Freescale Semiconductor
Application Note

Document Number: AN3941
Rev. 0, 01/2010

Wireless Serial Link Driver on 2.4 GHz

by: Pavel Krenek,
Application Engineering
Roznov CSC, Czech Republic

Contents

1 Introduction

This application note describes the software device driver for the MC1319x, MC1320x, MC1321x transceivers, and how it can be integrated and used in an application on the HCS08 and ColdFire V1 families of MCUs. This driver must be used only for the peer to peer transparent data transfer (example of communication: Sensor – Actator, Sensor – PC, and PC – PC).

The base of this data transfer is half-duplex communication between two MCUs using a Freescale 802.15.4 transceiver. This driver was built to work with HC(S)08 and ColdFire V1 families and can easily be adapted to almost any processor core. The data transfer supports an asynchronous acknowledged protocol that can be modified by the user to non-acknowledged protocol. Full source code and example applications are available on www.freescale.com.

The driver may be configured to run on any HC(S)08 and ColdFire MCUs. You can specify different timer channels and I/O pins on the MCU allowing flexibility in the application design.

Features of the driver include:

- Low power, bi–directional RF communication link
- Configurable statically for frequency band, net number, device ID, and packet ID
- Configurable acknowledge or non-acknowledge protocol
- Transmit and receive variable length messages with up to 110 bytes data
- Compatible with MC1319x, MC1320x, and MC1321x

- Compatible with 8-bit Freescale MCU families HC08, HCS08, and the 32-bit Freescale MCU family ColdFire V1

Examples of using the driver is shown in the following three figures:

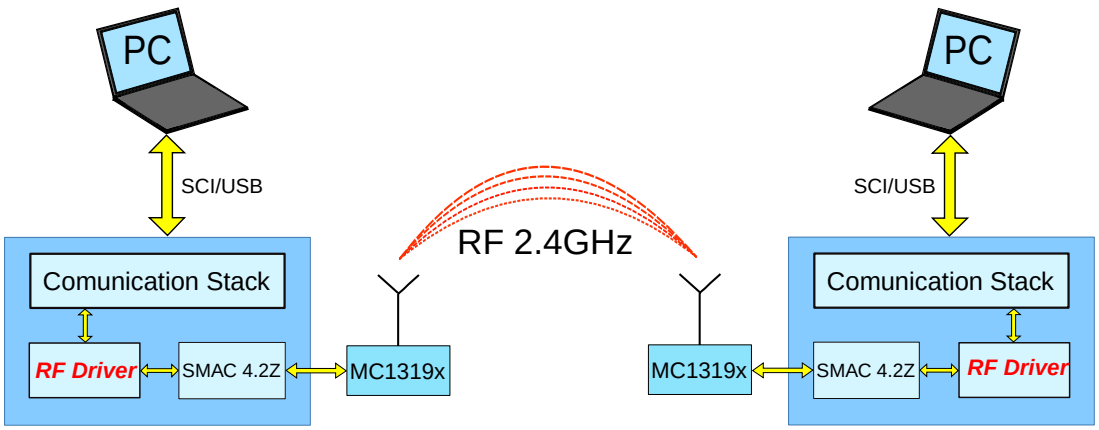

Figure 1. Example of communication between PC–PC

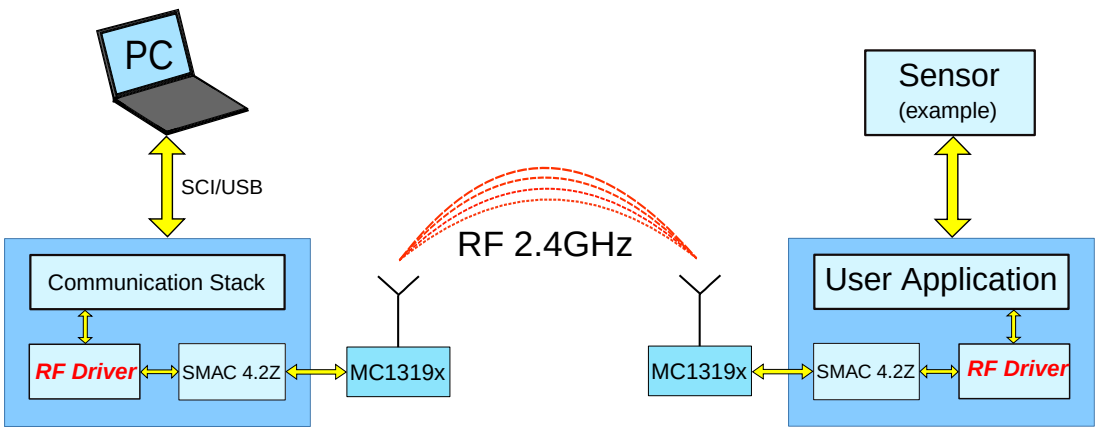

Figure 2. Example of communication between PC–Sensor

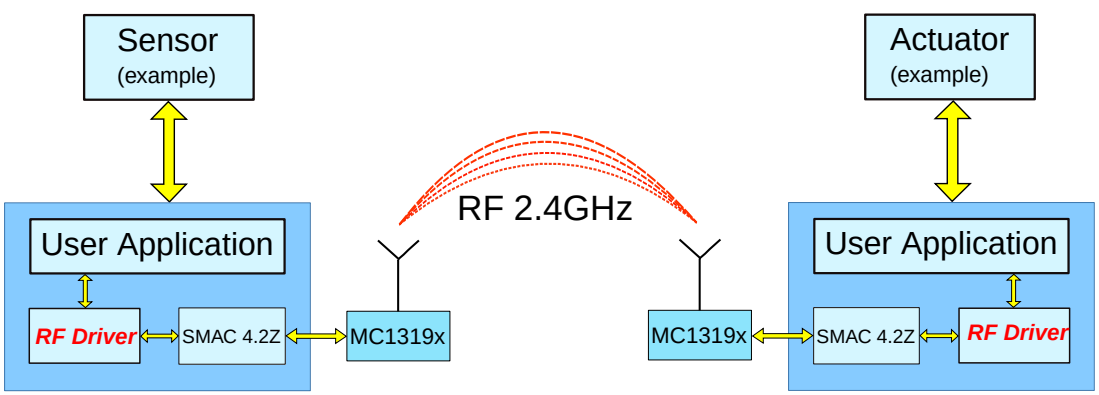

Figure 3. Example of communication between Sensor–Actuator

2 Communication Concept

The Freescale 802.15.4 transceiver can support communication on the 16 channels in the 2.4 GHz band. The RF driver can be used in an application requiring two-way communication or flexible allocation of transmitter and receiver roles that can be changed dynamically.

3 Packet Format

The driver supports sending messages in the following format. First is a netnum in the packet structure that is used for identification of a specific network (for example, more application with the RF communication). The command is used for sending required messages in the packet (data, handshake, acknowledge, and non-acknowledge). The packet ID is used for recognizing the sequence of a specific packet. The packet format is shown in the following table.

Table 1. Packet structure

NET-NUM	COMMAND	PACKET ID	DATA
4 bytes	2 bytes	2 bytes	(0-110) bytes

NETNUM—A unique number of network for communication between two connected devices. This number must be the same for both communicating devices.

COMMAND—The command represents information about the type of sending packet.

- DATA—This command is used to send data. The maximum size of the data in the packet is 110 bytes. The driver supports sending unlimited data. The command is defined in the header file as a character ASCII ‘D’.

  ```
  #define RF_COMM_CMD_DATA 'D'
  ```

- HANDSHAKE—This command is used to search and to connect with the opposite device. One of the devices begins to send periodically handshake packets and then waits for an answer from the other device. This process is repeated with time delays until the devices are connected. The command is defined in the header file as ASCII character ‘H’.

  ```
  #define RF_COMM_CMD_HANDSHAKE 'H'
  ```

- ACK—This command ACK (acknowledge) is used for confirmation of a received (handshake or data) message from the other device. The command is defined in the header file as ASCII character ‘A’.

  ```
  #define RF_COMM_CMD_ACK 'A'
  ```

- NACK—The receive side sends this command when it processes an operation and is not ready to receive the next data. The command is defined in the header file as ASCII character ‘N’.

  ```
  #define RF_COMM_CMD_NACK 'N'
  ```

PACKET ID—Is a unique number of tx packets. This number serves for receiving the correct data packet and sending the correct acknowledge. This number is incremented in every cycle and guarantees correctly receiving the messages

DATA—Contains the actual message sent, up to 110 data bytes

The following three figures show possible states of communication between the devices.

The first figure, Figure 4 shows connecting Device I. to the Device II. by using the handshake packet.

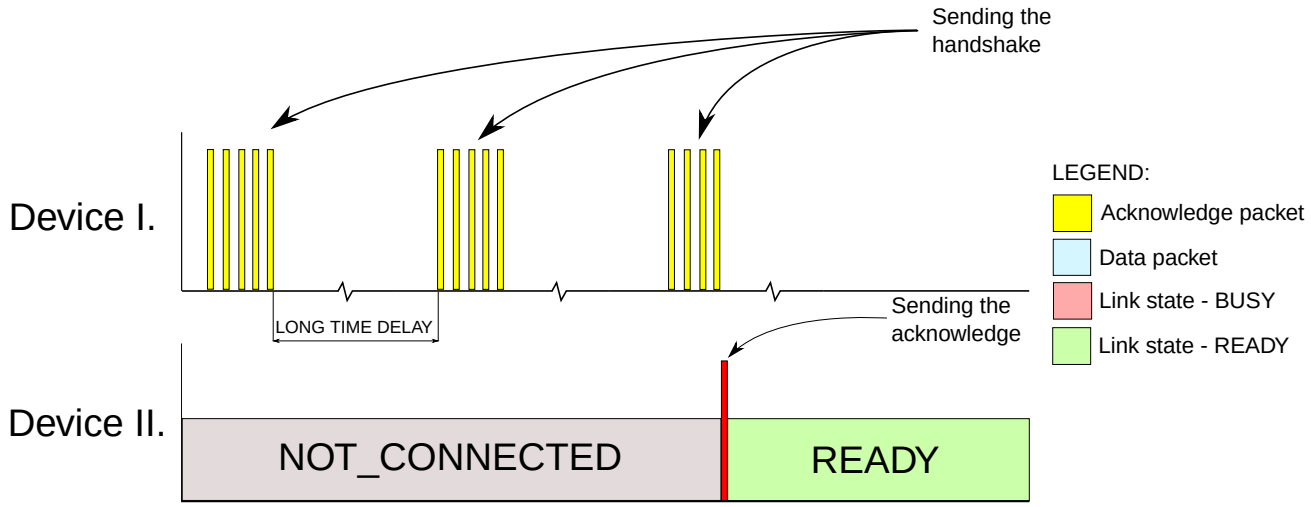

Figure 4. Illustration of the synchronization between two devices

The second figure, Figure 5 shows a sending data packet and receiving the acknowledge packet between the devices.

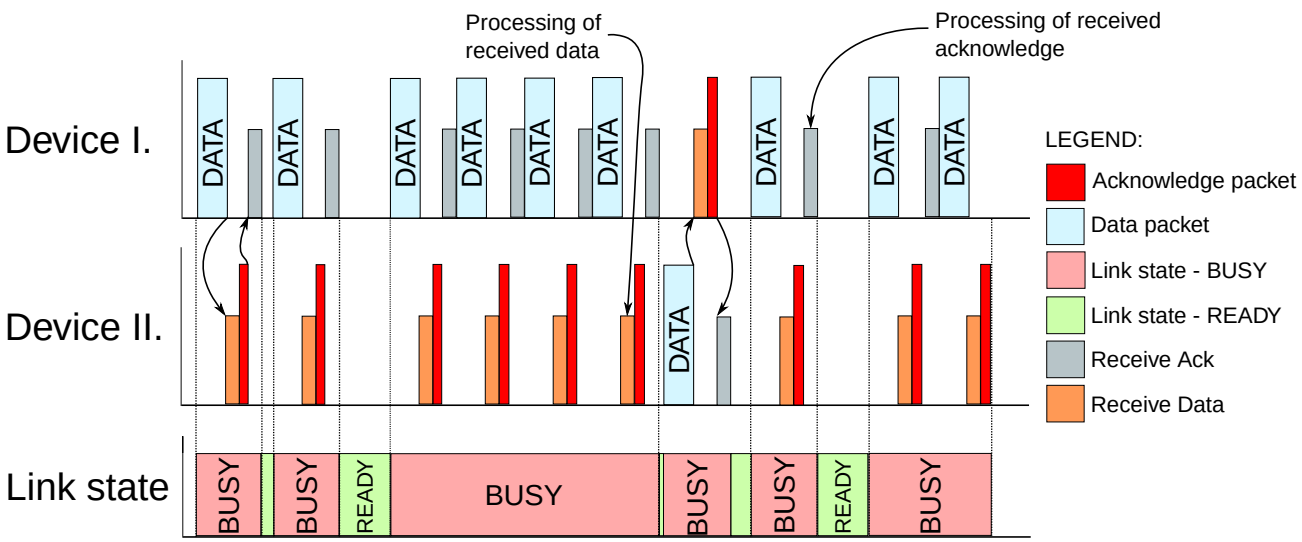

Figure 5. Illustration of two-way data transfer with actual link states

The third figure, Figure 6 shows the feature of the drive that can delay receiving the message by using the non-acknowledge packet.

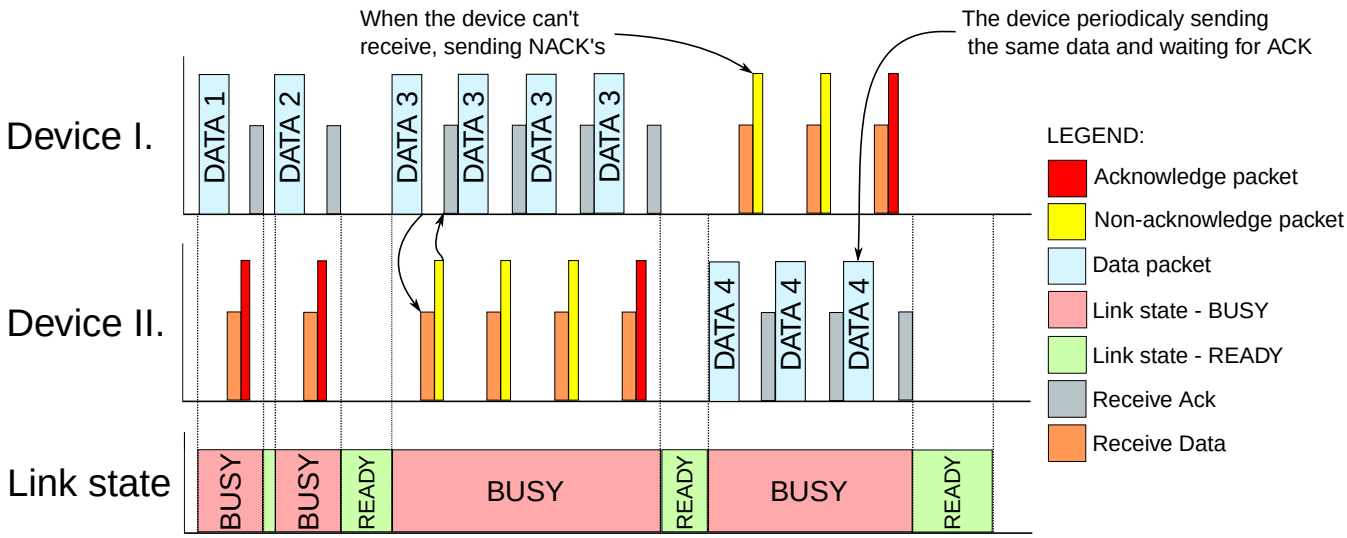

Figure 6. Illustration of two-way data transfer with a non-acknowledge protocol feature

4 Driver Overview

The driver is supplied in four C code files: rf_comm.c, rf_comm.h, rf_com_cfg.h, and rf_comm_private.h. This driver is configured by making selections of the specific request in file rf_com_cfg.h. This high level RF_comm driver primary uses the low level SMAC function, for more details refer to the user guide *Simple Media Access Controller* (SMAC).

The internal functions are located in file rf_comm.c. The description of the internal driver function is shown in the table below.

Table 2. Description of the internal function

Internal Driver Functions	Function
RF_Comm_Init	Sets the initial configuration for the rf_com (SMAC, radio, and MCU initialization). Must be called before using any other driver service.
RF_Comm_Check_device	Searches the opposite device.
RF_Comm_SendAck	Sends a character of Acknowledge (‘A’).
RF_Comm_SendNack	Sends a character of non-Acknowledge (‘N’).
RF_Comm_SendBuffer	Sends a complete data packet.
RF_Comm_TX_done	Signals the end of broadcasting.
RF_Comm_ChangeLinkState	Changes the link state between these three states (not connected, ready, or busy).
RF_Comm_TxGenerator	Sub–function that communicates with SMAC and sends a final data packet.
RF_Comm_Poll	The function for pulling any of six events internal driver.

4.1 API Control Functions

This section describes the structure and behavior of the API control functions (RF_Comm_Init, RF_Comm_TxBuff, RF_Comm_TxBuffFlush, and RF_Comm_TxBuffPending). The main feature is communication between the user application and the RF driver core. These functions are available for the user and must be implemented in the main program. The API control functions are described in detail in the following chapters. The structure of the callback functions is shown in Figure 7
.

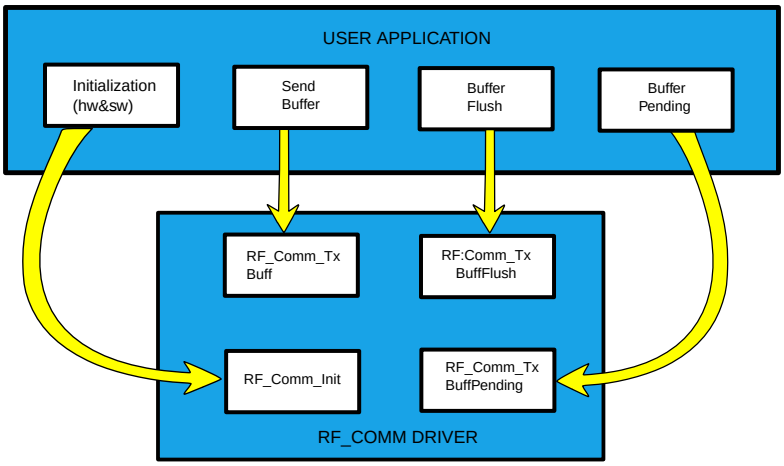

Figure 7. Architecture control functions

4.1.1 RF_Comm_Init

Syntax:

- void RF_Comm_Init(void);

Parameters:

- None

Return:

- None

Description:

- Consists of complete initialization (MCU, radio, and TX packet global SMAC). This function then changes the link state to NOT_CONNECTED and initializes the internal timer (Timer_init). The event check device is called at the end of the function and begins searching the other device.

4.1.2 RF_Comm_TxBuff

Syntax:

- byte RfComm_TxBuff(byte *ptr_data,byte len);

Return:

- len—Number of free bytes in the RF buffer

Parameters:

- ptr_data—Start address of data
- len—Length of data

- If the link state is not ready or the data buffer is not free, this function then returns to a value of 0. If the link state is ready this function then copies data from the start address (byte *ptr_data) to a specific address in the data buffer. Then called is function RfComm_TxBuffFlush, this buffer is sent and cleared.

4.1.3 RF_Comm_TxBuffFlush

Syntax:

- byte RfComm_TxBuffFlush(void);

Parameters:

- None

Return:
- If the link state is not ready it returns to 0 otherwise it returns to 1.

Description:
- This function calls the RF_Comm_ChangeLinkState and changes the actual link state to BUSY. The value Packet_ID is incremented for sending the next buffer. The maximum attempt to send a buffer is set up to 20 (MAX_SEND_BUFFER_CNT = 20).

4.1.4 RF_Comm_TxBuffPending

Syntax:
- byte RfComm_TxBuffPending(void);

Parameters:
- None

Return:
- Return free size of RF buffer

Description:
- This function is only for information about free size in the RF buffer.

4.2 API Callback Functions

This section describes the structure and behavior of API callback functions. The callback functions represent direct input to the main user application and are used together with API control functions. One of these functions is used for signaling the internal link state and the second for data received. The callbacks are defined and can be modified in file RF_comm_cfg.h. The structure of the callback functions is shown in Figure 8 .

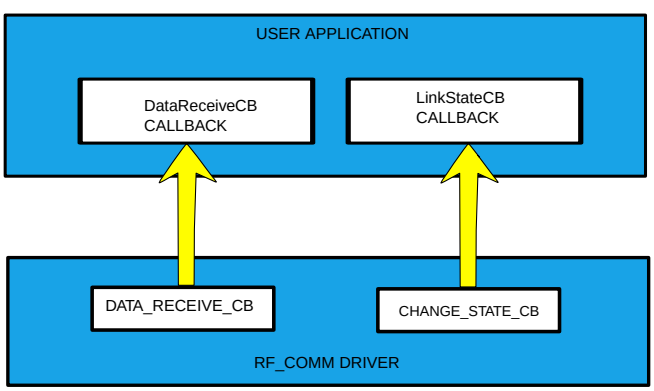

Figure 8. Architecture of callback functions

4.2.1 RF_Comm_LinkStateChangeCB

Syntax:

- `void RfComm_LinkStateChangedCB(LINK_STATE new_state);`

Parameters:

- new_state—Internal RF driver link state, which can be in state NOT_CONNECTED, READY, or BUSY. This type of variable (new_state) is enumerated LINK_STATE and defined as three states (NOT_CONNECTED, READY, and BUSY).

Description:

- Is called by the RF driver if any changing of link state occurs. Supports callback function to the main user application and informs about changing actual link state. This feature is used for other peripheral communications (USB, SCI, and so on.).

4.2.2 RF_Comm_DataReceivedCB

Syntax:

- `void RfComm_DataReceiveCB(byte * ptr_receive_buffer, byte length);`

Parameters:

- ptr_receive_buffer—Pointer shows where the first address is in the receive buffer length—Length of received data

Description:

- Is called by the RF driver, if any data is received. For data receiving in the main driver, this callback function uses internal SMAC function MCPSDataIndication. All settings and properties are set in the user configuration header file rf_com_cfg.h.

5 Using the Driver

5.1 General Conversion

The driver includes two configuration header files.

One of these files:

`rf_comm_cfg.h`

Is used for setting features and parameters of the main transfer. For example, acknowledged a non–acknowledged transfer, receive CB, link state CB, and timer. This file is available for user application.

The second header file:

`rf_comm_pr.h`

Provides the information about the packet structure and a list of used events, defines, functions, and so on. This file must not be modified by the user. For more information refer to Section 3 .

5.2 Setup and Initialization

Before the driver can be used some initial set up is necessary. This service is performed in the RF_Comm_Init. Consequently, the function RF_Comm_Init must be called before using any other driver service. After the driver has been initialized the messages can be sent or received and any function can be used. This initialization function must be used only in the user application. The following code blocks show a typical structure of the driver initialization.

```
void RF_Comm_LinkStateChangedCB(LINK_STATE new_state)
{
  // user declaration
}
```

```
void RF_Comm_DataReceiveCB(byte * ptr_receive_buffer,byte length)
{
  // user declaration
}
void main(void)
{
 RF_Comm_Init(); // initialization
 EnableInterrupts;

  for(;;)
  {
    RF_Comm_Poll();
    __RESET_WATCHDOG(); /* feeds the dog */
  }
}
```

The core of the RF_Comm driver is created by the function RF_Comm_Poll. This function is an internal state machine that can process defined events. The driver consists of six events and these events are defined in the header file RF_Comm_private.h:

Table 3. Description of the events

Num.	EVENT	VALUE	CALLED FUNCTION
1.	EVENT_TXDONE	0x01	RF_Comm_TX_done()
2.	EVENT_INIT	0x02	RF_Comm_Init()
3.	EVENT_SEND_ACK	0x04	RF_Comm_SendAck()
4.	EVENT_SEND_BUFF	0x08	RF_Comm_SendBuffer()
5.	EVENT_SEND_NACK	0x10	RF_Comm_SendNack()
6.	EVENT_CHECK	0x20	RF_Comm_Check_device()

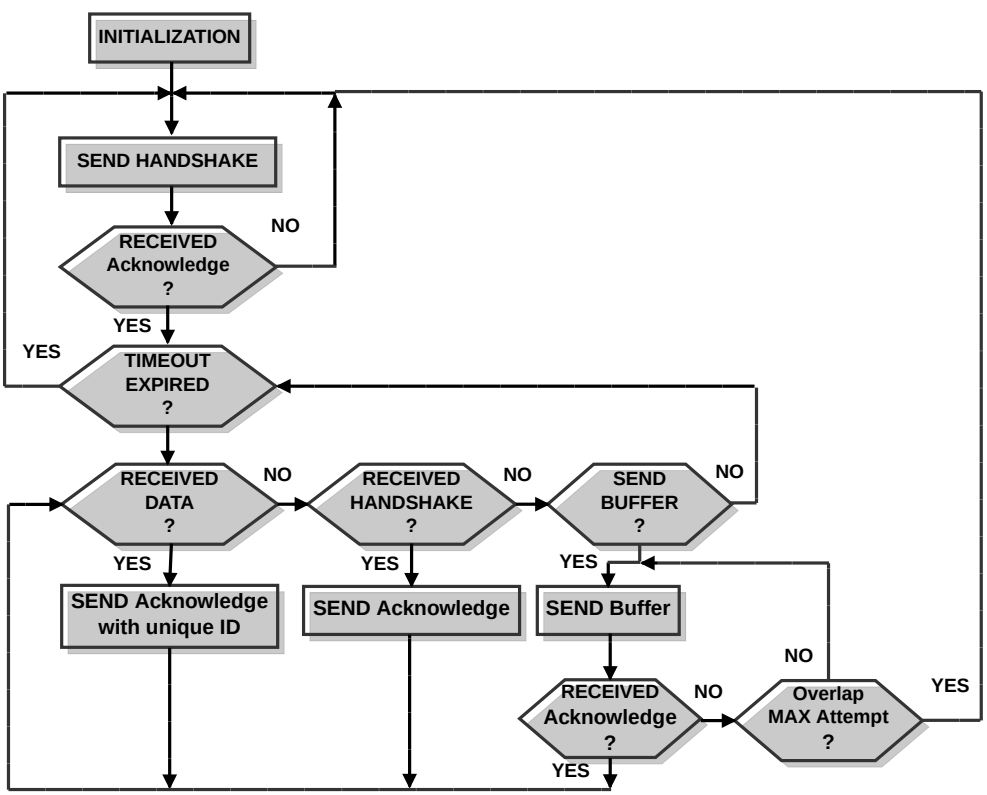

Figure 9. Structure of acknowledge communication protocol

6 Adding the RF Driver to an Application

To add the RF driver to an application:

1. Copy the files rf_comm.c, rf_comm.h, rf_com_cfg.h and rf_comm_private.h to the source directory used for the project.
2. Add the rf_comm.c, rf_comm.h, rf_com_cfg.h, and rf_comm_private.h driver files to the project. In CodeWarrior, right click on **Sources folder**, then select **Add Files**.
3. Add the line #include rf_comm.h to the main application program file.
4. Add the following callback functions to the main application:
   - void RF_Comm_LinkStateChangedCB (LINK_STATE new_state);
   - byte RF_Comm_DataReceiveCB (byte * ptr_receive_buffer,byte length);

```
void RfComm_LinkStateChangedCB(LINK_STATE new_state)
{
 /* add the sequence of source code that will respond to a change of the link state here
*/
}
/* WHEN DATA RECEIVE OCCURS IT IS CALLED, CALLBACK IN THE MAIN PROGRAM */

byte RfComm_DataReceiveCB(byte * ptr_receive_buffer,byte length)
 {
  /* add the sequence of source code that will process the received data here (example:
sending over USB or SCI, etc.) */
 }
```

5. Add the function RF_Comm_Poll() to the main application, the example is shown in the following block. This function must be implemented in each project that uses the RF driver. This function creates the main state machine of the RF driver.

```
void main(void) {

   MCU_Init();       // set up the requested clock on the MCU
   RfComm_Init();    // initialization of the Radio module
   LED_INIT_MACRO
   EnableInterrupts;

  for(;;)
   {
    RfComm_Poll();    /* internal state machine of the RF driver (must be
                  implemented in the main function) */
    /*INCLUDE APPLICATION CODE HERE*/
    __RESET_WATCHDOG();
   }
}
```

6. Modify rf_com_cfg.h to define the parameters required by the application. In this file, the timers and features of the wireless data transfer (acknowledge or no–acknowled transfer) are defined. The example of configuration rf_com_cfg.h file is shown in the following code block.

```
/* THIS SECTION IS TO SET–UP FEATURES FOR TRANSFER */

#define UNSUPPORTED 0
#define SUPPORTED   1

#define DATA_ACK_RECEIVE   SUPPORTED    /* ACKNOWLEDGE OR NO-ACKNOWLEDGE TRANSFER - feature
 for receive section */
#define DATA_ACK_SEND      SUPPORTED    /* ACKNOWLEDGE OR NO-ACKNOWLEDGE TRANSFER - feature
 for transmit section */

/* name of receive callback in main.c */
#define DATA_RECEIVE_CB RfComm_DataReceiveCB

/* name of Link state callback in file main.c */
```

```
    #define RFCOMM_CHANGE_STATE_CB RfComm_LinkStateChangedCB

    #define RFC_TIMER_INIT  {(void)TPM2SC;\
                             TPM2CNT = 0;\
                             TPM2SC = TPM2SC_TOIE_MASK | TPM2SC_CLKSA_MASK | TPM2SC_PS_MASK;\

                             TPM2MOD = 4688; /* 25ms time base*/ \
                            }

    #define RFC_CLEAR_TOF   { (void)TPM2SC;\
                              TPM2SC_TOF = 0;\
                            }
    #define RFC_TIMER_CNT  TPM2CNT

    #define RFC_DEBUG_ID 'T'
```

The files are now added to the project and ready to begin using the RF driver.

7 Example of RF Application

This example uses the file rf_comm_cfg.h. The configuration is shown in the following code block and has been configured to run on the ZSTAR3 USB stick.

```
#include "rf_comm.h"  // include the header RF driver's header file

void RfComm_LinkStateChangedCB(LINK_STATE new_state)
{
  // insert requested source code here
}
byte RfComm_DataReceiveCB(byte * ptr_receive_buffer, byte length)
{
  //  insert requested source code here
}

void main(void)
{
    USB_Init();   // enable D+ pull up, configure RESET & 3.3V reg.
    LED_INIT_MACRO     // Init LEDs
    USB_Enable();
    EnableInterrupts;       // Enable USB only

    while (configready==0)  // wait for USB ready
      COPCTL = 0;

   RfComm_Init();  // initialization of the RF module

  for(;;)          // infinite cycle
  {
   RfComm_Poll();   // internal state machine
   __RESET_WATCHDOG(); /* feeds the dog */
  }
}
```

The example for using control functions is shown in next code block. This implementation is for the HC08JW32 with a USB 2.0 module, and the combination of USB drivers with an RF driver creates a half duplex serial transfer over a wireless 2.4GHz. This is one of many variants where the driver can be used.

```
void USB_RxReadyCB2(uchar cnt)       // call-back function for the USB driver
{
  switch(RfComm_GetLinkState())      /* switch condition with signalization of link
                                 state */
  {
    case READY:                      // if link state READY
      if(RF_Comm_TxBuffPending() > cnt)
      {
```

```
        RF_Comm_TxBuff(EP2_BASE_ADR, cnt);
        if((cnt<16) || (RfComm_TxBuffPending() < 16))
         {
           RF_Comm_TxBuffFlush(); // transmit the data packet by function Flush
         }
        AckUsbData();            // send the acknowledgement
       }
      else
      {
        RF_Comm_TxBuffFlush();          // transmit the data packet
        usb_length = cnt;
      }
      break;
    case BUSY:                        // if link state BUSY
      usb_length = cnt;
      break;
    }
}
```

The following code block shows an example of using call-back function LinkState in the main program.

```
void RF_Comm_LinkStateChangedCB(LINK_STATE new_state)
{
 if(new_state != NOT_CONNECTED)
  {
   LED2 = LEDON;      // if the link state is not NOT_CONNECTED LED2 = ON
  }
 else
  {
   LED2 = LEDOFF;     // if the link state is NOT_CONNECTED or BUSY LED2 =  OFF
  }
}
```

The following code block shows an example of using call-back function DataReceive in the main program. This short example can send received data to the PC through the USB.

```
byte RF_Comm_DataReceiveCB(byte * ptr_receive_buffer, byte length)
{
  if(USB_TxBuffPending3())   /* Function returns number of bytes pending in user        return
 (1);                 buffer */

  else
  {
    USB_TxBuff(ptr_receive_buffer, length);   /* Function copy data from user buffer
    return (0);                                to EP bufer */
  }
}
```

8 References

For more information see the devices reference manuals and the documents listed in the following table.

Table 4. References

Document	Title
User's Guide	Simple Media Access Controller (SMAC)
MCF51QE128RM	ColdFire V1 Family Reference Manual for MCF51QE
MC68HC908JW32	M68HC08 Family Reference Manual for MC68HC908JW32
AN2961	Software drivers for MC33696
DRM103	The ZSTAR3 Reference Design Manual

Document	Title
AN3153	Using the Full-Speed USB Module on MC68HC908JW32

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3941
Rev. 0, 01/2010