Freescale
User's Guide

Document Number: 802154DPPERAUG
Rev. 1, 11/2013

802.15.4 Media Access Controller (MAC) Dual PAN Packet Error Rate (PER) Application Users Guide

1 Application Overview

This chapter introduces users to the MAC Packet Error Rate (PER) Demo application. The MAC PER Demo source code example applications include the Freescale MAC Dual PAN library, which is available only for MC1324x based platforms such as the MKW2x wireless system. The demo applications are included in Kinetis MAC Codebase, starting with the BeeKit Kinetis MAC Codebase 4.0.0. Updates for the Codebase and for applications can be downloaded from the Freescale ZigBee site at: http://www.freescale.com/zigbee

The scope of these applications is to demonstrate the MAC Packet Error Rate (PER) in case of Dual Pan Auto mode, using different settings for the Dual Pan Dwell timer.

Contents

1.1 Source File Hierarchy

Each MAC PER Demo represents a separate project in the MAC Kinetis Codebase. Source files for each MAC PER Demo application are found after exporting each project from BeeKit at the following path:

```
..\Project name\Application\Source
```

Where `Project name` is the name of the BeeKit project (e.g. MAC Packet Error Rate Demo End Device). The MAC PER Demo User's Guide focuses on the 802.15.4 Standard and Freescale specific issues only. It does not explain in detail the state machine that the code contains.

The application source files have the same name inside the projects (`MApp.c`). The project name is the one which differentiates each application.

The following projects are available in BeeKit. Each new project adds additional functionality until a fully functional application has been developed for both a coordinator and a device. Project names with ending in '(Coordinator)' are demonstrating coordinator behavior and project names with ending in '(End Device)' demonstrate device behavior.

- **MAC Packet Error Rate Demo Coordinator - Dual Pan —** This application builds upon the MyWirelessApp Demo Coordinator application**.** In this application, the coordinator is in continuous Rx, and it just receives data from other devices.
- **MAC Packet Error Rate Demo End Device —** This application builds upon the MyWirelessApp Demo End Device application. In this application, the device sends a burst of packets using direct transmission.

2 Setup and Initialization

The initialization procedure is done similar as in MyWirelessApp Demos. For more information please refer to 802154MWAUG.pdf.

2.1 Initialization

The application initialization is done in MApp_init function shown below:

```
void MApp_init(void)
{
  /* The initial application state */
  gState = stateInit;
  /* Initialize the MAC 802.15.4 extended address */
  Init_MacExtendedAddress();
#if gDualPanEnabled_d
  Init_MacExtendedAddress_PAN1();
#endif
  /* register keyboard callback function */
  KBD_Init(App_HandleKeys);
  /* initialize LCD Module */
  LCD_Init();

  /* Initialize the UART so that we can print out status messages */
#ifdef gUart_PortDefault_d
  Comm_SetBaud(UartX_DefaultBaud);
  Comm_SetRxCallBack(UartRxCallBack);
```

```
#else
#error No UART have been enabled!
#endif

  /* Prepare input queues.*/
  MSG_InitQueue(&mMlmeNwkInputQueue);
  MSG_InitQueue(&mMcpsNwkInputQueue);
#if gDualPanEnabled_d
  MSG_InitQueue(&mMlmeNwkInputQueue_PAN1);
  MSG_InitQueue(&mMcpsNwkInputQueue_PAN1);
#endif

  /* Enable MCU interrupts */
  EnableInterrupts();

  /*signal app ready*/
  Led1Flashing();
  Led2Flashing();
  Led3Flashing();
  Led4Flashing();

  CommUtil_Print("\n\rPress any switch on board to start running the application.\n\r",
gAllowToBlock_d);
  LCD_ClearDisplay();
  LCD_WriteString(1,"Press any key");
  LCD_WriteString(2,"to start.");
}
```

The `MApp_init` function is called in the `main` function. The `Init_802_15_4()` is called in `main`, before calling the `MApp_init` function.

Before the Freescale IEEE 802.15.4 MAC layer can be accessed, it must be initialized by calling the Init_802_15_4() function which will initialize both the MAC and PHY layers. The function call initializes internal variables of the modules, resets state machines among other things. Once this function is called, the MAC layer services are available and the MAC and PHY layers are in a known and ready state for further access.

Other modules like the timer, the UART and task scheduler are also initialized in `main` function.

After initialization and setup, the application waits for the user to press a switch on the Coordinator board. The keyboard handler, App_HandleKeys, will send the application task an gAppEvtDummyEvent_c event which will start up the application.

2.2 Implementing SAPs

Before being able to compile the project, it is necessary to create the SAP handlers that process the messages from the MAC layer to the next higher layer. For now, create empty functions and implement the functionality later as shown in this code.

```
/* The following functions are called by the MAC
   to put messages into the Application's queue.
   They need to be defined even if they are not
   used in order to avoid linker errors. */

uint8_t MLME_NWK_SapHandler(nwkMessage_t *pMsg)
```

```
{
  /* This only serves as a temporary way of avoiding a compiler warning.
     Later this will be changed so that we return gSuccess_c instead. */
  return pMsg->msgType;
}
uint8_t MCPS_NWK_SapHandler(mcpsToNwkMessage_t *pMsg)
{
  /* This only serves as a temporary way of avoiding a compiler warning.
     Later this will be changed so that we return gSuccess_c instead. */
  return pMsg->msgType;
}
uint8_t ASP_APP_SapHandler(aspToAppMsg_t *pMsg)
{
  /* This only serves as a temporary way of avoiding a compiler warning.
     Later this will be changed so that we return gSuccess_c instead. */
  return pMsg->msgType;
}
```

By default, the Coordinator uses the Dual Pan Feature, so the following SAPs must also be implemented:

```
uint8_t MLME_NWK_PAN1_SapHandler(nwkMessage_t *pMsg)
{
  /* This only serves as a temporary way of avoiding a compiler warning.
     Later this will be changed so that we return gSuccess_c instead. */
  return pMsg->msgType;
}

uint8_t MLME_NWK_PAN1_SapHandler(nwkMessage_t *pMsg)
{
  /* This only serves as a temporary way of avoiding a compiler warning.
     Later this will be changed so that we return gSuccess_c instead. */
  return pMsg->msgType;
}
```

Typically, the SAP handlers buffer the messages in a queue and then send an event to the application task to process them.

3 Starting and Joining a PAN

Once the MAC and PHY layers are initialized (See Section 2.1, “Initialization) it is now safe to access the MCPS, ASP, and MLME services of the MAC layer. Start by accessing the MLME. The example code for this is found in the `MApp.c` source file.

The units are now ready to start up a PAN. First, set up a PAN coordinator because all IEEE 802.15.4 Standard PANs must have a PAN coordinator.

The procedure used for starting and Joining a PAN, is identical to the one used in the MyWirelessAppDemo application. For more details about Starting and Joining a PAN, please refer to Chapter 4 of 802154MWAUG.pdf.

Since the Dual Pan feature is used by the Coordinator application, after both PANs have been started, the App_StartDualPan() function is called. This functions enabled the Dual Pan Auto mode, with a Dwell time of 3.5ms.

```
static void App_StartDualPan()
```

```
{
  mlmeMessage_t Msg;
  mlmeMessage_t *pMsg = &Msg;
  uint8_t value;

  pMsg->msgType = gMlmeSetReq_c;
  pMsg->msgData.setReq.pibAttribute = gMPibDualPanActiveNwk_c;
  value = 0;
  pMsg->msgData.setReq.pibAttributeValue = &value;
  (void) MSG_Send(NWK_MLME, pMsg);

  pMsg->msgType = gMlmeSetReq_c;
  pMsg->msgData.setReq.pibAttribute = gMPibDualPanAuto_c;
  value = TRUE;
  pMsg->msgData.setReq.pibAttributeValue = &value;
  (void) MSG_Send(NWK_MLME, pMsg);

  pMsg->msgType = gMlmeSetReq_c;
  pMsg->msgData.setReq.pibAttribute = gMPibDualPanDwell_c;
  value = (mDwellPrescaller &mDualPanDwellPrescallerMask) |
          (mDwellValue << mDualPanDwellTimeShift);
  pMsg->msgData.setReq.pibAttributeValue = &value;
  (void) MSG_Send(NWK_MLME, pMsg);
}
```

4 Monitoring Messages Using the Serial Port

Both the Coordinator and End Devices output messages to the serial port. In order to monitor these messages, the boards must be connected to a PC through the serial interface.

Launch Hyper Terminal (or a Hyper Terminal style program) and select the Comm Port to which the board is connected. Configure the port to use the following settings:

- 115200bps
- 8 data bits
- no parity bits
- 1 stop bit.

4.1 Monitoring messages on Coordinator

1. After powering on the Coordinator, the following message will appear on the Terminal:

```
Press any switch on board to start running the application.
```

2. After pressing a switch on the Coordinator board, the Terminal will output the following messages:

```
MAC Dual Pan PER Demo (Coordinator) application is initialized and ready.

Initiating the Energy Detection Scan
Sending the MLME-Scan Request message to the MAC...Done
Received the MLME-Scan Confirm message from the MAC
ED scan returned the following results:
  [07 07 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ]

Based on the ED scan the logical channel 0x0D was selected for PAN0,
and logical cannel 0x11 was selected for PAN1.
```

```
Starting as PAN coordinator on channel 0x0D and on channel 0x11
Sending the MLME-Start Request message to the MAC...Done
Sending the MLME-Start Request message for PAN1 to the MAC...Done
Started the coordinator with PAN ID 0xBEEF, and short address 0xCAFE.
Started the coordinator with PAN ID 0xDEAD, and short address 0xABCD.

Ready to start MAC Dual Pan PER test.
```

3. Because the device is operating in Dual Pan Auto mode, a configuration menu will also be displayed. The user can configure the value of the Dwell Timer to test different scenarios. This value represent the time spent on a PAN by the Coordinator. After this time expires, the Coordinator changes the Active PAN.

```
      Dual Pan Dwell settings
  +-----------+------------------+
  | PRESCALER |   DWELL TIME     |
  | (timebase)|   (min - max)    |
  +-----------+------------------+
  |  0.5 ms   |  0.5 - 32   ms   |
  |  2.5 ms   |  2.5 - 160  ms   |
  |  10  ms   |   10 - 640  ms   |
  |  50  ms   |   50 - 3200 ms   |
  +-----------+------------------+

Test Parameters:
================
 Dual Pan Dwell prescaller [ms]: 0.5
 Dual Pan Dwell time [ms]:       3.5


Test Menu:
==========
 1: Increase Dwell Prescaller
 2: Decrease Dwell Prescaller
 3: Increase Dwell Time
 4: Decrease Dwell Time

 > Enter your choice:
```

4.1.1 Test Results

After every burst of packets, the Coordinator must receive a special packet containing details about the test. If this last packet is received, the following message will be printed on the terminal, showing the test results and parameters for a specific PAN:

```
    ******************************
    *** Test Completed on PAN1 ***
    ******************************

Test Parameters:
================
```

```
 Dual Pan Dwell prescaller [ms]: 0.5
 Dual Pan Dwell time [ms]:       3.5

Test Results:
=============
 Packets Transmitted: 100
 Packets Recieved:    99
 Packet Error Rate (PER) [%]: 1
```

4.2 Monitoring messages on End Device

1. After powering on the End Device, the following message will appear on the Terminal:

```
Press any switch on board to start running the application.
```

2. After pressing a switch on the Coordinator board, the Terminal will output the following messages:

```
MAC Dual Pan PER Demo (End Device) application is initialized and ready.

Start scanning for a PAN coordinator
Sending the MLME-Scan Request message to the MAC...Done
Found a coordinator with the following properties:
--------------------------------------------------
Address............0xABCD
PAN ID.............0xF00D
Logical Channel...0x11
Beacon Spec.......0xCFFF
Link Quality......0x5E

Associating to PAN coordinator on channel 0x11
Sending the MLME-Associate Request message to the MAC...Done
Successfully associated with the coordinator.
We were assigned the short address 0x0001
```

3. Next the application's menu will be displayed. The user can configure the number of packets to be transmitted, the interval between the packets, the length of the payload and the state of the AckReq bit, to test different scenarios.

```
Test Parameters:
================

 No of packets:         100
 Packet interval [ms]:  10
 Payload len   [bytes]: 22
 Request ACK:           1


Test Menu:
==========

 0: Start MAC PER test
 1: Increase No of packets
 2: Decrease No of packets
 3: Increase packet interval
 4: Decrease packet interval
 5: Increase payload length
 6: Decrease payload length
```

```
7: Toggle request ACK
8: Stop MAC PER test

Enter your choice:
```

5 Software Implementation

This chapter describes the implementation of the functions used for MAC PER testing of the Coordinator and the End Device. For more consistency, the code has been developed based on the MyWirelessApp application set. Freescale recommends that users refer to the MyWirelessApp documentation available from the www.freescale.com/zigbee web site as needed.

5.1 Software implementation on the Coordinator

After the Coordinator has started, the application will print over the communication interface the configuration menu using the AppPrintMenu() function. This menu helps the user confugure the value of the Dwell time used by the Dual Pan Auto mode.

5.1.1 Processing user commands

Every time a character is received over the UART, the AppProcessUartData() function is called. Valid user commands are:

- '1' / '2' to increase/decrease the value of the Dwell Prescaller
- '3' / '4' to increase/decrease the value of the Dwell Timer

5.1.2 Processing packets received OTA

When a MAC MCPS-DATA.Indication is received, the AppProcessPkt() is called. Depending on the message type, the function will increment a counter or print the test results:

```
static void AppProcessPkt(mcpsToNwkMessage_t *pMsgIn, uint8_t pan)
{
  uint32_t tmp;

  /* Check if this is the End of Test message */
  if (pMsgIn->msgData.dataInd.pMsdu[0] == 0x00)
  {
    /* Print the PAN on which the test has finished */
    CommUtil_Print("\n\n\r", gAllowToBlock_d);
    CommUtil_Print("\n\r    ****************************", gAllowToBlock_d);
    CommUtil_Print("\n\r    *** Test Completed on PAN", gAllowToBlock_d);
    CommUtil_PrintDec(pan); CommUtil_Print(" ***", gAllowToBlock_d);
    CommUtil_Print("\n\r    ****************************\n\r", gAllowToBlock_d);

    /* Display only the Test Parameters */
    AppPrintMenu(1);

    CommUtil_Print("\n\n\rTest Results:", gAllowToBlock_d);
    CommUtil_Print("\n\r=============", gAllowToBlock_d);
    FLib_MemCpy(&tmp, &pMsgIn->msgData.dataInd.pMsdu[1], sizeof(tmp));
```

```
    CommUtil_Print("\n\r Packets Transmitted: ", gAllowToBlock_d);
    CommUtil_PrintDec(tmp);
    CommUtil_Print("\n\r Packets Recieved:    ", gAllowToBlock_d);

    if (pan == 0)
    {
      CommUtil_PrintDec(mPktsRecvOnPan0);
      CommUtil_Print("\n\r Packet Error Rate (PER) [%]: ", gAllowToBlock_d);
      tmp = (tmp-mPktsRecvOnPan0)*100 / tmp;
      CommUtil_PrintDec(tmp);

      mPktsRecvOnPan0 = 0;
    }
    else
    {
      CommUtil_PrintDec(mPktsRecvOnPan1);
      CommUtil_Print("\n\r Packet Error Rate (PER) [%]: ", gAllowToBlock_d);
      tmp = (tmp-mPktsRecvOnPan1)*100 / tmp;
      CommUtil_PrintDec(tmp);

      mPktsRecvOnPan1 = 0;
    }
  }
  else
  {
    /* This is a simple data packet. Increase the counters. */
    if (pan == 0)
      mPktsRecvOnPan0++;
    else
      mPktsRecvOnPan1++;
  }
}
```

5.2 Software implementation on the End Device

After the End Device has been associated to the Coordinator, it will output on the terminal the test configuration menu, using the AppPrintMenu() function:

The user's input is processed using the AppProcessUartData() function. Valid user commands are:

1. '0' / '8' to start/stop the PER test
2. '1' / '2' to increase/decrease the number of packets to be transmitted OTA (multiple of 10 pkts)
3. '3' / '4' to increase/decrease the time interval between two packets (multiple of 10 ms)
4. '5' / '6' to increase/decrease the length of the payload (multiple of 1 byte)
5. '7' activate/deactivate ACK request for every transmitted packet.

When the user sends the start command '0', the AppStartTest() function is called. This function initializes the variables used by the PER test (see AppInitPER() function below), and then it starts an interval timer, used for packet transmission.

Depending on the state of the gFillWithRandom_d define, the AppInitPER() function fills the transmit buffer with a predefined character or with random values: The first byte of payload represents the message type, and it should be different from zero in case of the test data packets.

```
static void AppInitPER(void)
{
  uint32_t index;

  for (index=0; index < mPayloadLen; index++)
  {
#if gFillWithRandom_d
    RNG_GetRandomNo((uint32_t*)&mPayload[index]);
    index += 3;
#else
    mPayload[index] = gFillPattern_d;
#endif
  }

  if (mPayload[0] == 0x00)
    mPayload[0]++;

  mCurrentPacketIdx = 0;
}
```

The Callback function of the interval timer checks if there are more data packets to transmit, and if there are, a new buffer is allocated and filled with required info, before it is sent to the MAC.

```
static void AppTransmitPacket (uint8_t tmr)
{
  if (mCurrentPacketIdx >= mNoOfPackets )
  {
    AppStopTest();
    return;
  }

  mpPacket = MSG_Alloc(gMaxRxTxDataLength_c);

  if(mpPacket != NULL)
  {
    mpPacket->msgData.dataReq.pMsdu = mPayload;
    mpPacket->msgType = gMcpsDataReq_c;
    /* Create the header using coordinator information gained during
       the scan procedure. Also use the short address we were assigned
       by the coordinator during association. */
    FLib_MemCpy(mpPacket->msgData.dataReq.dstAddr, mCoordInfo.coordAddress, 8);
    FLib_MemCpy(mpPacket->msgData.dataReq.srcAddr, maMyAddress, 8);
    FLib_MemCpy(mpPacket->msgData.dataReq.dstPanId, mCoordInfo.coordPanId, 2);
    FLib_MemCpy(mpPacket->msgData.dataReq.srcPanId, mCoordInfo.coordPanId, 2);
    mpPacket->msgData.dataReq.dstAddrMode = mCoordInfo.coordAddrMode;
    mpPacket->msgData.dataReq.srcAddrMode = mAddrMode;
    mpPacket->msgData.dataReq.msduLength = mPayloadLen;
    /* Request MAC level acknowledgement of the data packet */
    if (mRequestAck)
      mpPacket->msgData.dataReq.txOptions = gTxOptsAck_c;
    else
      mpPacket->msgData.dataReq.txOptions = 0;

    /* Give the data packet a handle. The handle is
       returned in the MCPS-Data Confirm message. */
    mpPacket->msgData.dataReq.msduHandle = mMsduHandle++;
#ifdef gMAC2006_d
    /* Don't use security */
```

```
    mpPacket->msgData.dataReq.securityLevel = 0;
#endif //gMAC2006_d
    /* Send the Data Request to the MCPS */
    (void)MSG_Send(NWK_MCPS, mpPacket);
    /* Prepare for another data buffer */
    mpPacket = NULL;
    mCurrentPacketIdx++;
  }
  else
  {
    CommUtil_Print("\n\rAlloc failed!", gAllowToBlock_d);
  }
}
```

If no more test packets need to be sent, the AppStopTest() function is called. This function stops the interval timer, and sends one last data packet to the MAC. This message contains test information (number of Tx data packets), and it is used by the Coordinator to print the test statistics. This message can be identified by reading the first byte of payload, which should be 0x00 in this case.

```
static void AppStopTest (void)
{
  TMR_StopTimer(mTimer_c);
  CommUtil_Print("Done.\n\r", gAllowToBlock_d);

  mPayload[0] = 0;
  FLib_MemCpy(&mPayload[1], &mCurrentPacketIdx, sizeof(mNoOfPackets));
  mpPacket = MSG_Alloc(gMaxRxTxDataLength_c);
  if(mpPacket != NULL)
  {

    ...

    mpPacket->msgData.dataReq.msduLength = sizeof(mNoOfPackets) + 1;
    /* Request MAC level acknowledgement of the data packet */
      mpPacket->msgData.dataReq.txOptions = gTxOptsAck_c;

    ...

  }
}
```

NOTE

The last packet of the PER test is always sent with AckReq bit set.

How to Reach Us:

Home Page:
www.freescale.com

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

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
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 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: 802154DPPERAUG
Rev. 1
11/2013