	MAC Src	Dest PAN	MAC Dest	MAC Seq No	Packet Type
1	24	22:45:31.518				0xffff	0xffff	0x00	Command: Beacon Request
2	24	22:45:46.370	+00:00:14.851		0x0000	0x5069	0xffff	0x01	NWK Command: Link Status
3	24	22:45:50.548	+00:00:04.179			0xffff	0xffff	0x00	Command: Beacon Request
4	24	22:45:50.552	+00:00:00.004	0x5069	0x0000			0x00	Beacon: BO: 15, SO: 15, PC: 1, AP: 1, Nw..
5	24	22:45:51.060	+00:00:00.508		0x0002	0x5069	0x0000	0x01	NWK Command: Rejoin Request
6	24	22:45:51.062	+00:00:00.001					0x01	Acknowledgment
7	24	22:45:51.242	+00:00:00.180		0x0002	0x5069	0x0000	0x02	Command: Data Request
8	24	22:45:51.243	+00:00:00.001					0x02	Acknowledgment
9	24	22:45:51.246	+00:00:00.003		0x0000	0x5069	0x0002	0x02	NWK Command: Rejoin Response
10	24	22:45:51.248	+00:00:00.002					0x02	Acknowledgment
11	24	22:45:51.257	+00:00:00.009		0x0002	0x5069	0x0000	0x03	Command: Data Request
12	24	22:45:51.257	+00:00:00.001					0x03	Acknowledgment
13	24	22:45:51.293	+00:00:00.035		0x0002	0x5069	0x0000	0x04	APS Data
14	24	22:45:51.294	+00:00:00.002					0x04	Acknowledgment
15	24	22:45:51.300	+00:00:00.006		0x0000	0x5069	0xffff	0x03	APS Data
16	24	22:45:51.305	+00:00:00.005		0x0002	0x5069	0x0000	0x05	APS Data
17	24	22:45:51.306	+00:00:00.001					0x05	Acknowledgment
18	24	22:45:51.311	+00:00:00.005		0x0000	0x5069	0xffff	0x04	APS Data
19	24	22:45:51.349	+00:00:00.038		0x0000	0x5069	0xffff	0x05	APS Data
20	24	22:45:51.383	+00:00:00.034		0x0002	0x5069	0x0000	0x06	Command: Data Request
21	24	22:45:51.384	+00:00:00.001					0x06	Acknowledgment
22	24	22:45:51.388	+00:00:00.004		0x0000	0x5069	0x0002	0x06	APS Ack
23	24	22:45:51.389	+00:00:00.001					0x06	Acknowledgment
24	24	22:45:51.392	+00:00:00.003		0x0002	0x5069	0x0000	0x07	Command: Data Request
25	24	22:45:51.393	+00:00:00.001					0x07	Acknowledgment
26	24	22:46:01.472	+00:00:10.079		0x0002	0x5069	0x0000	0x08	Command: Data Request
27	24	22:46:01.473	+00:00:00.001					0x08	Acknowledgment
28	24	22:46:01.510	+00:00:00.037		0x0002	0x5069	0x0000	0x09	APS Data
29	24	22:46:01.511	+00:00:00.001					0x09	Acknowledgment
30	24	22:46:01.600	+00:00:00.089		0x0002	0x5069	0x0000	0x0a	Command: Data Request
31	24	22:46:01.601	+00:00:00.001					0x0a	Acknowledgment
32	24	22:46:01.603	+00:00:00.002		0x0000	0x5069	0x0002	0x07	APS Ack
33	24	22:46:01.604	+00:00:00.001					0x07	Acknowledgment
34	24	22:46:01.608	+00:00:00.005		0x0002	0x5069	0x0000	0x0b	Command: Data Request
35	24	22:46:01.609	+00:00:00.001					0x0b	Acknowledgment
36	24	22:46:01.736	+00:00:00.127		0x0000	0x5069	0xffff	0x08	NWK Command: Link Status
37	24	22:46:11.713	+00:00:09.978		0x0002	0x5069	0x0000	0x0c	Command: Data Request
38	24	22:46:11.714	+00:00:00.001					0x0c	Acknowledgment
39	24	22:46:11.751	+00:00:00.037		0x0002	0x5069	0x0000	0x0d	APS Data
40	24	22:46:11.753	+00:00:00.001					0x0d	Acknowledgment
41	24	22:46:11.842	+00:00:00.089		0x0002	0x5069	0x0000	0x0e	Command: Data Request
42	24	22:46:11.842	+00:00:00.001					0x0e	Acknowledgment

Question 8 : Complétez le tableau suivant :

Question	**Réponse**
Nombre de modules ZigBee en communication.	
Adresses logiques des modules ZigBee en communication.	
A quoi reconnait-on le coordinateur ?	
Donnez l'identifiant PAN.	
Quels sont les types de trames en jeux dans l'échange ?	
A quoi sert la trame 5 ? Justifier.	
A quoi sert la trame 9 ? Justifier.	
Se trouve-t'on en présence du communication en mode beacon ou non-beacon ? Justifier.	



Question 9 : Complétez le tableau suivant :

Question	Réponse
A qui s'adresse la trame 16 ?	
Quel sont les valeurs des endpoints source et destination ?	
Quel est l'utilité de la notion de endpoint ?	
Quel est la données transportée ?	
Quelle est la taille de la trame ?	
Calculez la durée de transmission de la trame.	