Freescale Semiconductor
Application Note

Document Number: AN3577
Rev. 0, 01/2008

Creating a USB-to-Wireless Bridge with the MC1319x/20x and ColdFire Processors with USB OTG Module

by: Shen Li
Asia & Pacific Operation Microcontroller Division

This document introduces a USB-to-wireless bridge application based on Freescale ColdFire processors with USB OTG module, such as MCF5222x/5221x device.

MCF5222x/5221x provides a full-speed USB device/OTG module.

The wireless part of the bridge is based on the MC1319x/20x, Freescale's 2.4 GHz ISM band transceiver. In this system, the USB-CDC class device is used for USB connection.

This application note introduces

- System architecture
- Hardware connection
- Software design flow.

Customers can get a concept on how to use the MCF5222x/5221x USB function and implement the wireless connection for ColdFire microcontroller products.

Contents

1 System Introduction

Figure 1 shows the system architecture.

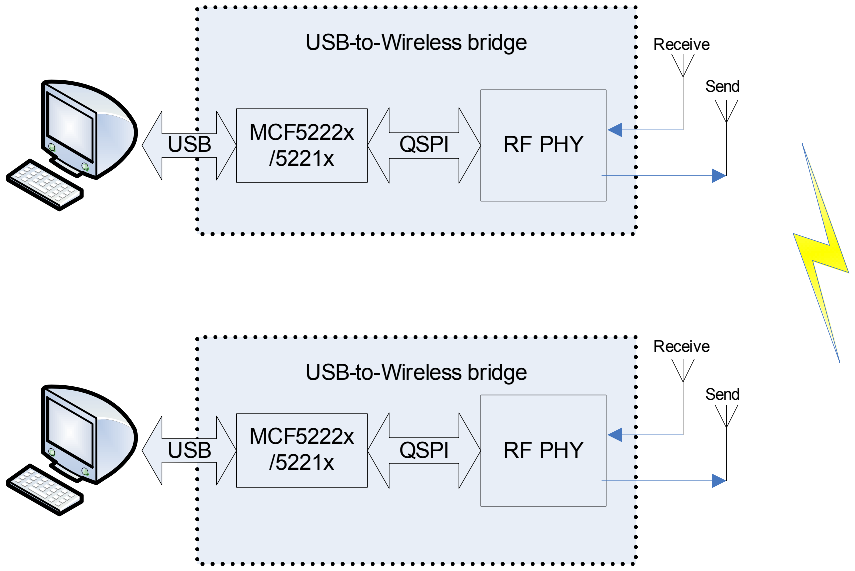

Figure 1. System Architecture

This USB-to-wireless bridge transfers USB data wirelessly back and forth, and it can enable the wireless communication of the computers or other embedded systems. This application is based on Freescale's V2 ColdFire Microcontroller MCF5222x/5221x, a product that provides a full/low-speed USB OTG module. The USB module provides the USB port for the bridge.

This system adopts Freescale's 2.4 GHz ISM band transceiver MC13192 for wireless connection. MC13192 is a short range, low-power transceiver that contains a complete 802.15.4 physical layer (PHY) modem supporting star and mesh networking. When combined with the ColdFire MCU, the MC13192 provides a cost-effective solution for short-range data links and networks. The interface between MCF and MC13192 is accomplished utilizing a four-wire serial peripheral interface (SPI) connection. In this application, we use SMAC stack instead of the complete 802.15.4 stack for the wireless connection.

2 System Hardware

Figure 2 shows the hardware connection of the USB-to-Wireless bridge.

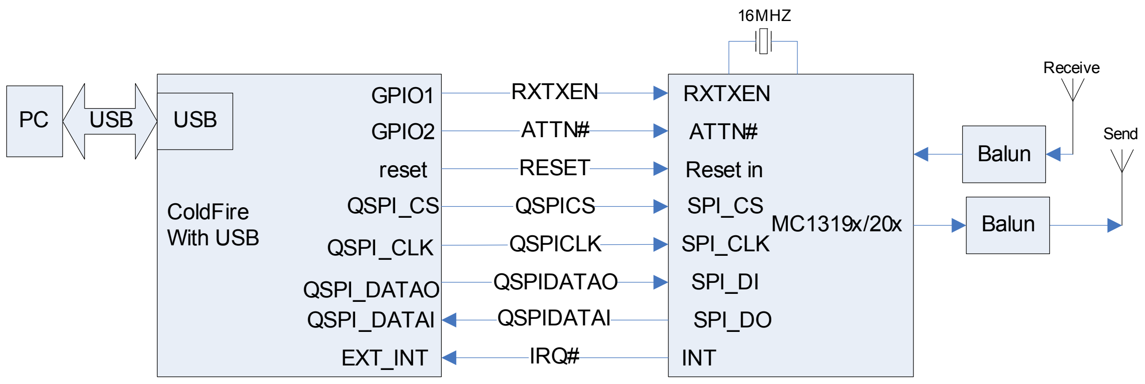

Figure 2. Hardware Connection

2.1 Wireless Port

MC13192 uses 16 MHz crystal oscillator with warp capability as the reference oscillator for the transceiver. At the RF part, the transceiver has a low-noise amplifier, 1.0 mW PA, and differential RF inputs and outputs. In the application, chip baluns are used with balanced PCB antenna for the RF signal. For more details, please refer to *13192RFC Reference Design Zigbee* at www.freescale.com.

2.2 MCU Connection

MCF5222x/5221x connects to the MC13192/20x by QSPI interface and two GPIO pins for control. Figure 2 gives the details of the connection. The QSPI bus includes QSPICS, QSPICLK, QSPIDATAO, and QSPIDATAI.

There are four control signals :

- IRQ# is for the interrupt request from MC13192 to MCU
- RESET is system reset signal.
- RXTXEN is to enable the receival and transfer functions.
- ATTN# is the attention signal for MC13192. When it goes low, it transitions MC13192 from hibernation or doze mode to idle.

MCF5222x/5221x USB module works on 48 MHz frequency. The 48 MHz clock can be provided by MCU clock module or feeded in from USB_ALT_CLK pin. For more information, please refer to M52223EVB from www.freescale.com. The USB module contains 16 endpoints, each configured as IN or OUT.

3 System Software

The USB-to-wireless system software is based on CMX USB device stack and SMAC wireless stack. The CMX USB device stack provides the CDC class and communicates with the PC USB host. And the SMAC provides a reliable wireless connection between two MC13192s.

3.1 CMX USB Stack – CDC class

CMX company provides a free USB stack based on Freescale ColdFire MCF5222x/5221x, and it can be downloaded at www.freescale.com.

The USB stack is maintained by a status machine triggered by the USB interrupt server function usb_it_handler() in usb.c. Figure 3 and Figure 4 show the status machine of CMX USB stack.

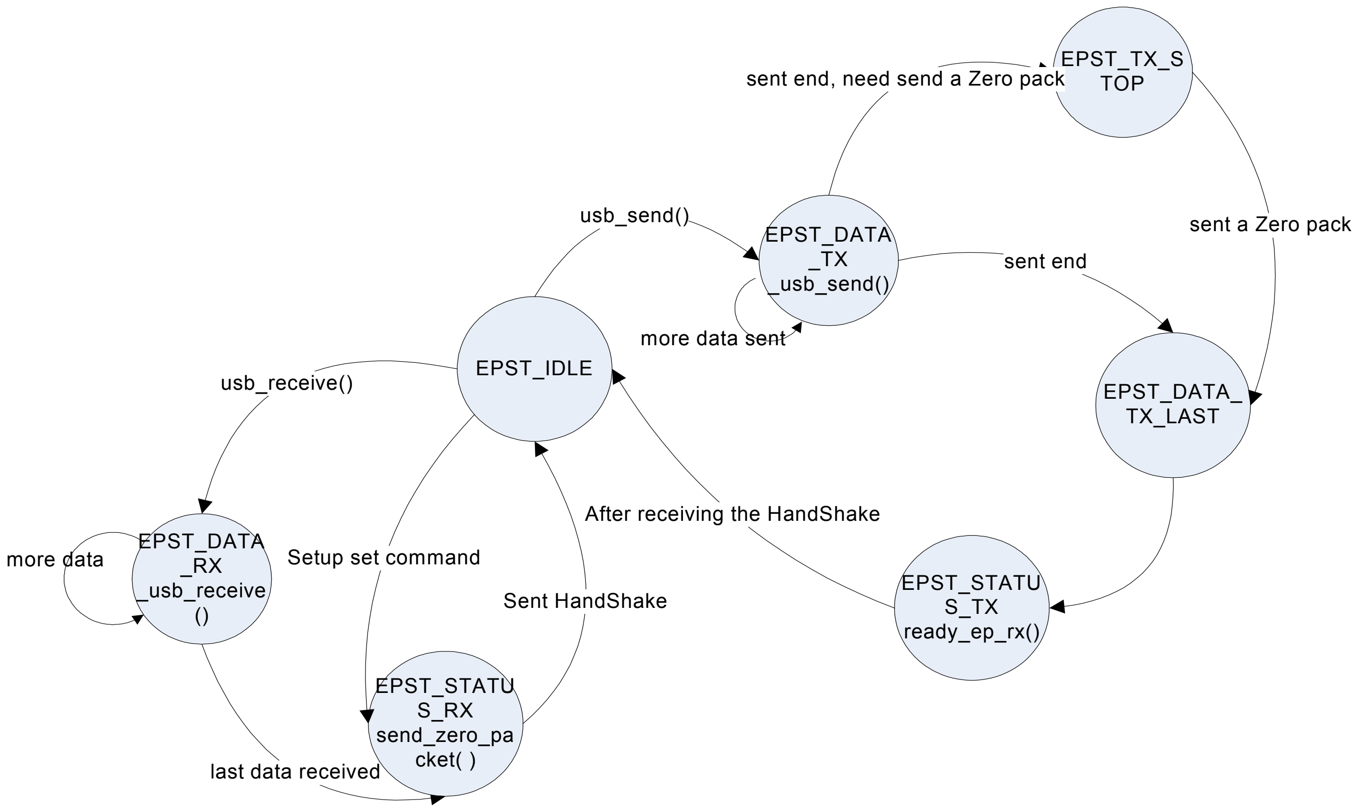

Figure 3. Control Endpoint Status Machine

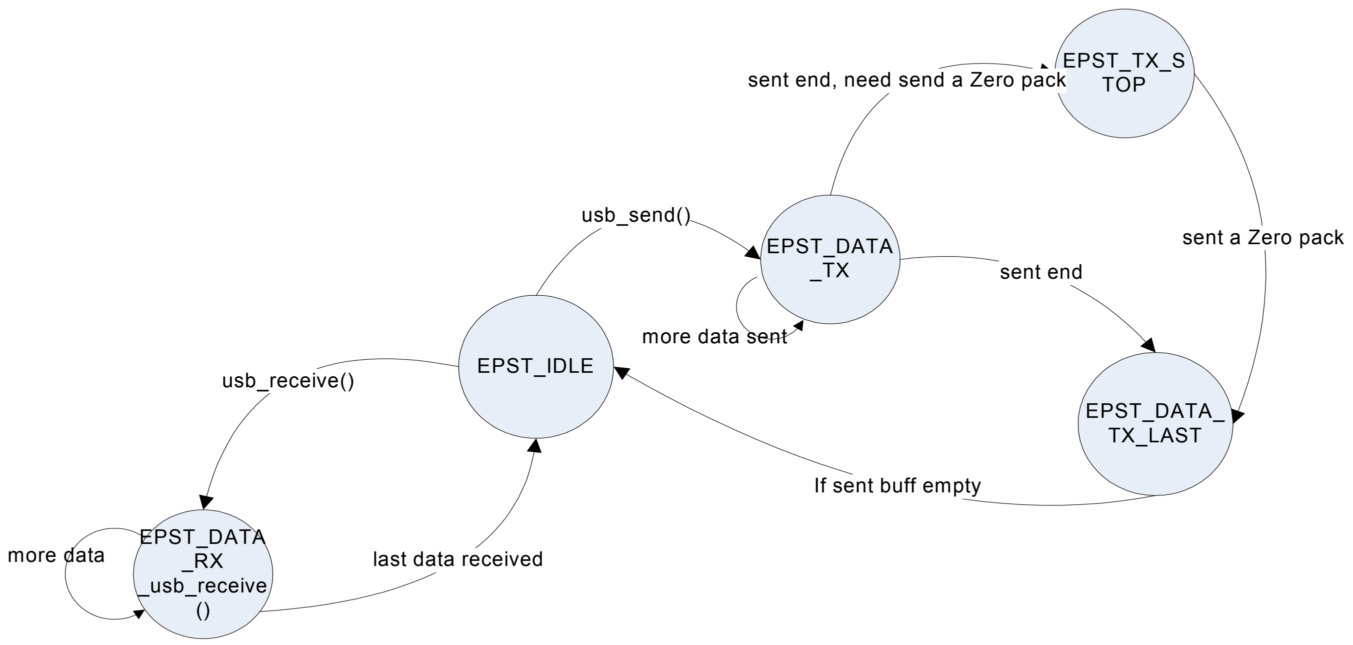

Figure 4. None Control Endpoint Status Machine

With the endpoint status machine running, the system transfers data back and forth through MCU’s USB port with PC host.

In this application, the MCF5222x/5221x uses the CDC device class to communicate with the PC host. The PC host recognizes the MCU as a virtual COM port in the system.

The USB CDC class device provides API function for the application layer:

- USB_INIT() to initiate the USB stack
- CDC_INIT() to initiate the CDC class device
- CDC_LINE_CODING_CHANGED() to indicate that the CDC class communication configuration has been changed
- CDC_INPUT_READY() to indicate that data be transferred from the USB module
- CDC_GETCH() to get the data from the USB module
- CDC_PUTCH() to put a data to the USB module for transferring through the USB

Figure 5 is the CDC class device working flow.

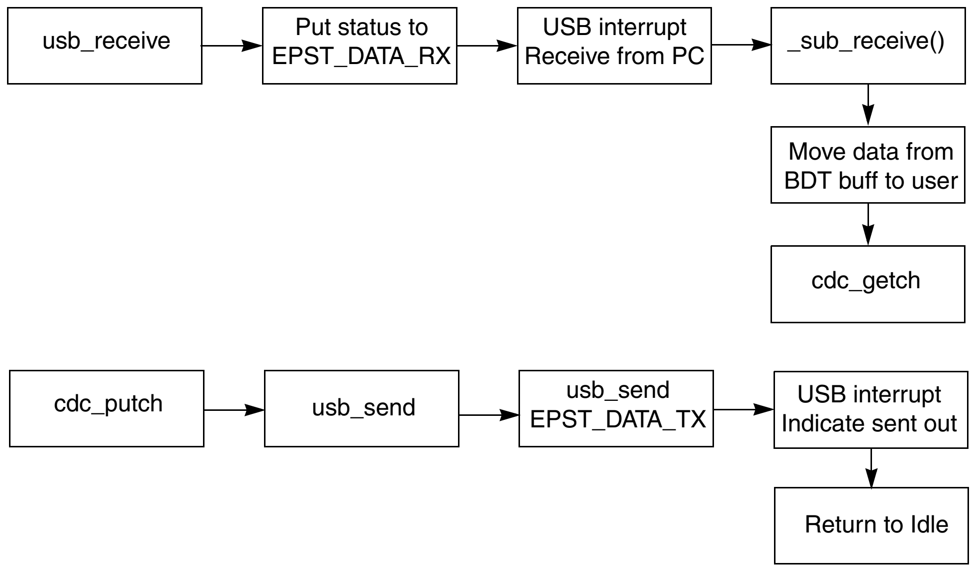
usb_receive
Put status to EPST_DATA_RX
USB interrupt Receive from PC
_sub_receive()
Move data from BDT buff to user
cdc_getch
cdc_putch
usb_send
usb_send EPST_DATA_TX
USB interrupt Indicate sent out
Return to Idle

Figure 5. CDC Class Device Working Flow

Refer to AN3492 (USB and Using the CMX USB) for more details on CMX USB stack.

3.2 SMAC Stack

SMAC (simple media access controller) is a simple ANSI C based code stack available as sample source code, which can be used to develop proprietary RF transceiver applications using the MC1319x. The SMAC works with any HCS08 MCU and ColdFire with an SPI. SMAC for ColdFire MCF5213 stack can be downloaded at www.freescale.com. The available stack is for ColdFire MCF5213, but it can be easily ported to MCF5222x/5221x products because they are the same at architecture and QSPI module. The users only need to change registers and assign GPIO for ATTN# and RXTXEN signal.

SMAC provides the application layer a set of API functions:

- MCPSDataRequest to send data through wireless
- MLMERXEnableRequest to enable receival function
- MLMERXDisableRequest to disable receival function
- MLMEHibernateRequest to place the MC13192 into hibernate mode
- MLMEDozeRequest to place the MC13192 into doze mode
- MLMEWakeRequest to bring MC13192 out of low power mode
- MLMESetChannelRequest to set the actual frequency the MC13192 transmits and receives on
- MLMEEnergyDetect to start an energy detection and return the energy value
- MLMELinkQuality to return the link quality from the last received packet
- MLMEMC13192FEGainAdjust to adjust the energy by tuning the AGC of MC13192
- MLMEMC13192PAOutputAdjust to adjust the output power of the transmitter
- MLMEMC13192SoftReset to perform the soft-reset of the MC13192
- MLMESetMC13192TmrPrescale to change the rate on which the MC13192 timers operate
- MLMEMC13192XtalAdjust to adjust the MC13192 reference clock by a trim value

- MCPSDataIndication to be called at receiving data, and to be filled by user application

The most important functions are MCPSDataIndication() and MCPSDataRequest().

- MCPSDataIndication is a callback function called in the interrupt server function of MC13192. In this function, user is required to write his own code to do further actions on the received data.
- MCPSDataRequest() is used to send user data out through wireless. It is a block function and it won't return until the sending action completes.

For more information about SMAC of Freescale, refer to the *Simple Media Access Controller User Guide*, which can be downloaded from www.freescale.com.

3.3 Software Architecture

The USB-to-wireless system software flow is shown in Figure 6

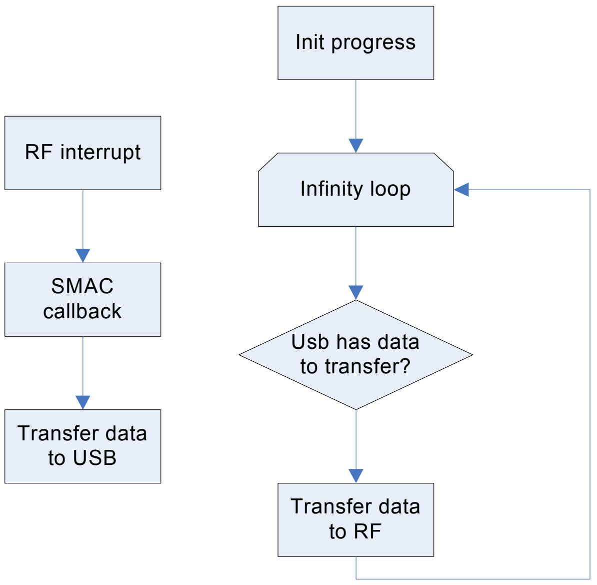

Figure 6. System Software Flow

The bridge software transfers data between USB and wireless. In the software flow chart, the program steps into infinity loop after initializing progress. In the loop, the program checks the USB data. If the USB data is available, the USB data is transferred to wireless port. When the wireless port received data from external, MC13192 raises an interrupt. In the interrupt server, SMAC callback function is called and the data is transferred to USB port. The whole flow is maintained by a status machine.

3.3.1 Initiation Progress

Here is the initialization progress code

```
…
hw_init();                                           /*initiate hardware*/
usb_init((2<<3) | 2, 0);  /*USB stack initialization, USB irq is level 2, priority = 2. */
cdc_init();                                          /*USB CDC class initiation*/
PortInit();                                          /*QSPI init*/
```

```
MC13192Init();                                  /* MC13192 init*/
PLMESetTrxStateRequest(IDLE_MODE);              /*Init SMAC TX/RX status to IDLE*/
MLMESetChannelRequest(channel);                 /*Set RF channel*/
MLMEMC13192PAOutputAdjust(power_level);         /* set RF power level*/
tx_pkt1.pu8Data = tx1_data_buffer;
rx_pkt1.pu8Data = rx1_data_buffer;
app_status = APP_RECV;
MLMERXEnableRequest(&rx_pkt1,0);                /* start receive, timeout disable*/
…
```

3.3.2 System status machine

In this application, the main data flow is maintained by a status machine. Figure 7 shows the machine.

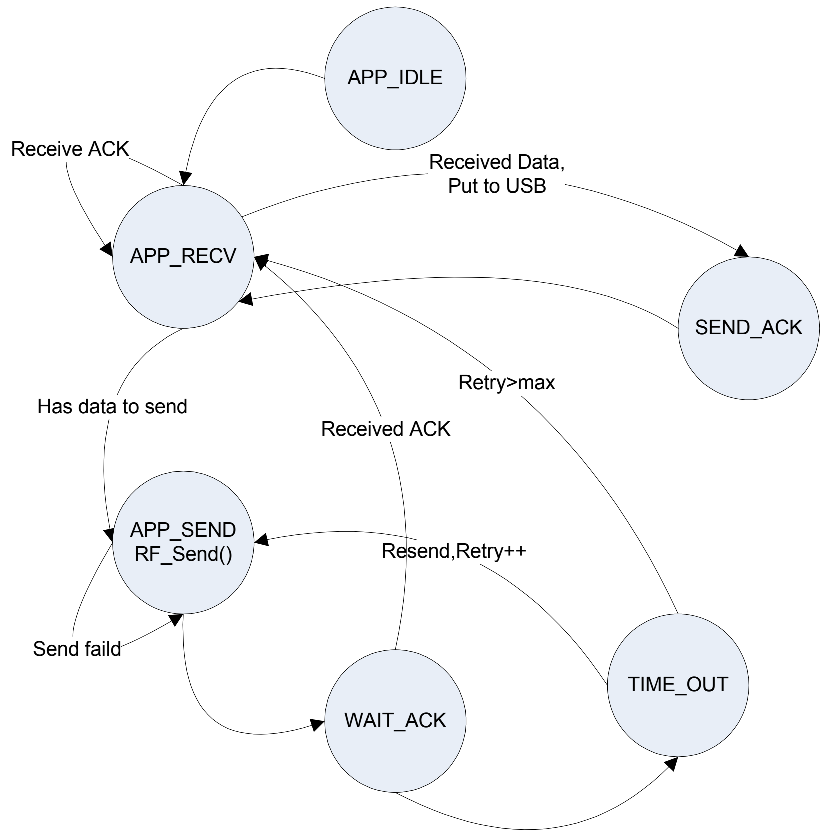

Figure 7. System Status Machine

After receiving USB data, the system enters APP_SEND status and calls RF_Send() to send out the data by wireless port. After that, it enters WAIT_ACK status to wait for the acknowledge signal sent by peer end wireless receiver. At the same time, a timer in MC13192 is started. If no acknowledge signal has been received before the timer times out, the system resends the data. If the system receives the acknowledge signal, the system enters APP_RECV status. If the system resends the data for the maximum times and has not received the acknowledge signal, it enters APP_RECV status.

The status machine code in USB-to-wireless direction is demonstrated as follows.

```
while(1)
```

```
{
      if (cdc_input_ready())
      {
          c=(UINT8)cdc_getch();
          uart_putch((hcc_u8)c);
          app_status = APP_SEND;
      }
      switch(app_status)
      {
          case APP_IDLE:
              app_status = APP_RECV;
              break;
          case APP_RECV:
              break;
          case APP_SEND:
              if(RF_Send((unsigned char)c,0)!= SUCCESS)
                  app_status = APP_SEND; //retry
              break;
          case SEND_ACK:
              //these line are only for delay
              i++;
              MCF_GPIO_PORTTC = (UINT8)(MCF_GPIO_PORTTC ^ MCF_GPIO_PORTTC_PORTTC0);
              if(i>= 1000)
              {
                  RF_Send('a', 1);
                  i = 0;
              }
              break;
          case WAIT_ACK:
              break;
          case TIME_OUT:
              if(RetryNo< MAX_RETRY)
              {
                  app_status = APP_SEND;
                  RetryNo++;
              }
              else //give up
                  app_status = APP_RECV;
                  MLMERXEnableRequest(&rx_pkt1,0); //start as reciever.
              break;
          default:
              break;
      }
}


UINT8 RF_Send(unsigned char c, unsigned char is_ACK)
{
      UINT8 temp;
      temp = PLMESetTrxStateRequest(IDLE_MODE);
      if(temp != SUCCESS)
          return temp;
      app_status = APP_IDLE;
      if(is_ACK == 0)
      {
                  tx_pkt1.pu8Data[0] = c;
                  tx_pkt1.pu8Data[1] = 0;
```

```
                   tx_pkt1.u8DataLength = 2;
                   temp = MCPSDataRequest(&tx_pkt1);
                   if(temp == SUCCESS)
                   {
                        //PLMESetTrxStateRequest(IDLE_MODE);
                        temp = MLMERXEnableRequest(&rx_pkt1,0x2000); //back to reciever.
                        app_status = WAIT_ACK;//APP_RECV;//
                   }
      }
      else
      {
           tx_pkt1.pu8Data[0] = 'A';
           tx_pkt1.pu8Data[1] = 'C';
           tx_pkt1.pu8Data[2] = 'K';
           tx_pkt1.pu8Data[3] = 0;
           tx_pkt1.u8DataLength = 4;
           app_status = APP_RECV;
           temp = MCPSDataRequest(&tx_pkt1);
      }
      PLMESetTrxStateRequest(RX_MODE);
      return temp;
}
```

For the data from wireless-to-USB, the process is different. When MC13192 receives the data from external device, it raises an interrupt to the MCF5222x/5221x. The interrupt server function is executed. As shown in Figure 7, the system enters SEND_ACK, forwards the received data to USB module, sends the acknowledge signal back, and then returns APP_RECV status. Following is the callback function for data receival interrupt.

```
void MCPSDataIndication(tRxPacket *rx_pkt)
{
      unsigned char i;
      if(rx_pkt->u8Status == SUCCESS)
      {
           if(rx_pkt->pu8Data[0] == 'A' && rx_pkt->pu8Data[1] == 'C' && rx_pkt->pu8Data[2] ==
           'K')
           {
                if(app_status ==  WAIT_ACK)
                {
                     app_status = APP_RECV;
                }
                MLMERXEnableRequest(&rx_pkt1,0); //receiver.
           }
           else/*Packet received*/
           {
                for(i=0;i<rx_pkt->u8DataLength;i++)
                cdc_putch((rx_pkt->pu8Data)[i]);
                app_status = SEND_ACK;
                return;
           }
      }
      else if(rx_pkt->u8Status == TIMEOUT)
      app_status = TIME_OUT;
}
```

4 Conclusion

With free software CMX USB stack and SMAC stack, a reliable USB-to-wireless bridge can be easily realized on Freescale ColdFire MCF5222x/5221x with MC13192/20x. With this bridge, you can easily enable the PC or any wireless connection of the embedded system. This application source code can be downloaded at www.freescale.com.

Document Number: AN3577
Rev. 0
01/2008