反射内存卡收发示例及常用 API 说明

目录

一、 反射内存卡基本特征

型号：vmipci-5565-11000

1． 板载内存 128MB ，地址空间：0x0 ~ 0x7FFFFFF
2． 4k FIFOs
3． Transmission Mode=Multimode
4． No Conformal Coating 保形[角]涂料

二、 中断式通信流程

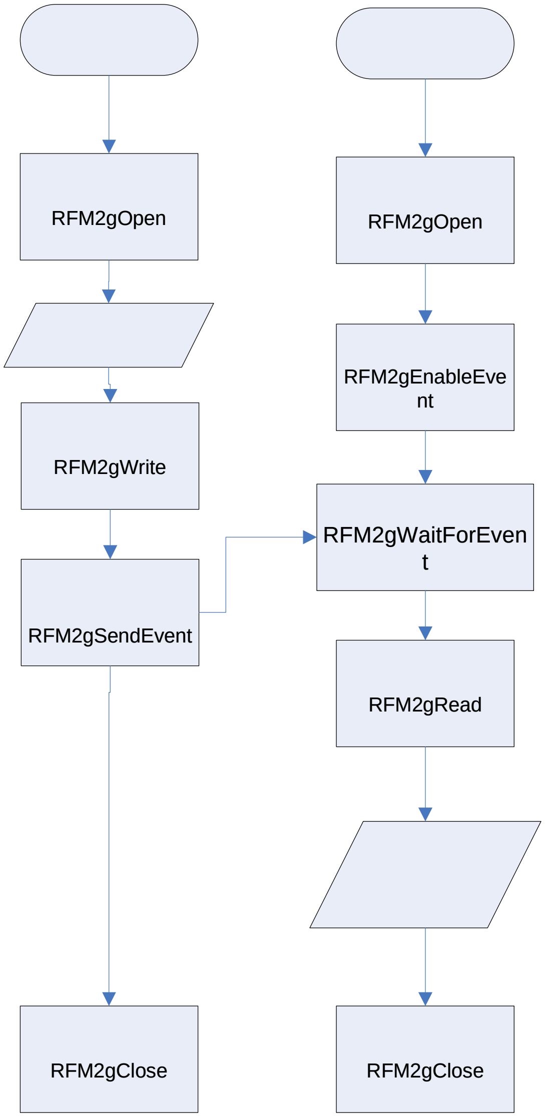

图 1 中断式通信流程图

2.1 特点：

一、发送方和接收方通过事件进行同步，CPU 占用少；
二、发送方可以向多个指定的接收方发送数据，即 1 对多方式；也可以实现广播方式。

2.2 注意事项：

1. 当接收方调用 RFM2gWaitForEvent 函数后，将挂起当前线程。直到有事件发生或等待超时才能恢复，因此接收部分的代码应采用多线程编程；

2. RFM2gSendEvent 需要指定接收设备的 NodeID，该参数由板卡上的跳线决定（Each RFM2g device on an RFM2g network is uniquely identified by its node ID, which is manually set by jumpers on the device when the RFM2g network is installed. The driver determines the node ID when the device is initialized）。本机的 NodeID 可以通过 API RFM2gNodeID 获取；如果采取广播方式，则参数 NodeID 应指定为宏定义 RFM2G_NODE_ALL；
3. 数据读写有两种方式：直接读写和内存映射。直接读写的相关函数有：RFM2gRead 和 RFM2gWrite 。内存映射的相关函数有：RFMUserMemory 和 RFMUnMapUserMemory。后者将板载内存按页(page)映射到程序的内存空间，对映射内存的操作将直接反应到板载内存上。按照手册的解释：使用内存映射后，数据的传输将使用 PIO 方式，不使用 DMA 方式。而直接读写函数的数据传输将尽可能采取 DMA 方式。

三、 代码

3.1 收发一体的通信代码

（摘自例程 rfm2g_send.c，为便于理解，去掉了其中的错误处理代码）：

```
#include "rfm2g_windows.h"  //屏蔽在 Vs2005 中编译时的警告
#include "rfm2g_api.h"     //rfm API
#define BUFFER_SIZE    256       //缓冲区大小 256byte
#define OFFSET_1      0x1000     //写数据起始位置 4k
#define OFFSET_2      0x2000     //读数据起始位置 8k
#define TIMEOUT       60000      //超时时间 60s
#define DEVICE_PREFIX   "\\\\.\\rfm2g"   //win 系统的 PCI 设备名前缀

RFM2G_STATUS   result;    // RFM2g API 调用的返回值,成功为 RFM2G_SUCCESS
RFM2G_CHAR     device[40]; // 完整的设备名由前缀和设备编号组成
RFM2GHANDLE    Handle = 0; //设备操作句柄，由 RFM2gOpen 返回

//构造设备名,如"\\\\.\\rfm2g1"
   sprintf(device, "%s%d", DEVICE_PREFIX, numDevice);

//打开设备
result = RFM2gOpen( device, &Handle );

//使网络中断可用。默认情况下，反射内存网的中断是不可用的，RFM2gEnableEvent
函数使得接收设备可以响应网络中断。如果发送方不需响应中断，则不必调用该函数
result = RFM2gEnableEvent( Handle, RFM2GEVENT_INTR2 );

//将数据写入反射内存卡的板载内存。
result  =  RFM2gWrite(  Handle,  OFFSET_1,  (void  *)outbuffer,
BUFFER_SIZE*4 );
```

```
/*在板载内存的有效范围之内，从第二个参数指定起始地址开始写入数据。写入的长度按字节计算。
字长换算法则：
1 byte= 1 RFM2G_UINT8
1 word= 1 RFM2G_UINT16= 2* RFM2G_UINT8
1 long word= 1  RFM2G_UINT32= 4* RFM2G_UINT8
*/

//发网络中断
result = RFM2gSendEvent( Handle, otherNodeId, RFM2GEVENT_INTR1, 0);

//等待中断
RFM2GEVENTINFO EventInfo;
EventInfo.Event = RFM2GEVENT_INTR2;  //等待的网络中断类型
EventInfo.Timeout = TIMEOUT;         //等待多久即超时
result = RFM2gWaitForEvent( Handle, &EventInfo );//调用后程序挂起

//读数据.与 RFM2gWrite 函数类似，需要事先分配读取缓冲和指定读取数据的长度
result = RFM2gRead( Handle, OFFSET_2, (void *)inbuffer,
BUFFER_SIZE*4);

//关闭设备
   RFM2gClose( &Handle );

//通用错误处理
if( result != RFM2G_SUCCESS )
{
   printf("Error: %s\n", RFM2gErrorMsg(result) );
   RFM2gClose( &Handle );
   return(result);
}
```

3.2 测试用代码段

3.2.1 获取板卡配置信息

```
   RFM2G_STATUS Status;
   RFM2GCONFIG Config;
   /* Get the board Configuration */
   Status = RFM2gGetConfig( Handle, &Config );
   if( Status != RFM2G_SUCCESS )
   {
      printf( "Could not get the board Configuration.\n" );
```

```
       printf( "Error: %s.\n\n",  RFM2gErrorMsg(Status));
       return(-1);
   }

   /* Print board configuration */
   printf ("   Driver Part Number    \"%s\"\n", Config.Name);
   printf  ("        Driver  Version                  \"%s\"\n",
Config.DriverVersion);
   printf ("   Device Name          \"%s\"\n", Config.Device);
   printf ("   Board Instance       %d\n",    Config.Unit);
   printf ("   Board ID            0x%02X\n", Config.BoardId);
   printf ("   Node ID             0x%02X\n", Config.NodeId);
   printf ("   Installed Memory     %ud (0x%08X)\n",
          Config.MemorySize, Config.MemorySize);
   printf ("   Memory Offset:       ");

   switch (Config.Lcsr1 & 0x0030000 )
   {
      case 0x00000000:
            printf ("0x00000000\n");
            break;

      case 0x00010000:
            printf ("0x04000000\n");
            break;

      case 0x00020000:
            printf ("0x08000000\n");
            break;

      case 0x00030000:
            printf ("0x0c000000\n");
            break;

      default: /* S/W Error */
            printf("\n");
   }

   printf  ("        Board  Revision                  0x%02X\n",
Config.BoardRevision);
   printf  ("       PLX  Revision                     0x%02X\n",
Config.PlxRevision);

   /* Display Board Configuration */
```

```
printf ("   Config.Lcsr1         0x%08x\n", Config.Lcsr1);

printf("RFM2g Configuration:\n");

if (Config.Lcsr1 & 0x01000000)
{
   printf ("       Rogue Master 0 Enabled\n");
}

if (Config.Lcsr1 & 0x02000000)
{
   printf ("       Rogue Master 1 Enabled\n");
}

if (Config.Lcsr1 & 0x04000000)
{
   printf ("       Redundant Mode\n");
}
else
{
   printf ("       Fast Mode\n");
}

if (Config.Lcsr1 & 0x08000000)
{
   printf ("       Local Bus Parity Enabled\n");
}

if (Config.Lcsr1 & 0x10000000)
{
   printf ("       Loopback Enabled\n");
}

if (Config.Lcsr1 & 0x20000000)
{
   printf ("       Dark-on-Dark Enabled\n");
}

if (Config.Lcsr1 & 0x40000000)
{
   printf ("       Transmitter Disabled\n");
}

/* PCI Configuration Info */
```

```
   printf("RFM2g PCI Configuration:\n");
   printf ("   bus              0x%02x\n", Config.PciConfig.bus);
   printf  ("       function                           0x%02x\n",
Config.PciConfig.function);
   printf ("   type             0x%04x\n", Config.PciConfig.type);
   printf  ("       devfn                              0x%08x\n",
Config.PciConfig.devfn);
   printf  ("       revision                           0x%02x\n",
Config.PciConfig.revision);
   printf   ("        rfm2gOrBase                      0x%08x\n",
Config.PciConfig.rfm2gOrBase);
   printf    ("           rfm2gOrWindowSize            0x%08x\n",
Config.PciConfig.rfm2gOrWindowSize);
   printf   ("          rfm2gOrRegSize                 0x%08x\n",
Config.PciConfig.rfm2gOrRegSize);
   printf   ("        rfm2gCsBase                      0x%08x\n",
Config.PciConfig.rfm2gCsBase);
   printf    ("           rfm2gCsWindowSize            0x%08x\n",
Config.PciConfig.rfm2gCsWindowSize);
   printf   ("          rfm2gCsRegSize                 0x%08x\n",
Config.PciConfig.rfm2gCsRegSize);
   printf  ("       rfm2gBase                          0x%08x\n",
Config.PciConfig.rfm2gBase);
   printf   ("         rfm2gWindowSize                 0x%08x\n",
Config.PciConfig.rfm2gWindowSize);
   printf   ("         interruptNumber                 0x%02x\n",
Config.PciConfig.interruptNumber);
```

3.2.2 测试反射内存网连接

```
   RFM2G_STATUS status;
   /* If the ring is intact, the "own-data" bit will be set */
   status = RFM2gCheckRingCont(cmd->Handle);
   if( status == RFM2G_SUCCESS )
   {
      printf( "The Reflective Memory link is intact.\n" );
   }
   else
   {
      printf( "Error: %s.\n\n",  RFM2gErrorMsg(status));
   }
```

3.2.3 DMA 性能测试

```
   RFM2G_INT32    index;  /* Selects appropriate nomenclature     */
   char *        name;   /* Local pointer to command name        */
   char *        desc;   /* Local pointer to command description  */
   RFM2G_STATUS   Status;
   time_t        time1;  /* ANSI C time */
   double        mBytePerSec   = 0;
   int          numBytes     = 0;
   int          numIterations = 10000;
   int          count;
   int          timeToRunTest = 2;  /* Time to run tests in seconds
*/
   RFM2G_UINT32   dmaThreshold;

   /* Get the DMA Threshold */
   RFM2gGetDMAThreshold(Handle, &dmaThreshold);

   printf("\nRFM2g  Performance  Test  (DMA  Threshold  is  %d)\n",
dmaThreshold);

printf("---------------------------------------------------\n");
   printf("   Bytes    Read IOps   Read MBps    Write IOps    Write
MBps\n");

   for (numBytes = 0; numBytes < MEMBUF_SIZE; )
   {
      if (numBytes < 32)
      {
         numBytes += 4;
      }
      else if (numBytes < 128)
      {
         numBytes += 32;
      }
      else if (numBytes < 2048)
      {
         numBytes += 128;
      }
      else if (numBytes < 4096)
      {
         numBytes += 512;
      }
```

```
       else if (numBytes < 16384)
       {
          numBytes += 4096;
       }
       else if (numBytes < 131072)
       {
          numBytes += 16384;
       }
       else if (numBytes < 262144)
       {
          numBytes += 65536;
       }
       else
       {
          numBytes += 262144;
       }

       numIterations = 0;

       if (time(&time1) == -1)
          {
          printf("ANSI C time function returned ERROR\n");
          return(-1);
          }

       /* Wait for the timer to get to the begining of a second */
       while (difftime(time(0), time1) == 0)
          {
          /* Let's wait */
          count++;
          }

       /* Get the start time */
       time(&time1);

       /* The accuracy of the results is dependent on the amount of
time
         the test runs and the Number of IO's per second.  This is not
a
         precise test, the priority of this task along with the limited
         resolution (1 Sec) of the ANSI C time function affects the
         precision.  */

       while (difftime(time(0), time1) < timeToRunTest)
       {
          Status = RFM2gRead(Handle,
                     0, /* rfmOffset */
                     (void*) MemBuf,
                     numBytes);

          if (Status != RFM2G_SUCCESS)
          {
             printf("RFM2gRead : Failure\n");
             printf( "Error: %s.\n\n",  RFM2gErrorMsg(Status));
             return(-1);
          }

          numIterations++;

       }

       /* Calculate MByte/Sec = Total number of bytes transferred /
(Time
         in seconds * 1024 * 1024) */
       mBytePerSec   =   (double)(numBytes   *   numIterations)   /
( timeToRunTest *
                   1024.0 * 1024.0);

       printf("%8d   %10d     %6.1f", numBytes,
            (numIterations / timeToRunTest), mBytePerSec);

       numIterations = 0;

       time(&time1);

       /* Wait for the timer to get to the begining of a second */
       while (difftime(time(0), time1) == 0)
          {
          /* Let's wait */
          count++;
          }

       /* Get the start time */
       time(&time1);

       /* Perform the IO until the elapsed time occurs */
       while (difftime(time(0), time1) < timeToRunTest)
       {
          Status = RFM2gWrite(Handle,
                      0, /* rfmOffset */
                      (void*) MemBuf,
                      numBytes);

          if (Status != RFM2G_SUCCESS)
          {
             printf( "RFM2gWrite : Failure\n");
             printf( "Error: %s.\n\n",  RFM2gErrorMsg(Status));
             return(-1);
          }

          numIterations++;

       }

       mBytePerSec   =   (double)   (numBytes   *   numIterations)   /
(timeToRunTest *
                   1024.0 * 1024.0);

       printf("    %10d       %6.1f\n",
             (numIterations / timeToRunTest), mBytePerSec);

   } /* for */
```

3.3 常用 API

3.3.1 RFM2gOpen()

Several programs and execution threads may have the same RFM2g interface open at any given time. The driver and the API library are thread-safe; （该函数支持多线程）

Syntax

```
STDRFM2GCALL RFM2gOpen( char *DevicePath, RFM2GHANDLE *rh );
```

Parameters

- *DevicePath* Path to special device file (I). Refer to your driver-specific manual for the format of DevicePath.
- *rh* Pointer to an RFM2GHANDLE structure (IO).

3.3.2 RFM2gClose()

Once the RFM2g handle is closed, all of the facilities using that handle are no longer accessible, including the local RFM2g memory, which may be mapped into the application program’s virtual memory space.

Syntax

```
STDRFM2GCALL RFM2gClose( RFM2GHANDLE *rh );
```

Parameters

rh Initialized previously with a call to **RFM2gOpen()** (I).

3.3.3 RFM2gRead()

The RFM2g driver attempts to fulfill the RFM2gRead() request using the bus master DMA feature available on the RFM2g device. The driver will move the data using the DMA feature if the length of the I/O request is at least as long as the minimum DMA threshold.

If the RFM2g device does not support the bus master DMA feature, or if the I/O request does not meet the constraints listed above, then the driver will move the data using PIO.

Syntax

```
STDRFM2GCALL RFM2gRead( RFM2GHANDLE rh,RFM2G_UINT32 Offset,void *Buffer,
RFM2G_UINT32 Length );
```

Parameters

- *rh* Handle to open RFM2g device (I).
- *Offset* Width-aligned offset to Reflective Memory at which to begin the read (I).Valid offset values are 0x0 to 0x3FFFFFF for 64MB cards, 0x0 to 0x7FFFFFF for 128MB cards and 0x0 to 0x0FFFFFF for 256MB cards.
- *Buffer* Pointer to where data is copied from Reflective Memory (O).
- *Length* Number of bytes to transfer (I). Valid values are 0 to ([*RFM Size*] *-rfmOffset*).

3.3.4 RFM2gWrite()

If the RFM2g interface supports the bus master DMA feature and the I/O request meets certain constraints, the RFM2g device driver will use DMA to perform the I/O transfer.

Syntax

```
STDRFM2GCALL RFM2gWrite( RFM2GHANDLE rh,RFM2G_UINT32 Offset,void *Buffer,
RFM2G_UINT32 Length );
```

Parameters

- *rh* Handle to open RFM2g device (I).
- *Offset* Width-aligned offset in Reflective Memory at which to begin the write (I).Valid offset values are 0x0 to 0x3FFFFFF for 64MB cards, 0x0 to 0x7FFFFFF for 128MB cards and 0x0 to 0x0FFFFFFF for 256MB cards.
- *Buffer* Pointer to where data is copied to Reflective Memory (I).
- *Length* Number of bytes units to write (I). Valid values are 0 to ([*RFM Size*] *-rfmOffset*).

3.3.5 RFM2gEnableEvent()

RFM2g network interrupts are not enabled by default. The RFM2gEnableEvent() function enables an event so an interrupt can be generated on the receiving node. User applications are notified of received events by using the RFM2gWaitForEvent() or RFM2gEnableEventCallback() function.

Syntax

```
STDRFM2GCALL RFM2gEnableEvent( RFM2GHANDLE rh, RFM2GEVENTTYPE EventType );
```

Parameters

- *rh* Handle to open RFM2g device (I).
- *EventType* Specifies which interrupt event to enable (I).Interrupts correlate to the following event IDs:

Event Interrupt	**Event ID**
Reset Interrupt	RFM2GEVENT_RESET
Network Interrupt 1	RFM2GEVENT_INTR1
Network Interrupt 2	RFM2GEVENT_INTR2
Network Interrupt 3	RFM2GEVENT_INTR3
Network Interrupt 4 (Init Interrupt)	RFM2GEVENT_INTR4
Bad Data Interrupt	RFM2GEVENT_BAD_DATA
RX FIFO Full Interrupt	RFM2GEVENT_RXFIFO_FULL
Rogue Packet Detected and Removed Interrupt	RFM2GEVENT_ROGUE_PKT
RX FIFO Almost Full Interrupt	RFM2GEVENT_RXFIFO_AFULL
Sync Loss Occurred Interrupt	RFM2GEVENT_SYNC_LOSS
Memory Write Inhibited	RFM2GEVENT_MEM_WRITE_INHIBITED
Memory Parity Error	RFM2GEVENT_LOCAL_MEM_PARITY_ERR

3.3.6 RFM2gClearEvent()

The RFM2gClearEvent() function clears any or all pending interrupt events for a specified event.

Syntax

```
STDRFM2GCALL RFM2gClearEvent( RFM2GHANDLE rh,RFM2GEVENTTYPE EventType );
```

Parameters

- *rh* Handle to open RFM2g device (I).
- *EventType* The event FIFO to clear (I).Interrupts correlate to the following event IDs:

Interrupt	**Event ID**
Reset Interrupt	RFM2GEVENT_RESET
Network Interrupt 1	RFM2GEVENT_INTR1
Network Interrupt 2	RFM2GEVENT_INTR2
Network Interrupt 3	RFM2GEVENT_INTR3

Network Interrupt 4 (Init Interrupt)	RFM2GEVENT_INTR4
Bad Data Interrupt	RFM2GEVENT_BAD_DATA
RX FIFO Full Interrupt	RFM2GEVENT_RXFIFO_FULL
Rogue Packet Detected and Removed Interrupt	RFM2GEVENT_ROGUE_PKT
RX FIFO Almost Full Interrupt	RFM2GEVENT_RXFIFO_AFULL
Sync Loss Occurred Interrupt	RFM2GEVENT_SYNC_LOSS
Memory Write Inhibited	RFM2GEVENT_MEM_WRITE_INHIBITED
Memory Parity Error	RFM2GEVENT_LOCAL_MEM_PARITY_ERR
All interrupts	RFM2GEVENT_LAST

3.3.7 RFM2gGetEventCount()

The RFM2gGetEventCount() function returns the event count for a specified event.

Syntax

```
STDRFM2GCALL  RFM2gGetEventCount(RFM2GHANDLE  rh,  RFM2GEVENTTYPE
EventType),  RFM2G_UINT32 *Count);
```

Parameters:

- *rh* Handle to open RFM2g device (I).
- *EventType* The event FIFO to clear (I).
- *Count* Pointer to where the event count of the specified event is written (O).

Interrupts correlate to the following event IDs:

Interrupt	**Event ID**
Reset Interrupt	RFM2GEVENT_RESET
Network Interrupt 1	RFM2GEVENT_INTR1
Network Interrupt 2	RFM2GEVENT_INTR2
Network Interrupt 3	RFM2GEVENT_INTR3
Network Interrupt 4 (Init Interrupt)	RFM2GEVENT_INTR4
Bad Data Interrupt	RFM2GEVENT_BAD_DATA
RX FIFO Full Interrupt	RFM2GEVENT_RXFIFO_FULL
Rogue Packet Detected and Removed Interrupt	RFM2GEVENT_ROGUE_PKT
RX FIFO Almost Full Interrupt	RFM2GEVENT_RXFIFO_AFULL
Sync Loss Occurred Interrupt	RFM2GEVENT_SYNC_LOSS
Memory Write Inhibited	RFM2GEVENT_MEM_WRITE_INHIBITED
Memory Parity Error	RFM2GEVENT_LOCAL_MEM_PARITY_ERR

3.3.8 RFM2gClearEventCount()

The RFM2gClearEventCount() function clears event counts for a specified event or all events.

Syntax

```
STDRFM2GCALL   RFM2gClearEvent(   RFM2GHANDLE   rh,RFM2GEVENTTYPE
EventType );
```

Parameters:

- *rh* Handle to open RFM2g device (I).
- *EventType* The event FIFO to clear (I).
- *Count* Pointer to where the event count of the specified event is written (O).

Interrupts correlate to the following event IDs:

Interrupt	**Event ID**
Reset Interrupt	RFM2GEVENT_RESET
Network Interrupt 1	RFM2GEVENT_INTR1
Network Interrupt 2	RFM2GEVENT_INTR2
Network Interrupt 3	RFM2GEVENT_INTR3
Network Interrupt 4 (Init Interrupt)	RFM2GEVENT_INTR4
Bad Data Interrupt	RFM2GEVENT_BAD_DATA
RX FIFO Full Interrupt	RFM2GEVENT_RXFIFO_FULL
Rogue Packet Detected and Removed Interrupt	RFM2GEVENT_ROGUE_PKT
RX FIFO Almost Full Interrupt	RFM2GEVENT_RXFIFO_AFULL
Sync Loss Occurred Interrupt	RFM2GEVENT_SYNC_LOSS
Memory Write Inhibited	RFM2GEVENT_MEM_WRITE_INHIBITED
Memory Parity Error	RFM2GEVENT_LOCAL_MEM_PARITY_ERR
All interrupts	RFM2GEVENT_LAST

3.3.9 RFM2gSendEvent()

The RFM2gSendEvent() function sends an interrupt event and a 32-bit longword value to another node.

Syntax

```
RFM2G_STATUS RFM2gSendEvent( RFM2GHANDLE rh,RFM2G_NODE ToNode,RFM2GEVENTTYPE EventType,RFM2G_UINT32 ExtendedData );
```

Parameters

- *rh* Handle to open RFM2g device (I).
- *ToNode* Who will receive the interrupt event (I) (RFM2G_NODE_ALL sends the event to all nodes).**NOTE:** A node cannot send an event to itself.
- *EventType* The type of interrupt event to send (I).Interrupts correlate to the following event IDs:

Interrupt	**Event ID**
Reset Interrupt	RFM2GEVENT_RESET
Network Interrupt 1	RFM2GEVENT_INTR1

Network Interrupt 2	RFM2GEVENT_INTR2
Network Interrupt 3	RFM2GEVENT_INTR3
Network Interrupt 4	RFM2GEVENT_INTR4

I *ExtendedData* `User-defined data (I).`