APPLICATION NOTE

HANDLING SUSPEND MODE ON A USB MOUSE

by Microcontroller Division Application Team

INTRODUCTION

All USB devices must support Suspend mode. Suspend mode enables the devices to enter low-power mode if no bus activity is detected for more than 3.0 ms.

Like USB keyboards and pointing devices, USB mice must be able to exit Suspend mode if a button has been pressed or if a movement has been detected. This feature is called Remote wake-up mode. A Remote wake-up involves a Resume sequence on the USB lines and recovery of communication between the mouse and the host.

The following application note describes the implementation of Suspend and Remote wake-up modes on a USB mouse using the ST7263 microcontroller. The first chapter focuses on the recommendations before entering Suspend mode. Then a description of the RC external circuit for handling Remote wake-up mode is detailed. It contains power management recommendations and RC value proposals. The third chapter describes Resume mode. Then chapter 4 and chapter 5 describe software implementation and program flow.

It is assumed that the reader is familiar with the ST7263 microcontroller and USB.

Table of Contents

1 ENTERING SUSPEND MODE

1.1 SPECIFICATION

The USB mouse must be able to enter Suspend mode from any powered state. It goes into suspend mode when it detects a constant idle state on the USB data lines for more than 3.0 ms. (No Start Of Frames are issued by the Host for more than 3.0 ms). Any bus activity keeps the mouse out of the suspend state.

When the mouse is in the suspend state, it must draw less than 500 µA from the bus.

1.2 ST7263 IMPLEMENTATION

All the ST7263 microcontroller USB events are managed by interrupts. As soon as a constant idle state occurs for more than 3.0 ms, the SUSPEND bit of the **interrupt status register (ISTR)** is set by hardware. Then, the firmware determines the interrupt origin by reading the ISTR register, sets a bit in a software register (bmUsbIntFlag), and clears the interrupt flag. The USB polling routine reads the software register (bmUsbIntFlag) to determine the USB interrupt source and jumps to the corresponding interrupt routine.

To get the 500 µA specification, the ST7263 microcontroller must be placed in **HALT low-power mode**. HALT mode is entered by executing the HALT instruction in the SUSPEND interrupt routine. The internal oscillator is then turned off, causing all internal processing to be stopped, including the operation of the on-chip peripherals.

1.3 RECOMMENDATIONS BEFORE ENTERING SUSPEND MODE

Special care must be taken of the ST7263 port configuration before entering Suspend mode.

The Ports A and C contain integrated pull-up resistors. If these Ports are configured as inputs, they draw about 50 µA on each pin. (See ST7263 Data Sheet electrical characteristics). This means that the 8 Port A pins and the 3 port C pins could draw 550 µA together. Consequently it would be impossible to get the 500 µA current consumption specification.

Then Port A and Port C must then be therefor configured as outputs before entering Suspend mode. Correct voltage values must be applied to the output pins to make the current consumption as low as possible.

Port B does not contain any integrated pull-up resistor, but external pull-up resistors are implemented in hardware at the photo transistor outputs. Thus it is strongly recommended to configure these as outputs and to apply a 5V value on each pin before entering Suspend mode.

2 EXITING SUSPEND MODE

2.1 SPECIFICATION

A USB mouse can be woken-up from Suspend state by switching the bus state to the resume state, by normal bus activity, or by signalling a reset. A Mouse device can be woken-up by actions associated with internal functions and then cause signalling on its upstream connections to wake or alert the rest of the system. This feature is called remote wake-up.

2.2 ST7263 IMPLEMENTATION

All the ST7263 microcontroller USB events are managed by interrupts.

The ST7263 microcontroller exits Suspend mode when one of the following events occurs:

a) USB reset on the USB data lines for more than 10 ms min and 20 ms max.

The microcontroller exits Halt mode and jumps to the USB reset interrupt routine.

b) Resume signal on the USB data lines from upstream devices for at least 20 ms.

The microcontroller exits Halt mode and jumps to the USB_end_suspend interrupt routine.

c) External interrupt on a pin of the microcontroller (It can either be a rising or falling edge or a level change). The oscillator is then turned on, the microcontroller exits Halt mode and jumps to the corresponding interrupt routine.

This external interrupt is very important for managing remote wake-up mode. As described in the previous section, when the microcontroller enters HALT mode, it must draw less than 500 µA average and it must be able to check if a mouse button has been pressed or if the mouse as been moved. This implies that external interrupt must periodically wake-up the microcontroller to make it check its port states.

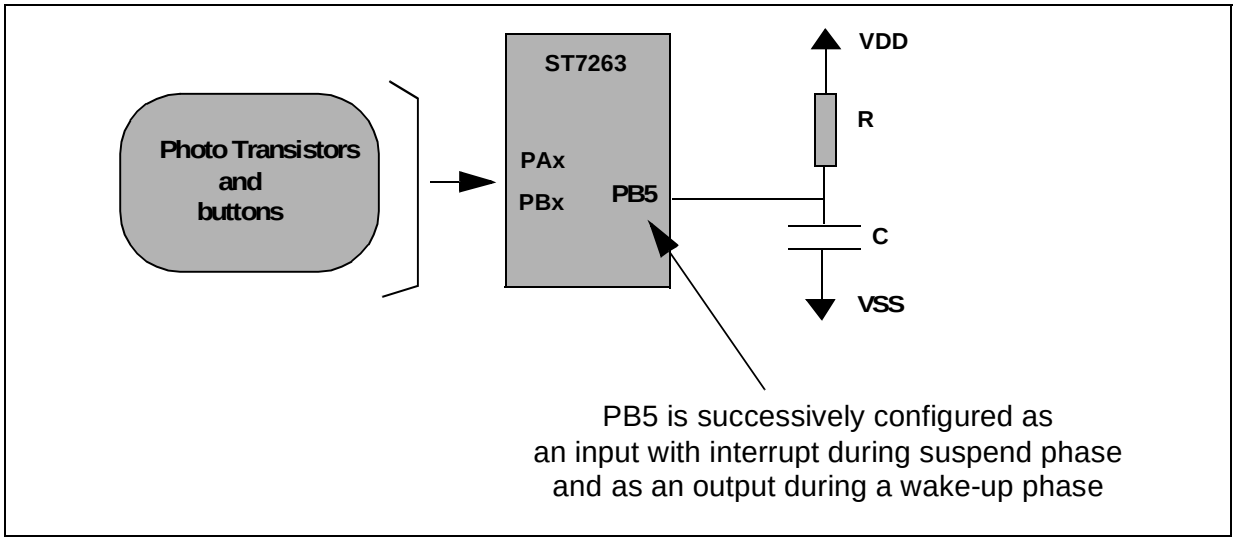
The following section proposes a cost effective solution for generating this periodic external interrupt.

2.3 EXTERNAL RC CIRCUIT IMPLEMENTATION

During Suspend mode, as the internal oscillator has been turned off, we must use an external event to generate an interrupt on a dedicated pin of the microcontroller. Moreover this interrupt signal must occur periodically.

The most common and cost-effective solution consists of using an external RC circuit.

Figure 1. RC external circuitry

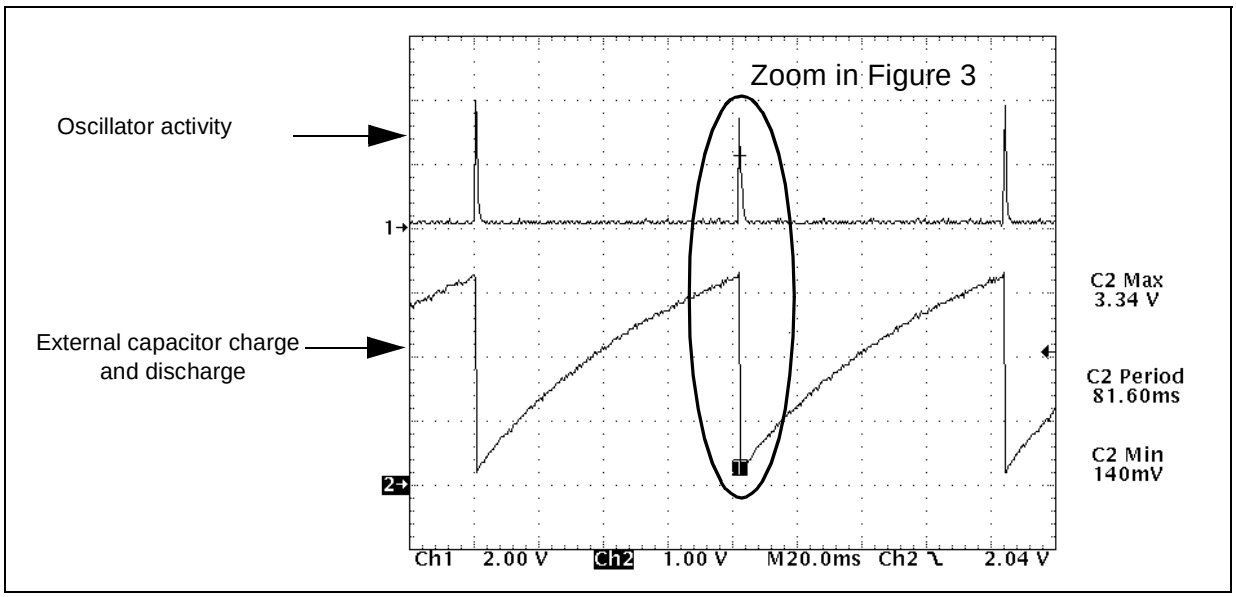


An external capacitor loads through an external resistor. This capacitor is connected to the PB5 external interrupt pin of the microcontroller. As soon as the capacitor load has reached the Low-to-High trigger level, the external interrupt vector is set. Then, the entire microcontroller is woken-up: The internal oscillator is turned on, causing all internal processing to resume, including the operations of the on-chip peripherals.

If the mouse has moved or if the buttons have been pressed, the application performs a Remote Wake-up sequence, otherwise the application discharges the capacitor and re-enters suspend mode.

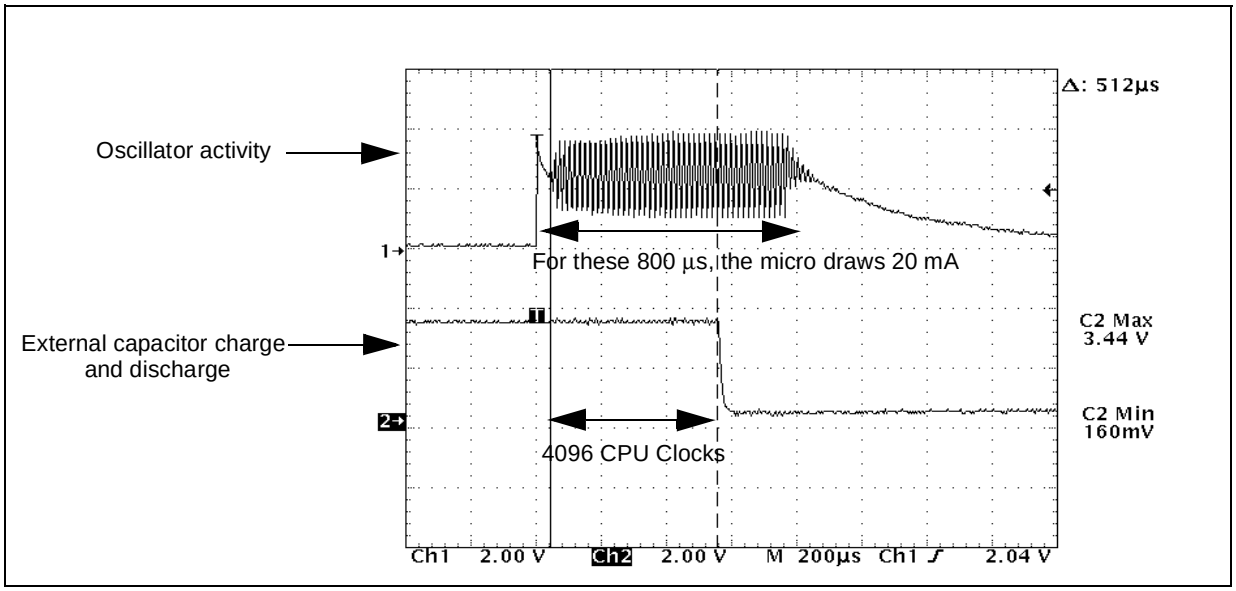
Figure 2. shows the RC circuit behaviour when the mouse is not moved.

Figure 2. RC circuit behaviour in suspend mode



During the charge of the capacitor, the microcontroller is in low-power mode, it draws only about 250 µA. As soon as the charge of the capacitor reaches the low-to-high interrupt trigger value (about 3.35 V), the microcontroller wakes-up. The oscillator is then turned on and a stabilization time is required before releasing CPU clock cycles. This stabilization time is 4096 CPU clock cycles. During this activity phase, the microcontroller draws about 20 mA. This oscillator wake-up phase is shown in Figure 3.

Figure 3. Oscillator Wake-up sequence

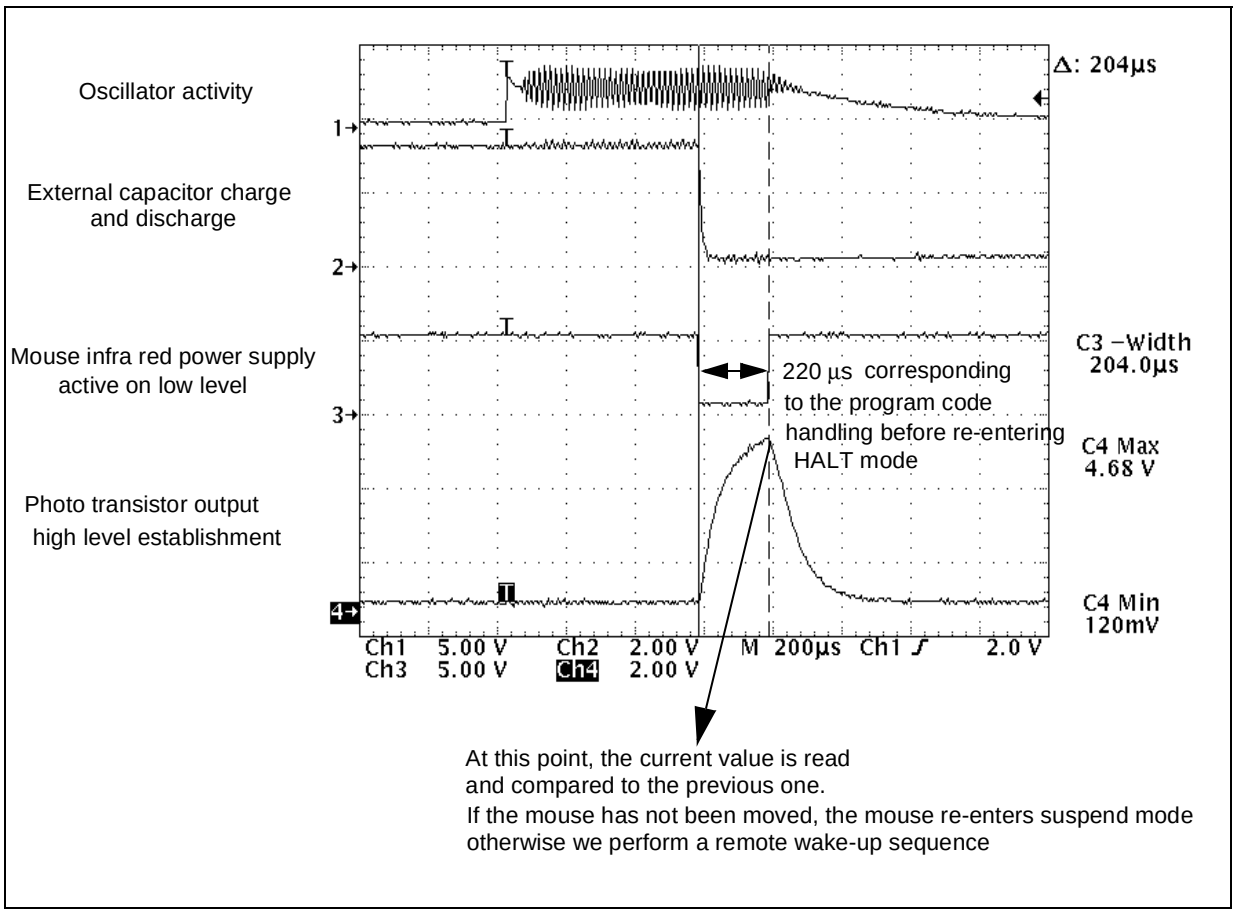


2.4 PHOTO TRANSISTORS TIMING REQUIREMENTS

To check if the mouse has moved, it is necessary to compare the current state of the Photo transistor outputs with the previous ones (recorded before entering suspend mode).

As soon as the microcontroller executes its program code, the infra red LEDs of the mouse are turned on. However, to establish the previous Photo transistor values (especially in case of high level values) a certain amount of time is required. This period is less than 200 µs with the components used . This is due to current establishment in the infra-red diode.

Figure 4. Photo transistor behaviour



2.5 RC TIME CONSTANT CALCULATION

As shown on the previous oscillogram, the microcontroller is active for 800 µs and it draws 20 mA. It is in low-power mode during the rest of the period and draws about 250 µA.

The following equation enables you to calculate the correct value of the RC time period to be within the Suspend mode 500 µA average max current specification:

$$Specvalue = \frac{(Imax \times WakeupTime) + (Imin \times (Period - WakeupTime))}{Period}$$

Example:

If you choose 450 µA to stay well within the specification and with the following parameters:

Imax = 20 mA - WakeupTime = 800 µs - Imin = 250 µA

You obtain the following equation:

$$450\mu s = \frac{(20mA \times 800\mu s) + (250\,\mu A \times (Period - 800\mu s))}{Period}$$

A Period value of 80 ms is obtained.

The RC value can then be calculated to get this period.

You need to pay attention to 2 parameters:

– The high level voltage imposed by the high trigger level (about 3.4V with a +/- 10% variation).

– The input impedance of the pin which modifies the equivalent resistance of the RC circuit.

The following table presents some RC values and the corresponding time periods and consumption measured on a ST7263 microcontroller.

Depending on the application supporting Suspend mode, you will have to find a compromise between the suspend interval time and the suspend average current.

Table 1. RC Value proposal

RC Values	Suspend interval time	Suspend average current
C= 330 nF - R= 1 MΩ	t= 306 ms	I= 400 µA
C= 330 nF - R= 670 KΩ	t= 200 ms	I= 405 µA
C= 330 nF - R= 330 KΩ	t= 97 ms	I= 440 µA
C= 100 nF - R= 670 KΩ	t= 80 ms	I= 450 µA
C= 100 nF - R= 330 KΩ	t= 37 ms	I= 540 µA **(out of spec.)**

Note 1: Suspend mode consumption has been measured with Ports A and C configured as outputs (This has been done to avoid additional current consumption in their pull-up resistors)

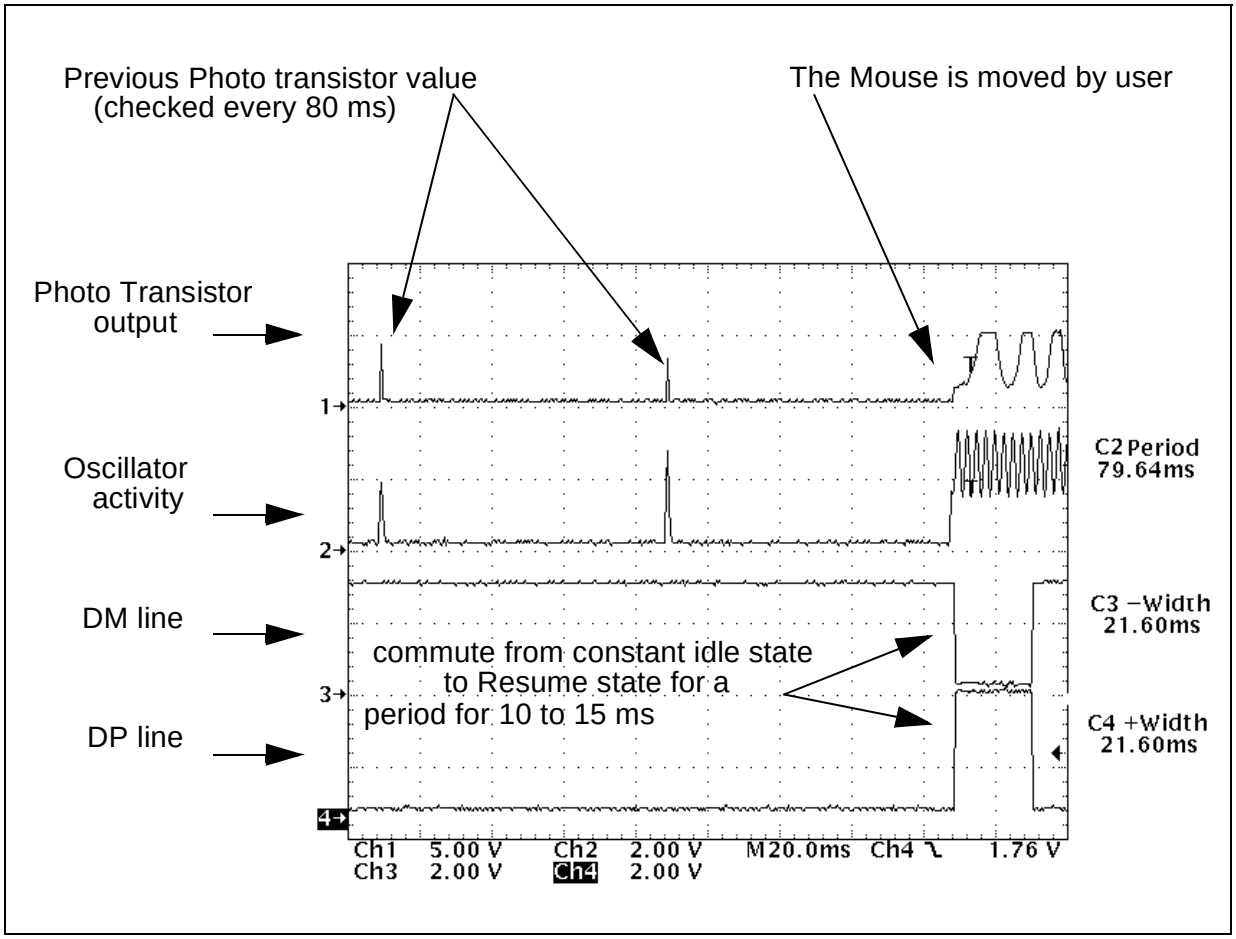
Note 2: A USB Mouse is a Human Interface Device. This means that it is manipulated by an operator at quite a low frequency. For mouse pointing devices, the best compromise is obtained by chosing an RC value which allows you to enter remote wake-up mode and frequently check the device sensor states (10 times per second is enough).

3 RESUME MODE

As soon as a different value has been read from the buttons or photo transistor outputs, the microcontroller performs a Remote Wake-up sequence to resume the communication flow between the Device and the Host.

An example of such a sequence is shown in Figure 5.

Figure 5. Resume state behaviour



Note that the DM and DP lines are maintained in Resume state for 21.6 ms while the USB specification release 1.1 requires the device to assert Resume signalling for a period between 10 ms and 15 ms. In fact the device asserts Suspend state for only 12 ms and then the host maintain this state for 21.6 ms. This 20 ms minimum Resume signalling time insures that all devices in the network that are enabled to see the resume are woken-up.

4 SOFTWARE IMPLEMENTATION

In the following section, all the programs have been developed using the Hiware C compiler Toolchain. ST provides a complete architecture as well as firmware drivers to help you develop your application quickly and easily. A list of reference documents is provided at the end of the application note.

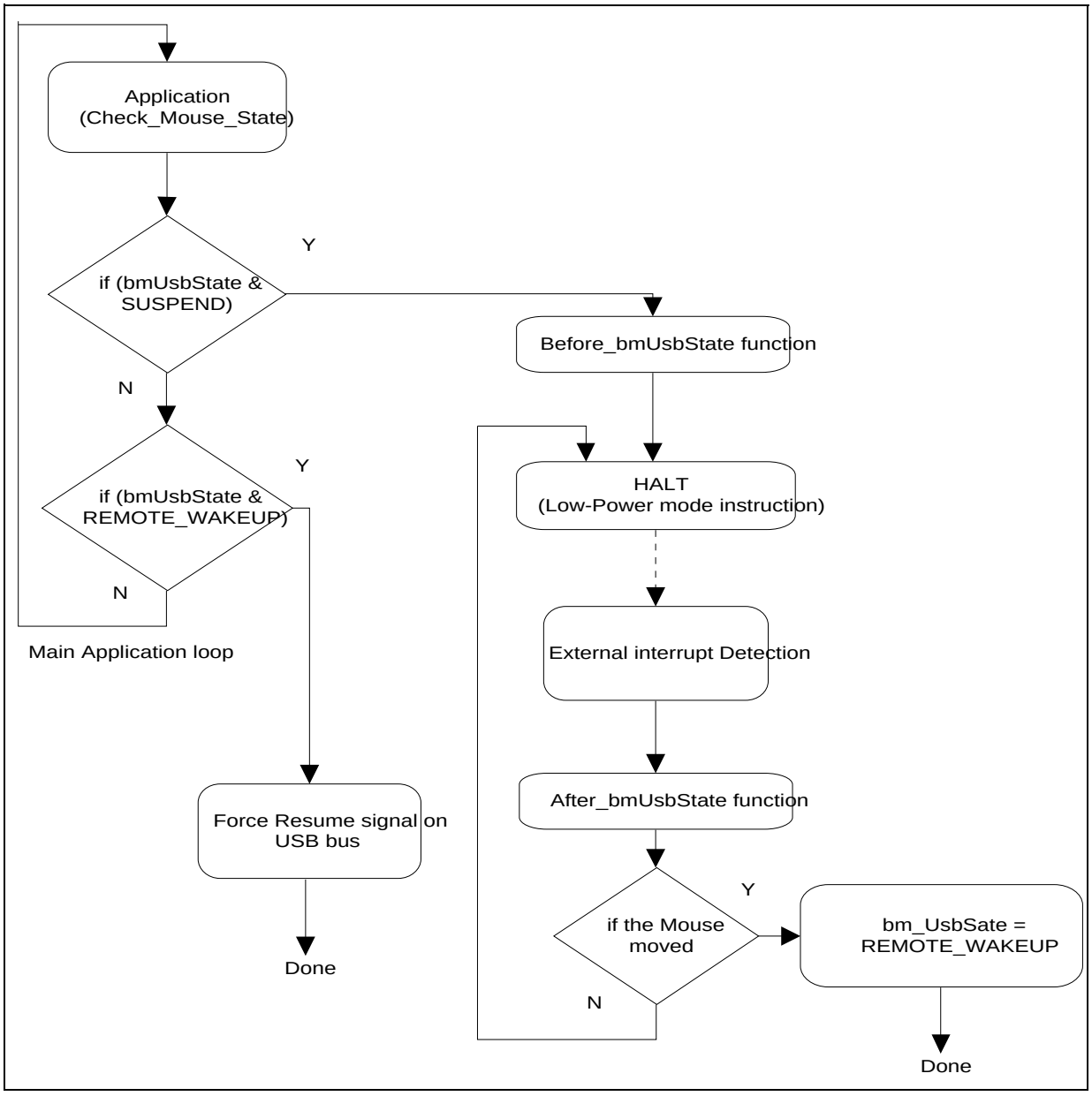
Depending on the ST7263 microcontroller, all USB events are managed by interrupt. When an USB event occurs, a flag of the Interrupt State Register (ISTR) is set by hardware. Then, the firmware determines the interrupt origin by reading the ISTR register, sets the corresponding bit in a software register (bmUsbIntFlag), and clears the interrupt flag. The USB polling routine reads the software register (bmUsbIntFlag) to determine the USB interrupt source and jumps to the corresponding interrupt routine.

Each interrupt routine sets the global variable “bmUsbState” to the corresponding USB state: SOF, Ennumerated, Suspended or Remote wake-up.

In the *Check_UsbState.c* file you only have to adapt your code to the example given below to handle resume and remote wake-up modes. Don’t forget to put all the I/Os in low-power mode in order to draw less than 500µA.

Figure 6. shows the flow-chart of the main application loop.

Figure 6. USB Suspend / Remote wake-up handling



The remote wake-up command is sent from device to Host. This happens when the microcontroller has been suspended and an external interrupt (PB5) restarts the oscillator. To send a remote wake up signal, the bmUsbState variable is set in the *Check_UsbState.c* file if the mouse has been moved. Otherwise, we re-enter Suspend mode.

5 PROGRAM FLOW

The following code gives an example of a USB Mouse application for handling suspend and remote wake-up modes.

```
/*-------------------------------------------------------------------------
ROUTINE NAME : Before_Enter_Suspend
INPUT/OUTPUT :
DESCRIPTION : This function details what we have to do before entering suspend
mode
/*-------------------------------------------------------------------------
void Before_Enter_Suspend(void)
   {
   unsigned int k;            // variable used in the capacitor discharge loop
   PADDR = 0xFF;              // The 2 Ports are configured in output for minimum
                              current consumption
   PCDDR = 0xFF;
   SetBit(PBDDR,5);           // PB5 in output mode (external RC circuit)
   ClrBit(PBDR,5);            // Clr PB5 : Discharge the external capacitor
   for (k=17; k>0;k --)       // 50us loop: This time interval is mandatory to
      asm nop;                // completly discharge the external capacitor
   Previous_Suspend_State = (PBDR & 0xD8); // Read the Phototransistor value be-
                              fore entering SUSPEND
   Current_Suspend_State = Previous_Suspend_State;
   SetBit(PCDR,0);            // Switch OFF the Mouse LED
   ClrBit(PBDDR,5);           // PB5 configured in input mode (We let the capacitor
                              charge)
   Wake_Up_Flag = 0;
   ITRFRE |= 0x20;            // Enable IT6 (PB5)
   }
/*-------------------------------------------------------------------------
ROUTINE NAME : After_End_Suspend
INPUT/OUTPUT :
DESCRIPTION : This function details what we have to do after exiting from suspend
mode
/*-------------------------------------------------------------------------
void After_End_Suspend(void)
   {
   unsigned int j;            // variable used in the 200 us loop
   if (Wake_Up_Flag == 1)
      {
      ClrBit(PCDR,0);         // switch on the Mouse LED
      SetBit(PBDDR,5);        // PB5 in output mode (external RC circuit)
      ClrBit(PBDR,5);         // Clr PB5 : Discharge the external capacitor
      ITRFRE |= 0x20;         // Re enable IT5
      for (j=68; j>0; j--)    // 200us loop: This time interval is mandatory to let
```

```
      asm nop; }              // each Phototransistor reach his previous value
   }

/*---------------------------------------------------------------------------
ROUTINE NAME : Check_BmUsbState
INPUT/OUTPUT :
DESCRIPTION : This subroutine polls the USB global variable bmUsbState
/*---------------------------------------------------------------------------
void Check_BmUsbState(void)
   {
   if(bmUsbState & SUSPEND)
      {
      Before_Enter_Suspend();
      asm
      {
      Loop: halt;             // Enters suspend mode
      }

      /***** return from interrupt in SUSPEND Mode *******/

      After_End_Suspend();
      Current_Suspend_State = (PBDR & 0xD8); // Filter on the 4 X-Y axis Optocou-
                              pler sensors
      if (Current_Suspend_State == Previous_Suspend_State)
                              {
         Current_Suspend_State = Previous_Suspend_State;
         SetBit(PCDR,0);      // switch OFF the Mouse Led
         ClrBit(PBDDR,5);     // PB5 configured in input (We let the capacitor
                              charge)
         ITRFRE |= 0x20;      // Re enable IT6
         Wake_Up_Flag = 0;
         asm jp Loop;
         }
      else
         {
         ITRFRE = 0x00;       // Disable IT6 (PB5)
         bmUsbState &= ~SUSPEND; // Reset "Go Suspend"
         bmUsbState |= REMOTE_WAKEUP; // Set "Remote Wake-up"
         ClrBit(PBDDR,5);     // PB5 configured in input
         }
      }

   if(bmUsbState & REMOTE_WAKEUP)
      RemoteWakeup();
```

```
   if(bmUsbState & SOF)
      {
      bmUsbState &= ~SOF; // Reset the SOF bit
      Mouse_Counter++; // This variable is used to refresh the Cursor Position
      }

   if(bmUsbState & SOF)
      nop;
   }


/*-------------------------------------------------------------------------
ROUTINE NAME : INT_IT1IT8
INPUT/OUTPUT : None
DESCRIPTION  : Rising edge interrupt
COMMENTS   :
/*-------------------------------------------------------------------------
#pragma TRAP_PROC SAVE_REGS
void INT_IT1IT8(void)
   {
    if(bmUsbState & SUSPEND) // We are in suspend mode
      {
      if((PBDR & 0x20) == 0x20)
         Wake_Up_Flag =1;
      }
   }
```

REFERENCE DOCUMENTS

– ST7263 Data Sheet

– AN 1017 Using the ST7 Universal Serial Bus Microcontroller

– AN 1069 Developing an ST7 USB Apllicaton

– AN 989 Starting with ST7 Hiware C.

– AN 1064 Writing Optimized Hiware C Language for ST7

A general training for ST7 (hardware and development tools) is available on the ST7 CD-ROM.

The USB Descriptor tool DT2.4 is available on the USB website at URL http//www.usb.org