AN1200.03 Application Note

Frequency Hopping

Table of Contents

1 INTRODUCTION

This application note is designed to enable you to develop a frequency hopping system using an XE1205 on both the transmit and the receive part. The XE1205 is the ideal transceiver for wide band and narrow band communications. The 15-Byte FIFO is also ideal to gain CPU resources. This application is developed under the MSP430 with an 8MHz CPU clock.

2 SPREAD SPECTRUM SYSTEMS

In spread spectrum systems, the signal energy is spread over a wider frequency range. These systems have the advantages of being less sensitive to interference and of being harder to be detected. In the ISM bands, it is often used in the US band 902-928 MHZ, where there is a lot of interference. Moreover, using spread spectrum techniques allows one to transmit power up to 1W under FCC regulations. The Frequency hopping system is a spread spectrum system. FCC specifications for frequency hopping systems are detailed below.

3 FCC SPECIFICATIONS

According to the FCC regulations section 15.247, it is discriminated channels having a 20 dB bandwidth lower than 250 kHz and between 250 – 500 kHz.

For channels lower than 250 kHz, the system must use at least 50 hopping frequencies within a period of 20 sec. So the minimum hopping frequency is 2.5 Hz.

For channels having a 20 dB bandwidth between 250 – 500 kHz, the system must use at least 25 hopping frequencies within a period of 10 sec.

In the two cases, the system shall hop following a pseudo random order. The maximum output power is 1 Watt for systems using at least 50 hopping channels and 0.25 Watt for systems using less than 50 hopping channels.

4 PROTOCOL: SYNCHRONIZATION AND COMMUNICATION

In Frequency hopping systems, there are mainly two phases. First, transmitter and receiver must become synchronized. Then, they must hop and communicate together. During the communication phase, many cases must be considered, for example if some packets or the communication is lost. This application note will explain the way synchronization and communication phase are implemented.

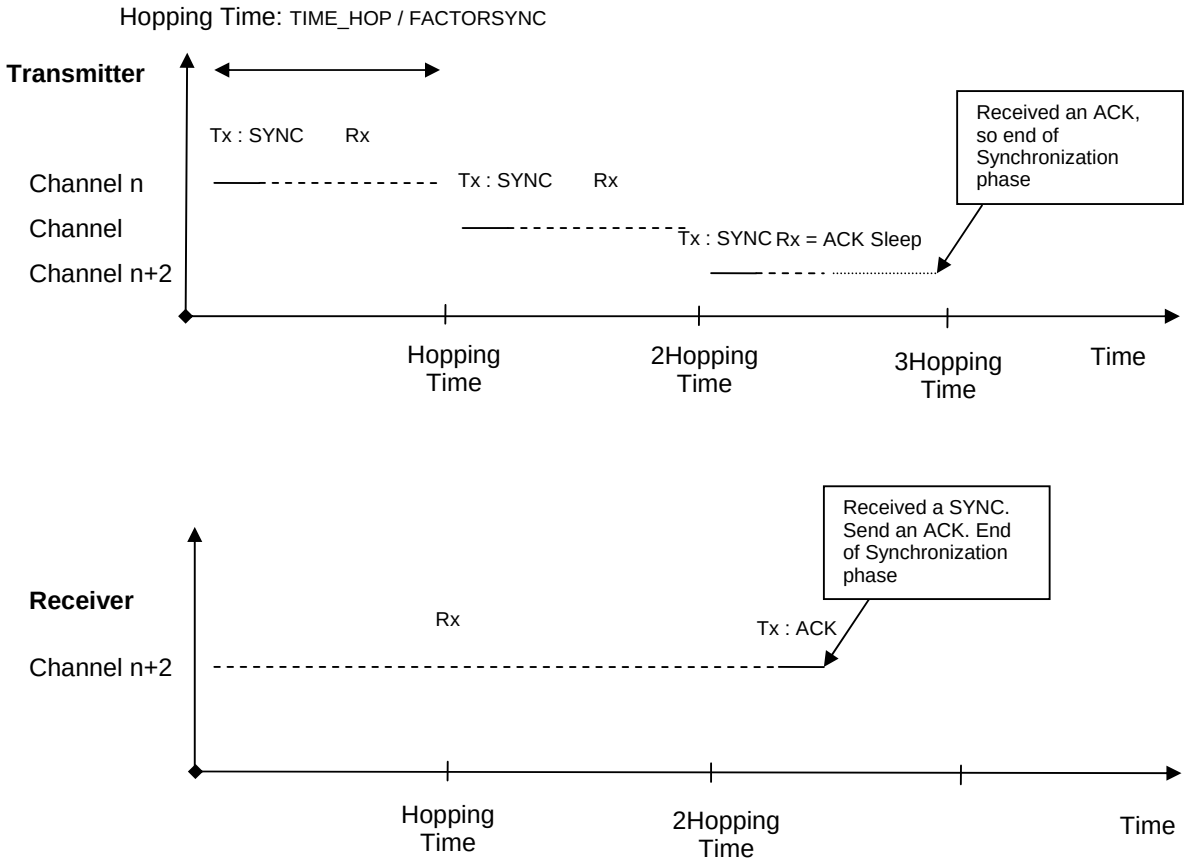
4.1 SYNCHRONIZATION

During the synchronization phase, the receiver waits on a channel and the transmitter hops to predefined channels. It always sends the same data “SYNC” phase and when the transmitter and the receiver are on the same channel and that the receiver recognizes this data, it then sends an “ACK” to the transmitter. They are then synchronized and can go into communication phase.

During the communication phase, the hopping time is defined by the variable TIME_HOP and adjusted to a clock of 125 KHz. So to program a hopping time of 80 ms, TIME_HOP = 80 * 125 = 10000.
During the synchronization phase and to reduce it, the transmitter hops quickly than during the communication phase. The hopping time is Hopping Time = TIME_HOP/FACTORSYNC.

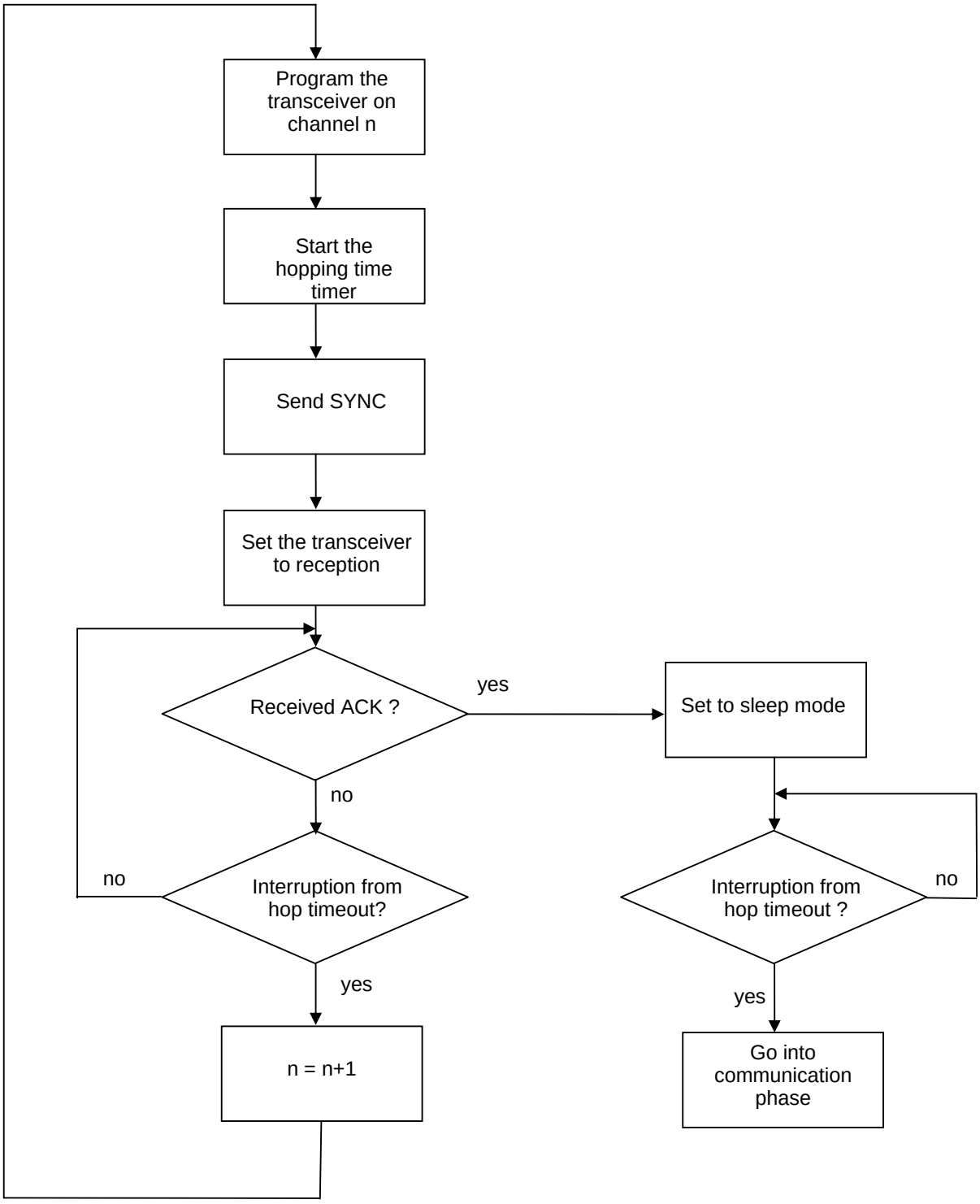
4.1.1 Timing and frequency behavior

Bellow is illustrated the transmitter and the receiver behavior (channel vs time) during the synchronization phase :

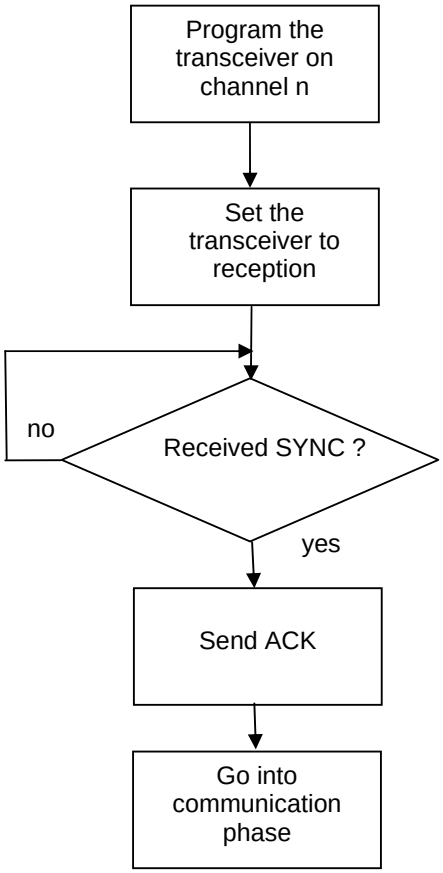


4.1.2 Flowchart

Synchronization for the transmitter:



Synchronization for the receiver:



4.1.3 Timing optimization

The maximum time to get synchronized is a function of the number of channels (NBCHANNEL), the communication hopping time (TIME_HOP) and the factor (FACTORSYNC) used to decrease the synchronization phase. So

$$T_{SyncMax} = \frac{NBCHANNEL * HoppingTime}{ClkTimer} = NBCHANNEL * \frac{TIME\_HOP}{ClkTimer * FACTORSYNC}$$

For NBCHANNEL = 50, TIME_HOP = 10000, FACTORSYNC= 16 and ClkTimer = 125 then $T_{syncmax}$ = 250 ms.

The hopping time during the synchronization should be greater than the time needed to send the frame SYNC and ACK. This timing is obviously a function of the Bit rate. For a Bite rate of 76.8 kbit/s, it takes 1.35 ms to send the four ascii byte “SYNC” and 1.25 ms to send “ACK” according to the frame structure (32 bits of preamble, 32 bits of pattern, 8 bits of the data size and the data). We must also take into account the time it takes to go from transmitter to receiver mode and the code execution. We can estimate it at about 800 us. So we can consider that the minimum hopping time is : 1.35 + 0.8 + 0.8 + 1.25 = 4.2 ms. For TIME_HOP = 10000, FACTORSYNC = 16 and ClkTimer = 125, the hopping time is 5 ms. It is then a quite good optimization.

The parameters NBCHANNEL, TIME_HOP and FACTORSYNC are located in the file FreqHopping.h.
The channels are defined in the table FreqTable[ ] in the file FreqHopping.c.

4.2 COMMUNICATION PHASE

After the synchronization phase, the transmitter and the receiver enter into the communication phase. At the beginning of this phase, the transmitter and the receiver hop to the next channel. Then the receiver is set in receive mode and the transmitter sends the data. When the data is received correctly the receiver sends an ACK to the transmitter. The receiver then hops to the next channel while the transmitter goes into sleep mode and waits until the end of the hopping time to hop to the next channel. We then ensure that the hopping time is well conserved independently to the length of the burst of data. In this configuration, we also ensure that the receiver is in reception before the transmitter sends its data. The transmitter and receiver hop to the next channel as defined in the table FreqTable [], they don't hop in a pseudo-random way.

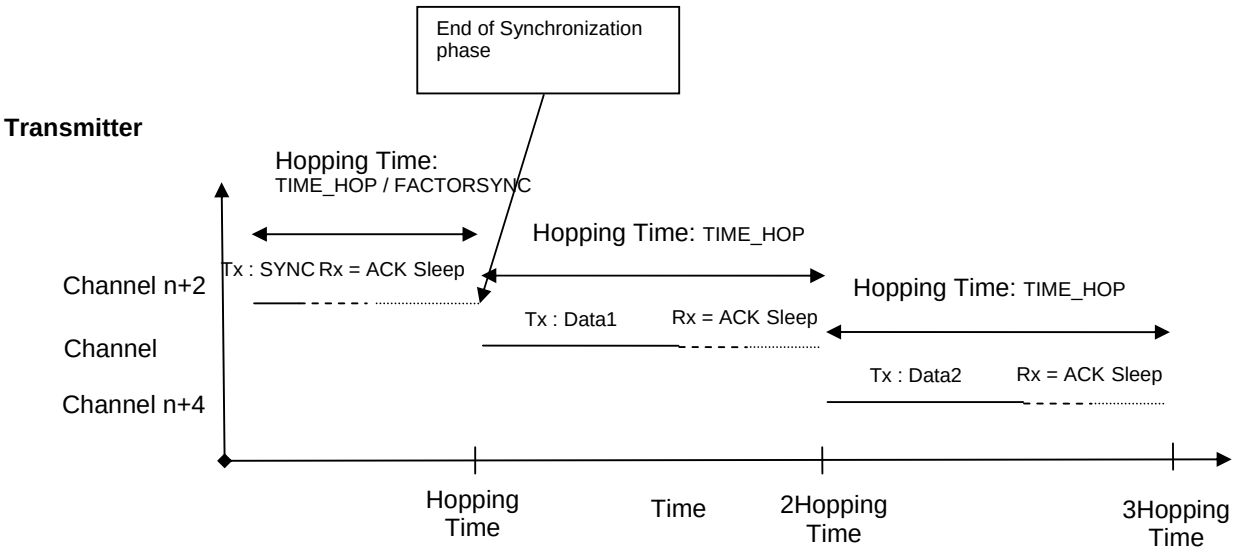
During this communication phase, the transmitter hops, transmits data and waits for an acknowledgement. If it doesn't receive the ACK, either it hops to the next channel and transmits the same data or it goes back into synchronization. It goes back into synchronization if it has already lost more than MAXLOSSACK acknowledgment.

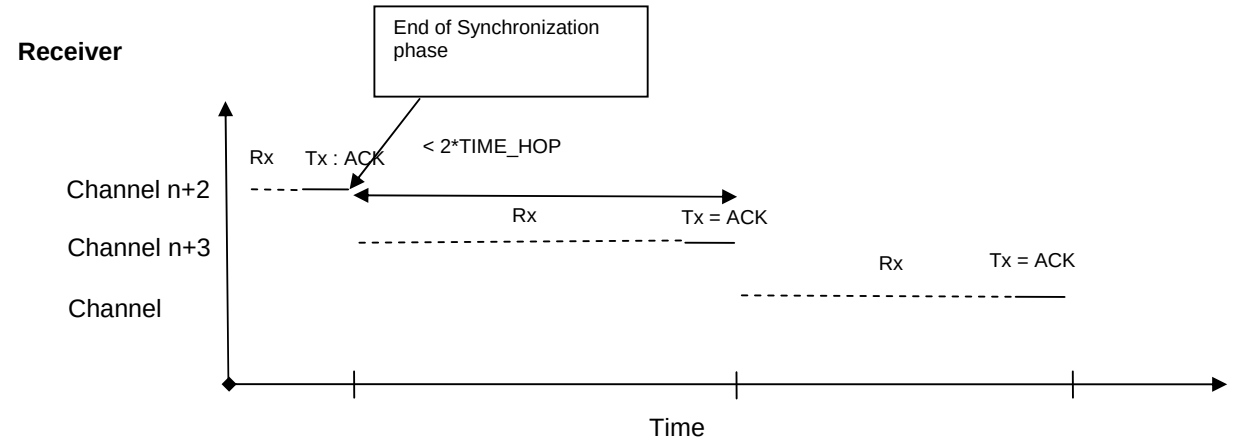
During the communication phase, the receiver hops, wait on a data channel until the Rx timeout. This timeout is 2* TIME_HOP. If it didn't receive good data, either it hops to the next channel + 1 regarding the timeout or if it has lost more than MAXRXLOSSBURST burst, it goes back into synchronization phase.

The length of a burst of data is defined by the variable NBDATABYTEBURST. It determines the number of byte send in a burst. The parameters MAXLOSSACK, MAXRXLOSSBURST and NBDATABYTEBURST are located in the file FreqHopping.h.

In the frame structure, you could introduce an ID field to check the packet ID received. In case the transmitter don't receive an acknowledgement and retransmit the same data, the receiver could compare the packet ID received with the previous packet ID. It could then decide to add it or not to the previous frame to construct the complete data buffer.

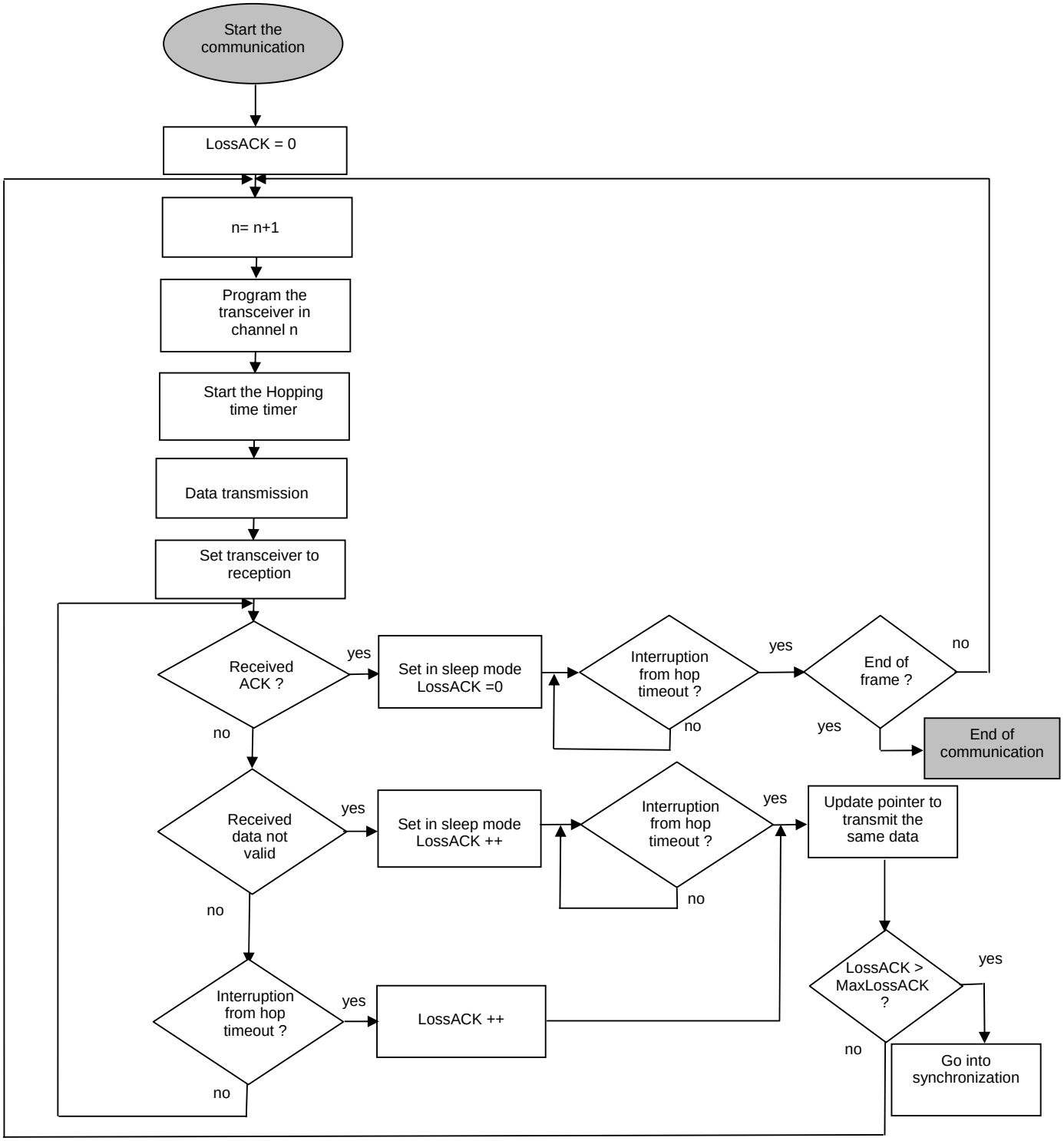
4.2.1 Timing and frequency behavior



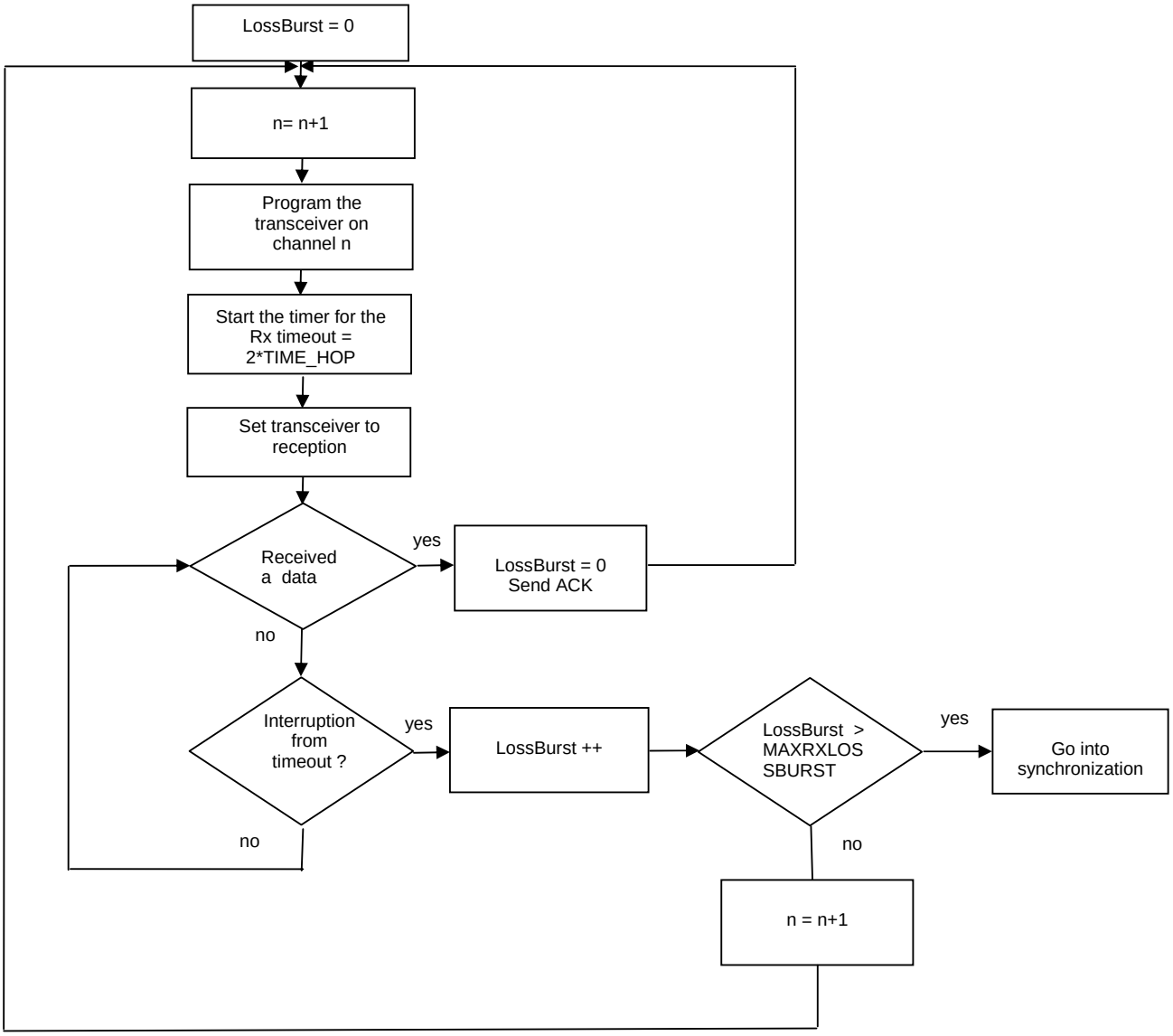


4.2.2 Flowchart

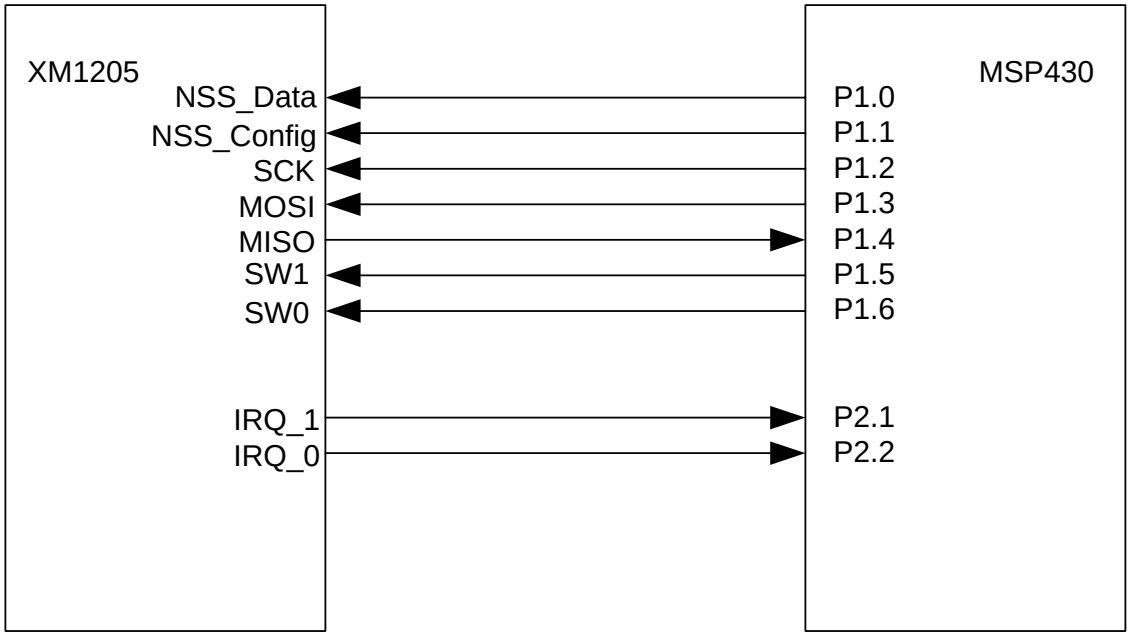
Communication phase for the transmitter



Communication phase for the receiver:



5 HARDWARE IMPLEMENTATION



6 SOFTWARE IMPLEMENTATION

The main function calls functions that first initialize the microcontroller InitMicro(); and the RF Chip InitRFChip();. When programming the transmitter, the EnableTransceiver variable must be set to true and false when programming the receiver.
As a demonstration of this frequency hopping application, the main function is developed as followed:
In a loop, the transmitter and the receiver are set in synchronization phase. When they are synchronized, the transmitter enters into communication phase. Then, during a burst, the transmitter sends one byte. The receiver shows the received byte on the port 4. The data sent are defined in the buffer RFbuffer. They communicate following the flowchart indicated in paragraph 4.2.2. When the complete buffer is sent, they then go back into synchronization phase. The same buffer is then sent and the received data shown on port 4 following a loop.
During a burst, the length of the data sent is limited to one byte. But it is only as a demonstration and to show that the transmit data is well received and to show it on port 4. The length of the data sent can obviously be increased. But one must take into account the hopping time to define the length of the data sent.

Following is the main function:

```
int main (void)
{

    _U8 returnCode = -1;
    _U8 txChannelIn, channel, rxChannelIn;


  //micro initialisation
  InitMicro();

  //transceiver initialisation
  InitRFChip();

  if (EnableTransceiver)//Transmitter
  {
        //First channel for the synchronization
        txChannelIn = 0;
        channel = txChannelIn;
```

```
            //Write datas in a buffer
            RFbuffer[0]= 0x01;
            RFbuffer[1]= 0x02;
            RFbuffer[2]= 0x04;
            RFbuffer[3]= 0x08;

            while(1)
            {
                  //Pointer initialization to transmit the datas
                  IniTxFreqHopping(RFbuffer,strlen(RFbuffer));

                  //Synchronization phase
                  channel = TxAcquisition(channel);

                  //Hopping and data transmission
                  returnCode = 0;
                  channel = TxHopping(channel, &returnCode);

                  //while datas are not totally transmitted,
                  do
                  {
                        //Case where too much burst are lost
                        if (returnCode & RX_LOSS)
                        {
                              channel = TxAcquisition(channel);//Go into synchronization phase
                              returnCode = 0;
                              //Hopping and data transmission
                              channel = TxHopping(channel, &returnCode);
                        }
                  } while(!(returnCode & FRAME_TX_DONE));

            }
      }

      else //Receiver
      {
            //First channel for the synchronization
            rxChannelIn = 0;

            //LO adjustement between transmitter and receiver
            FrequencyCorrection();

            //Synchronization phase
            RxAcquisition(rxChannelIn);

            //Hopping and data reception
            channel = RxHopping(rxChannelIn);

            while(1)
            {
                  RxAcquisition(channel);//Synchronization phase
                  channel = RxHopping(channel); //Hopping and data reception
            }
      }
}
```

All the functions called by the main function are located in 3 files: INITIALISATION.C, XE1205DRIVERMSP430.C and FREQHOPPING.C.

6.1 INITIALISATION.C

This file contains the functions and sub functions used to initialize the microcontroller.

InitMicro : Initializes the MicroController peripherals
InitPort1 : Initializes the Port 1
InitPort2 : Initializes the Port 2
InitPort3 : Initializes the Port 3
InitPort4 : Initializes the Port 4
InitPort5 : Initializes the Port 5
InitPort6 : Initializes the Port 6
InitClockModule : Initializes the Clock Module. MCLK is first connected to DCOCLK and when LFXT1 is settled MCLK = LFXT1, SMCLK=DCOCLK (Res=0, DCO=0 -> freq=74.1 KHz).
InitTimerA : Initializes the Timer A
InitTimerB : Initializes the Timer B

6.2 XE1205DRIVERMSP430.C

This contains all the functions used to drive and initialize the XE1205.
The InitRFChip() function initializes the RF Chip according to the variable RegistersCfg[]. The XE1205 drivers developed under the MSP430 is very similar to the XE1205 drivers developed with the XE8000 (see API TN8000.18 “XE8000 driving XE1200 transceivers”). The sub functions prototype and name variables are conserved. So for more explanation about XE1205 driver, please refer to this API.

The XE1205 is initialized with a bit rate of 76.8 kbps and a frequency deviation of 80 KHz. The 20 dB bandwidth obtained is then 160 KHz, so lower than 250 KHz.

Configuration functions
InitRFChip: This routine initializes the RF chip registers using pre initialized variables

SetRFMode: Sets the XE1205 operating mode (Sleep, Receiver, Transmitter)
WriteRegister: Writes the register value at the given address on the XE1205
ReadRegister: Reads the register value at the given address on the XE1205

Communication functions
SendRfFrame: Sends an RF frame
ReceiveRfFrame: Receives an RF frame
SendByte: Sends data to the transceiver trough the SPI interface
ReceiveByte: Receives data from the transceiver trough the SPI interface

Utility functions
Wait: Creates a delay using the timer A and ACLK=1MHz clock
StartTimerCLK125KHz: Initializes and start the timer B to create an interruption when the delay is reached using the ACLK/8=125KHz clock
EnableTimeOut : Enables/Disables the RF frame timeout. When enabled, the timer A is initializes to create an interruption when the delay is reached using SMCLK/8 = DCOCLK/64 = 1168 Hz clock
InvertByte: Inverts a byte. MSB -> LSB, LSB -> MSB
SpiInOut: Sends and receives a byte from the SPI bus

Interruption functions
interrupt [PORT2_VECTOR] void Port2(void): interruption from IRQ0_RX_IRQ_WRITE_BYTE, used by ReceiveRfFrame() to receive a frame.
interrupt [TIMERA0_VECTOR] void Timer_A0(void): interruption from the timer A

6.3 FREQHOPPING.C

All the functions used for the frequency hopping protocol are located in this file.

Transmitter functions:

IniTxFreqHopping : Variables and pointers initialization *for the frame transmition.*

TxAcquisition : Tx acquisition phase. The transmitter hops to channels and sends the data SYNC. When an ACK is received, it means that Rx and Tx are synchronized. They then go into communication phase.

TxHopping : Tx hopping and communication routine. The transmitter hops and sends the data until the complete frame is not transmitted or until the synchronization is lost with the receiver.

TransmitBurst : Burst transmission

Wait_ACK : Set the transmitter in reception mode until it receives the data ACK or it receives an interruption from the timer meaning the end of the hop. When the data ACK is received, the transmitter is set in sleep mode until the end of the timer interruption.

Receiver functions:

FrequencyCorrection : Frequency error between transceiver and receiver. Frequency table correction.

RxAcquisition : Rx Acquisition phase. The receiver listens on a channel. When the data SYNC is received, it then sends the data ACK. Tx and Rx are then synchronized. They then go into communication phase.

RxHopping : Rx hopping and communication routine. The receiver hops and waits on a channel for data until a timeout. When data are received, it then sends an ACK and hops to the next channel. If no data are received and the timeout occurs, it then hops to the next channel +1 or return into synchronization phase if the number of lost burst is greater than MAXRXLOSSBURST

ReceiveBurst : The receiver is set in receive mode. It waits for data until a timeout occurs.

Common functions:

ChannelHop : Transceiver channel programming

Contact Information

Semtech Corporation
Wireless and Sensing Products Division
200 Flynn Road, Camarillo, CA 93012
Phone (805) 498-2111 Fax : (805) 498-3804