To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

H8S/2218

Introduction to the H8S/2218 USB Peripheral

Introduction

This application note introduces the on chip USB peripheral of the H8S/2218 microcontroller. The H8S/2218 USB peripheral is a Full speed (12 Mbps) implementation and supports three USB transfer modes: Control, Bulk, and Interrupt.

Standard commands are processed automatically by hardware. Only the Set_Descriptor, Get_Descriptor, Class/Vendor Command, and SynchFrame commands should be processed by software

This application note will introduce the USB peripheral via three example applications. The first two use the Com Class driver to transfer data to and from the PC. The data are transferred using bulk transfers. The third uses the HID (Human Interface Device) Class to transfer data to and from the host PC. This data conforms to the HID report format and are transferred using control or interrupt transfers.

Contents

Overview of System

All of the introductory application notes were developed using a 3DK2218 board. This board is fitted with a Mini-B USB connector. For code development and debugging purposes, an E10A-USB emulator was used. The E10A-USB emulator enables software to be developed using the H8S/2218's on chip debug port. Figure 1.0 shows a graphical representation of the development environment.

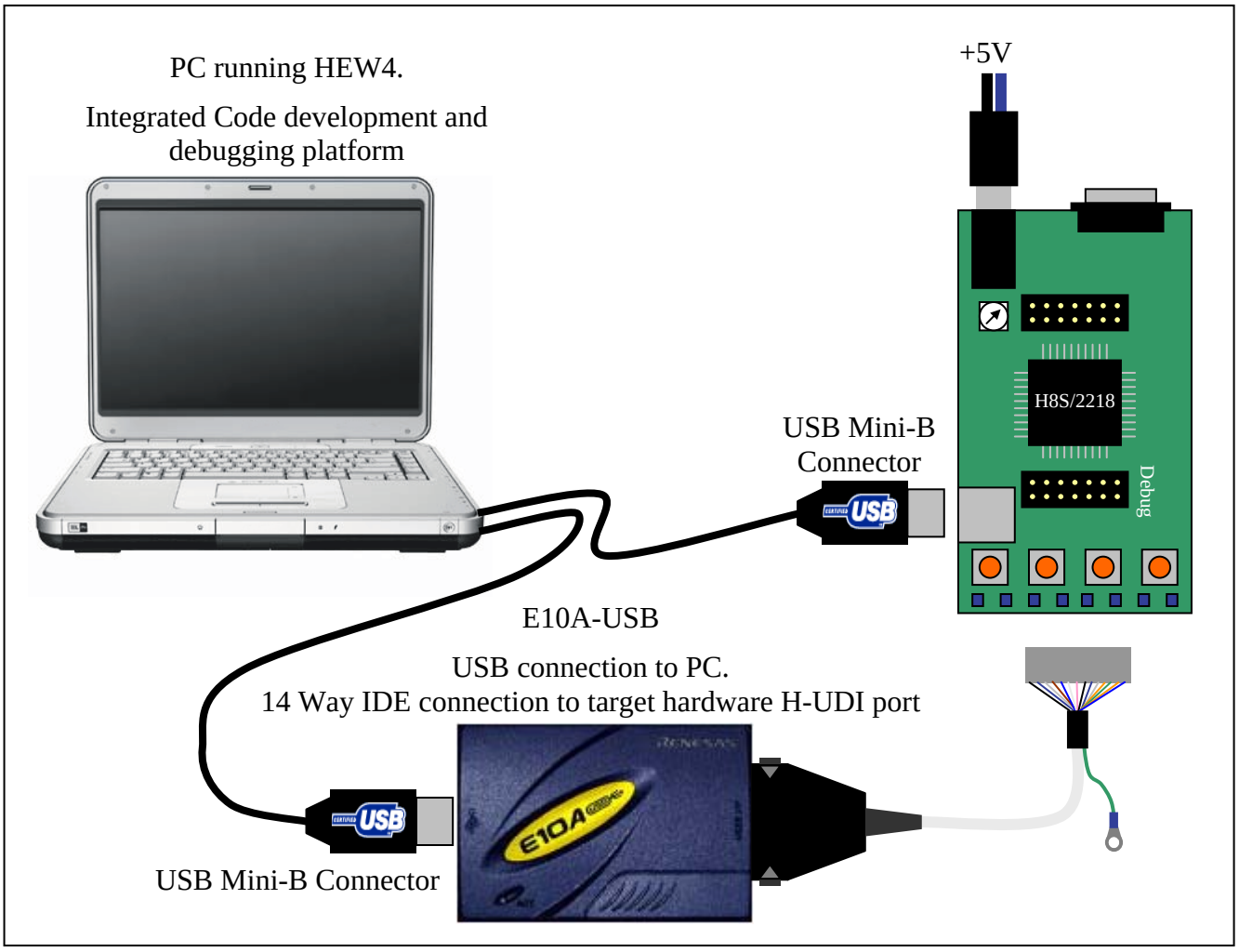


Figure 1.

The H8S/2218 & USB Peripheral

The H8S/2218 is a high performance 16-bit embedded microcontroller built around the high speed, 32-bit H8S/2000 CPU core.

The H8S incorporates 128 kBytes of FLASH memory and 12 kBytes of RAM. The on-chip peripherals include:

- DMA controller (DMAC)
- 16-bit timer-pulse unit (TPU)
- Watchdog timer (WDT)
- Real-time clock (RTC)
- Serial communication interface (SCI)
- Boundary scan
- Universal serial bus (USB)
- 10-bit A/D converter
- High-performance user debugging interface (H-UDI)
- Clock pulse generator

The main features of the USB peripheral are:

- On-chip UDC (USB Device Controller) conforming to USB 1.1
- Automatic processing of USB protocol
- Automatic processing of USB standard commands for endpoint 0
  - Note: Some commands need to be processed through the firmware
- Full-speed (12 Mbps) transfer supported
- Three transfer modes supported (Control, Bulk, Interrupt)
- 16 interrupt signals
- On-chip bus transceiver
- 4 Endpoints

Endpoint Name	Name	Transfer Type	Size	Max. Packet Capacity	FIFO Buffer	DMA Transfer
Endpoint 0	EP0s	Setup		8 bytes	8 bytes .	
	EP0I	Control-in		64 bytes	64 bytes .	
	EP0o	Control-out		64 bytes	64 bytes .	
Endpoint 1	EP1	Bulk-in		64 bytes	64 x 2 (128 bytes)	Possible
Endpoint 2	EP2	Bulk-out		64 bytes	64 x 2 (128 bytes)	Possible
Endpoint 3	EP3	Interrupt (in)		64 bytes	64 bytes (variable) .	

H8S/2218 USB Software

All of the applications detailed within this application note use many common USB functions and software in their operation. These common functions are detailed over the next pages.

Peripheral Initialisation

As with all H8S microcontrollers, the majority of peripherals are in Module Stop Mode when the H8S comes out of reset. To use the peripherals they have to be taken out of Module Stop Mode. This is no different for the USB peripheral. However, some pre-initialisation has to be performed prior to this.

The USB peripheral is mapped from address H'C00000 to H'C000FF, which is part of the external addressable area. Therefore, the Bus State Controller (BSC) registers have to be correctly configured prior to the USB peripheral being enabled, otherwise it will not be possible to communicate to the USB peripheral. It is also necessary to configure the Interrupt Controller prior to enabling the USB peripheral as the USB peripheral is interrupt driven.

The function `HardwareSetup()`[1] which is called as part of the Power-on Reset exception handler assigns interrupt priority levels to the peripherals and configures the Interrupt Controller for Mode 2 operation. Please refer to section 5 of the H8S/2218 Hardware User Manual for a detailed description of the interrupt controller.

The function `USBPreInitSetup()`[1] configures the BSC and takes the USB peripheral out of Module Stop Mode.

The BSC must be configured for 8-Bit, 3-State Access with 0 wait states.

Once the USB peripheral has been taken out of Module Stop Mode the USB clock[2] will start. When this clock is stable the CK48READY flag is set in the UIFR3 (USB Interrupt Flag Register 3).

When an Interrupt Flag is set an interrupt will be generated by the USB peripheral. The USB peripheral can 'direct' this to 1 of 2 interrupt vectors, depending on the settings of the UISRs (USB Interrupt Select Registers). If the corresponding bit is cleared to 0, the interrupt request will be handled by interrupt vector 104, EXIRQ0. If the corresponding bit is set to 1, the interrupt request will be handled by interrupt vector 105, EXIRQ1. By default, the UISRs have a value of 0. Therefore, the interrupt generated in response to the CK48READY flag will be handled by EXIRQ0. As many Interrupt Flags can generate the same interrupt, the Interrupt Service Routine (ISR) has to determine which interrupt has occurred. Interrogating the UIFRs (USB Interrupt Flag Registers) does this.

The function `HandleClockOK()`[1] is called by the EXIRQ0 ISR in response to the USB clock stabilisation and performs further USB peripheral initialisation, such as enabling and disabling the other USB interrupts and directing them to EXIRQ0 or EXIRQ1.

At this point the application could place the USB peripheral back into Module Stop Mode by setting Bit-0 of MSTPCRB, hence saving power. However, this application does not do this and now waits for a USB cable to be connected, which will generate a VBUS interrupt.

The VBUS interrupt is handled by EXIRQ0. The interrupt interrogates the USB Interrupt Flag Registers and in response to the VBUS interrupt, calls the `HandleVBus()`[1] function.

If a USB cable has been connected, the VBUS interrupt clears all of the USB FIFOs and outputs a logical '1' via Bit-6 of Port 3. This should be the enable trigger for some external logic, such as a SN74LVC126. The output of this logic pulls the D+ line high via a 1.5kΩ resistor to indicate that

[1] PLEASE REFER TO APPENDEX A FOR CODE LISTINGS

[2] THE USB PERIPHERAL HAS TO RUN AT 48 MHZ. THIS CLOCK RATE IS GENERATED FROM AN ON CHIP PLL WHICH HAS MULIPLICATION VALUES OF 2 OR 3. THEREFORE, THE H8S/2218 HAS TO OPERATE WITH AN EXTERNAL CRYSTAL OF 24 MHZ OR 16 MHZ.

the USB interface is Full Speed. The application will now wait for the next interrupt, which should be the Bus Reset Interrupt from the host PC.

If the USB cable has been disconnected, the VBUS interrupt clears Bit-6 of Port 3 to '0' and issues a software reset to the UDC. The application will now wait for the USB cable to be connected.

The function `HandleBusReset()`[1] simply clears all of the FIFOs and ensures that Stall conditions for all Endpoints are cleared.

The application now waits for a Setup Command from the host and which will generate the EXIRQ0 interrupt. In response to the SetupTS flag being set the function `HandleSetupCmd()`[1] is called.

This function calls `ReadSetupPacket()`[1] which reads the data from the UEDR0s (USB Endpoint Data Register 0s) register. UEDR0s stores the 8-Byte command sent to the host during set up. The function `ReadSetupPacket()`[1] assigns the 8-Byte command to the union `SetupData`.

With the data assigned to `SetupData` the function `DecodeSetupPacket()`[1] determines what has been requested. The functions `DecodeStandardSetupPacket()`[1], `GetDescriptorString()`[1], `DecodeClassSetupPacket()`[1] and `WriteControlInPacket()`[1] are subsequently called enabling the USB peripheral to successfully enumerate with the host PC.

As part of the enumeration process, the H8S provides information to the host PC. This information is held in the file `usbdescriptors.c & usbdescriptors.h`.

[1] PLEASE REFER TO APPENDIX A FOR CODE LISTINGS

Application One & Two: Com Class Demo

Overview

With the application software programmed into the H8S/2218, connect the 3DK to the host PC via a USB cable. If this is the first time the 3DK has been connected, the PC will require the '.inf' file. The .inf file is available for download as an accompaniment to this application note. When the host PC has successfully installed the driver and enumerated the H8S/2218 USB will appear as a COM port. This can be viewed via Device Manager. In this example the host PC has assigned COM7 to the H8S.

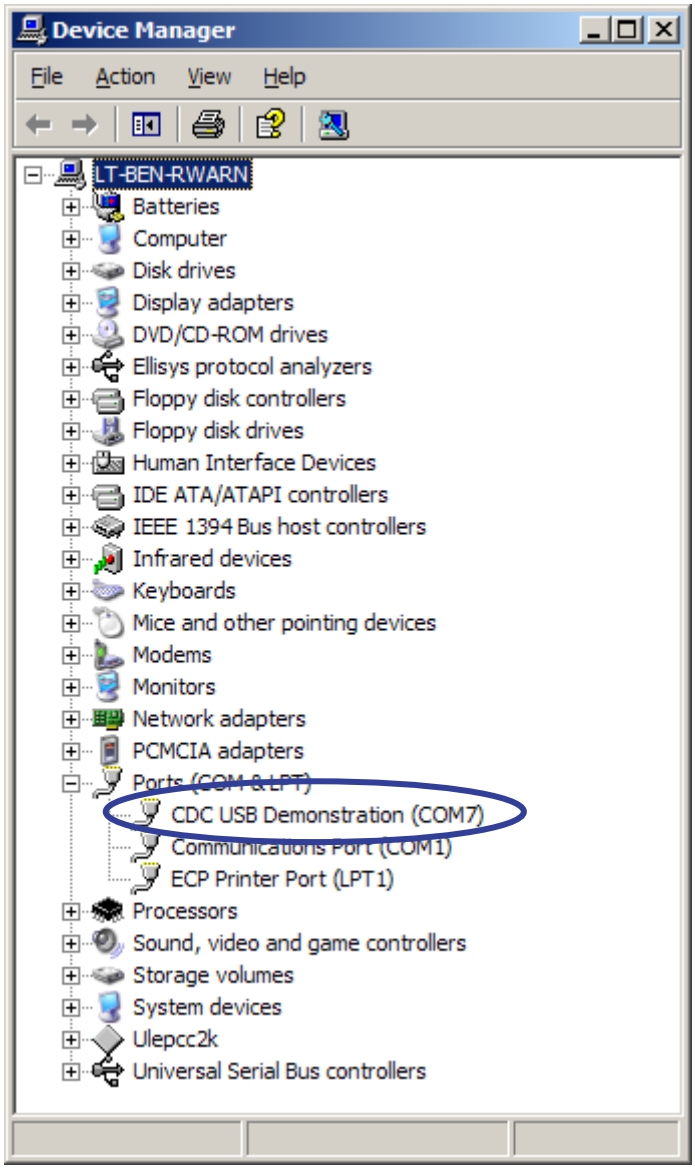


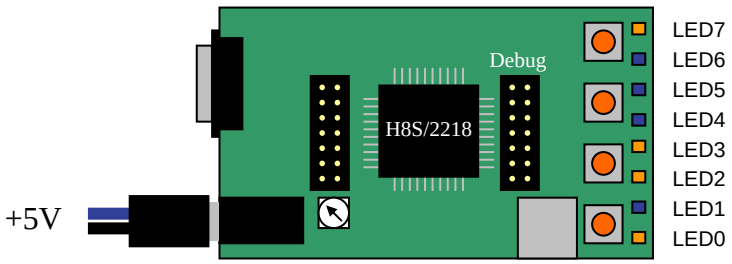
Figure 2.

To communicate to the H8S open a terminal emulator, such as HyperTerminal.

Configure the appropriate COM port for 152000 Baud, 8-N-1.

When a character is transmitted to the H8S, the ASCII value of a character will be displayed via the 3DK LEDs, as shown in figure 3. In AppNoteTwo the character value is echoed back to the PC and will be displayed. When any of the three switches on the 3DK board are pressed they generate an IRQ: Switch 1 – IRQ2, Switch 2 – IRQ4, Switch 3 – IRQ7. The IRQ ISRs transmit a fixed text

string to the host PC. Figure 4 shows an example screen shot of HyperTerminal communicating to the H8S/2218 running the application AppNoteTwo.



Value displayed is 0x72, character ‘r’

Figure 3.

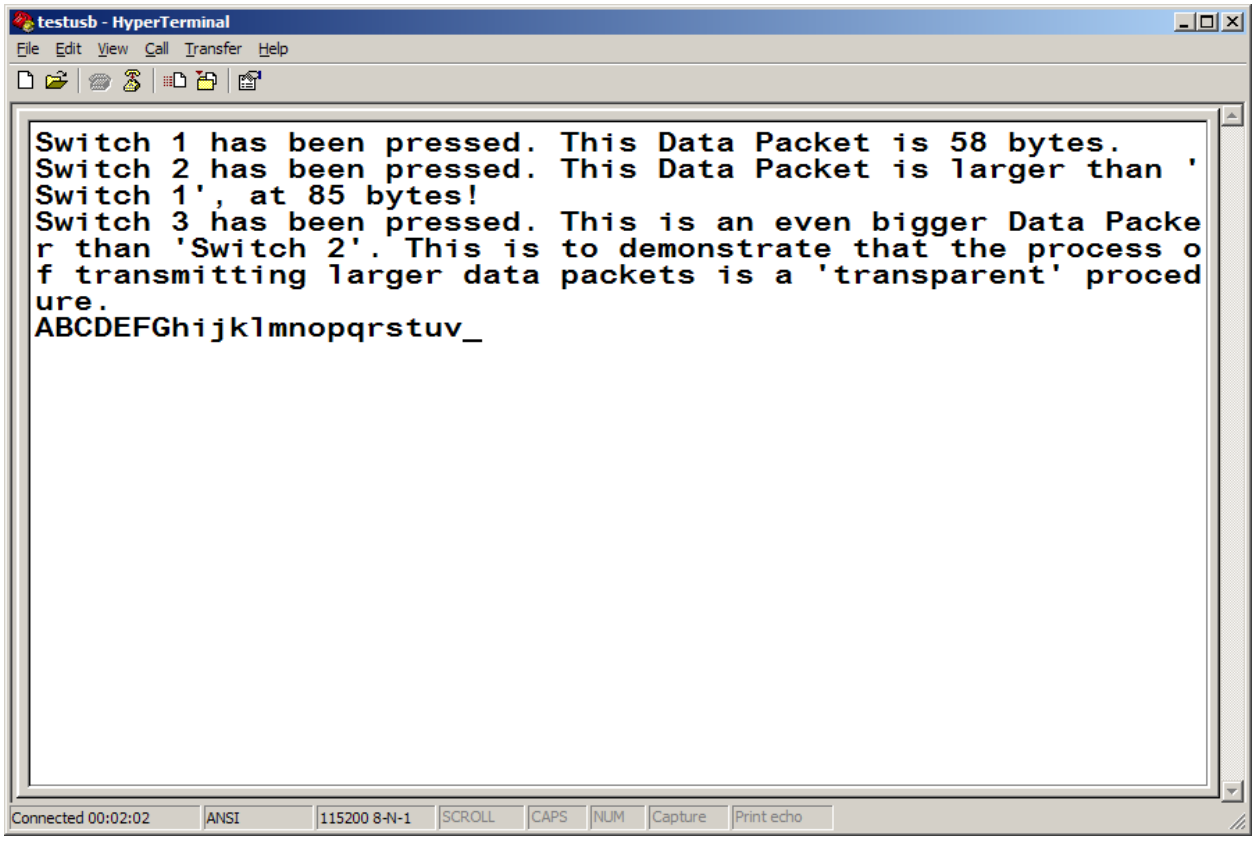


Figure 4.

H8S USB Operation

When the H8S is connected to the host PC, the device will perform enumeration. This process is detailed in the previous section. When the H8S has enumerated, the data can be transferred to and from the host PC.

Using the HyperTerminal application it is possible to transmit and receive data to and from the H8S. When data is sent to the H8S it is transmitted via Bulk Transfer. Endpoint 2 is used on the H8S for Bulk Out transfers. When data is successfully received in UEDR2 (USB endpoint data register 2) the EP2READY flag generates EXIRQ1[2].

In response to the EP2READY interrupt, the function `HandleEP2Ready()` is called.

This function copies the received data form the EndPoint2 Bulk-In data register to a local store, `BULKDataOUT.Data[]`.

It is also possible for the H8S to send data to the host PC. Bulk Transfer via End Point 1 transmits the data to the host PC. This can be demonstrated by pressing any of the three switches. The three switches are connected to three of the IRQ lines. In response to a switch being pressed, a constant text string is transmitted to the host PC.

Switch	***External Interrupt***	***Text String Transmitted***
SW1	IRQ2	"Switch 1 has been pressed. This Data Packet is 58 bytes.\r\n"
SW2	IRQ4	"Switch 2 has been pressed. This Data Packet is larger than 'Switch 1', at 85 bytes!\r\n"
SW3	IRQ7	"Switch 3 has been pressed. This is an even bigger Data Packer than 'Switch 2'. This is to demonstrate that the process of transmitting larger data packets is a 'transparent' procedure.\r\n"

Table 1.

The IRQx ISR initiates two data pointers, one to the start of the data string and the other to the end of the data string.

The IRQx ISR then calls a function `TransmitData();` This function enables the EP1EMPTYE (End Point 1 FIFO Empty) interrupt. The EP1EMPTY flag indicates that the EP1 FIFO is empty and can accept data.

In response to the EP1EMPTYE flag being set the function `WriteBULKINPacket()` is called. This function copies data into the EP1 FIFO and then sets the EP1PKTE bit. This generates a trigger to enable the transmission to EP1 FIFO.

The USB peripheral now performs the necessary USB protocol handling to transmit the data.

[2] REFER TO FUNCTION VOID HANDLECLOCKOK(VOID) FOR CONFIGURATION OF USB INTERRUPTS

Application Three: HID Demo

Overview

Application Three allows the user to control the RTC, ADC and TPU peripherals of the H8S via a PC application. Commands to Start, Stop & Configure the peripherals are transmitted to and from the host PC using Interrupt Transfer and HID Reports.

Figure 4 shows a screen shot of the PC application which can be used for controlling the H8S.

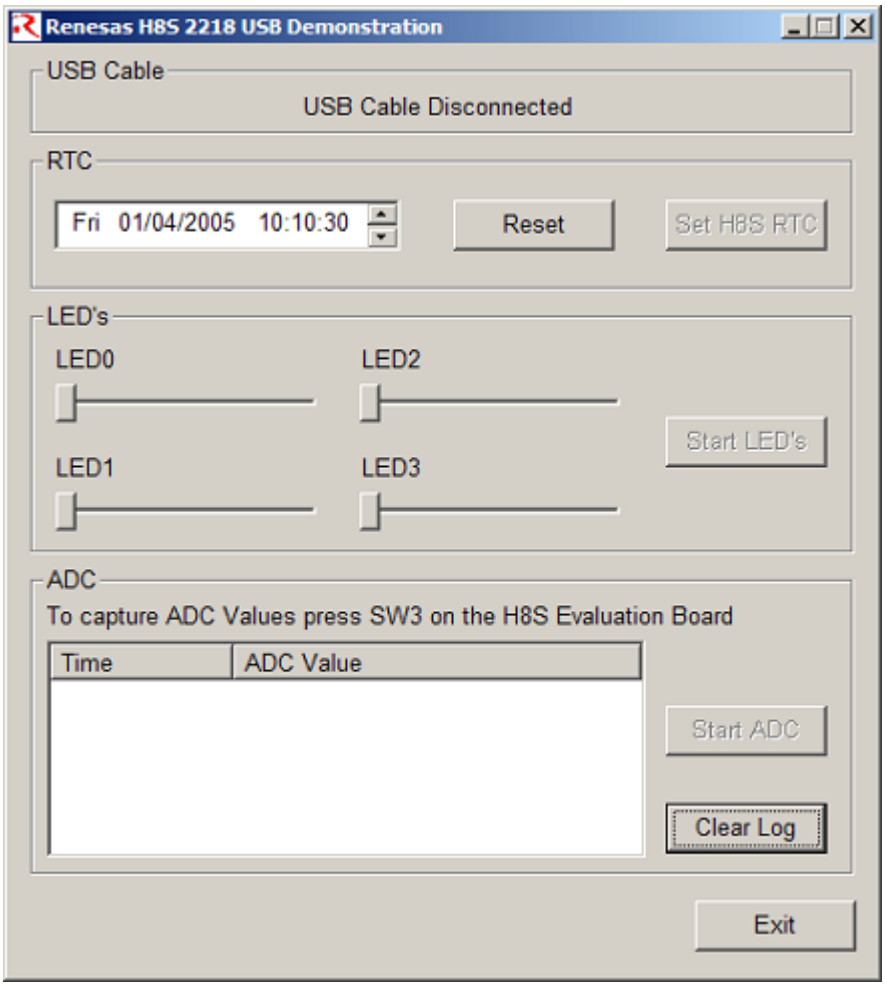


Figure 5.

When the USB cable is connected the H8S will enumerate. As the application uses the standard HID class then no .inf file is required, all of the information describing the H8S is provided by the H8S during the enumeration process. The information is held in the files `usbdescriptors.c` and `usbdescriptors.h`. One of the parameters, which the enumeration process provides, is the report size. In this application, the report size is specified as 6 bytes for In & Out transfers.

If the enumeration process is successful, then the application will report the fact by displaying the message as shown in figure 5.

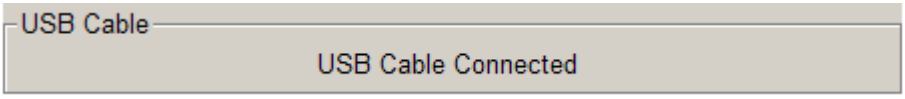


Figure 6.

As stated previously it is possible to control the RTC, TPU and ADC peripherals of the H8S by transmitting HID reports to the H8S device. On reception of these reports, the H8S decodes the data and responds accordingly.

RTC

When the PC application starts it will display the current system time in the RTC date-time picker.

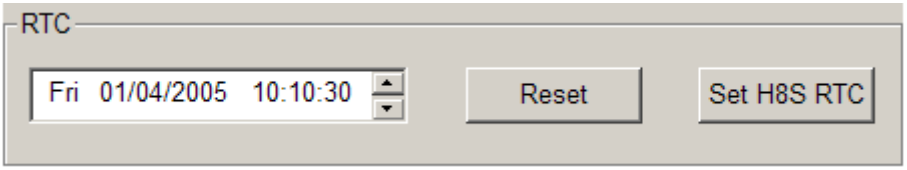


Figure 7.

It is possible to specify any date by manipulating the date and time. Pressing the Reset button will display the current system time.

When a time has been chosen, press the Set H8S RTC button.

This will transmit the date and time to the H8S. On successful reception the H8S will configure the RTC with the transmitted data and start the RTC.

TPU

The H8S application pre configures TPU0 to generate a 1 ms interrupt when it starts running. TPU0 can be started (and stopped) by pressing the Start LED's / Stop LED's button.

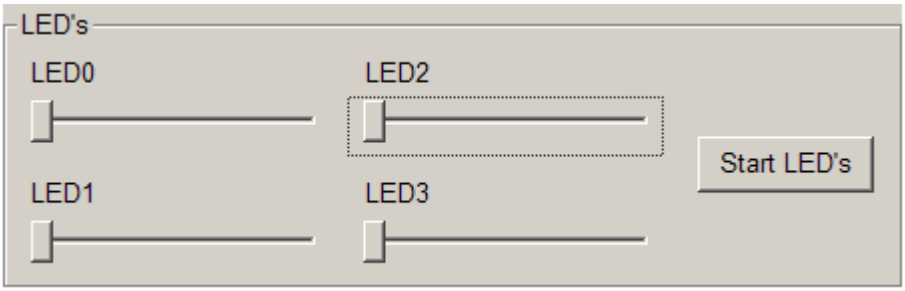


Figure 8.

With the slider in the left-hand position, as shown above, the LED's will not show any activity. Moving the sliders to the right will start LED activity.

ADC

The RTC and TPU functions utilise OUT Transfers to transfer data from the PC to theH8S. The ADC option utilises both IN & OUT Transfers. Pressing the Start ADC button performs an OUT transfer with the command to start the ADC. Pressing Switch 3 on the 3DK board causes the H8S to perform an Interrupt IN transfer with the results of the ADC conversion and the time at which the ADC operation occurred, by way of the RTC.

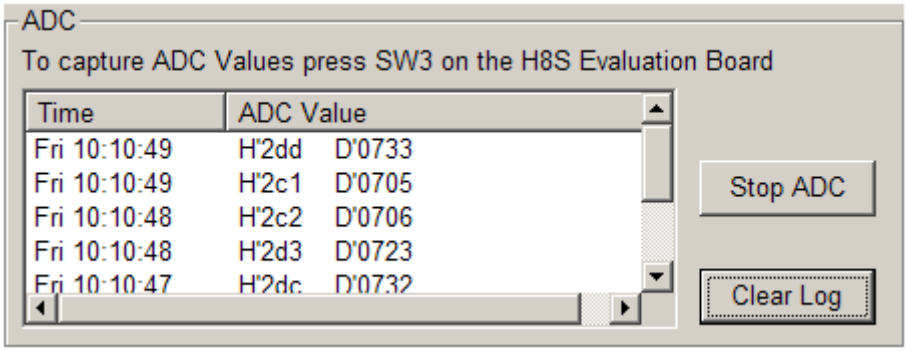


Figure 9.

H8S USB Operation

When the H8S is connected to the host PC, the device will perform enumeration. This process is detailed in the previous section. When the H8S has enumerated, the data can be transferred to and from the host PC.

When data are successfully received by the H8S the function `HandleEP0oTS()` will be called. This copies the data from Endpoint EP0o, Control_out tranfer to a local store, `g_OutputReport[]` and sets a software flag `USBDataReceivedFlag`. The software flag is continuously monitored by the application and when it is set, the received data are processed. The format of the data received is shown in table 2.

Local Store for received data.	Header Byte	Data Bytes				
	Byte[0]	Byte[1]	Byte[2]	Byte[3]	Byte[4]	Byte[5]
`g_OutputReport[6];`	ADC ‘0’	-	-	-	-	Start / Stop
	LED ‘1’	LED0 Val	LED1 Val	LED2 Val	LED3 Val	Start / Stop
	RTC ‘2’	Seconds	Minutes	Hours	Days	-

Table 2.

In response to an ‘ADC’ command, `g_OutputReport[0] == 0x00`, ADC operation is either started, `g_OutputReport[5] == 0x55`, or stopped, `g_OutputReport[5] == 0xAA`.

In response to an ‘LED’ command, `g_OutputReport[0] == 0x01`, the 16-bit timer TPU0 is either started, `g_OutputReport[5] == 0x55`, or stopped, `g_OutputReport[5] == 0xAA`. Additionally four compare match values are transmitted. These compare match values determine the toggle rate of the LED’s.

The toggle rate for LED 0 is specified by `g_OutputReport[1]`.
The toggle rate for LED 1 is specified by `g_OutputReport[2]`.
The toggle rate for LED 2 is specified by `g_OutputReport[3]`.
The toggle rate for LED 3 is specified by `g_OutputReport[4]`.

When TPU0 is started, it generates an interrupt every 1ms. The TPU0 ISR, `#pragma interrupt INT_TGI0A void INT_TGI0A(void)`, compares the toggle rate value with an incremental counter and toggles the LEDs appropriately.

In response to an 'RTC' command, `g_OutputReport[0] == 0x02`, the function `SetRTC(g_OutputReport)` is called, which configures the RTC with the received data and then starts the RTC running.

The 'Seconds' value is specified by `g_OutputReport[1]`.
The 'Minutes' value is specified by `g_OutputReport[2]`.
The 'Hours' value is specified by `g_OutputReport[3]`.
The 'Day of Week' value is specified by `g_OutputReport[4]`.

When switch 'SW3' on the 3DK board is pressed the IRQ3 interrupt will be generated. The IRQ3 ISR, `#pragma interrupt INT_IRQ3 void INT_IRQ3 (void)` will, if the ADC is running, transmit the result of the ADC conversion and the current RTC value to the host PC.

The format of the data transmitted is shown in table 3.

Local Store for transmit data.	ADC Bytes		Data Bytes			
	Byte[0]	Byte[1]	Byte[2]	Byte[3]	Byte[4]	Byte[5]
`g_InputReport[6];`	ADC H	ADC L	Seconds	Minutes	Hours	Days

Table 3.

The IRQ3 ISR stores the data to be transmitted to the PC in a local store, `g_InputReport[6]`. It then calls the function `WriteEP3INData()`. This function copies the data from the local store to the EP3 FIFO and then sets the EP3PKTE bit. This generates a trigger to enable the transmission to EP3 FIFO.

The USB peripheral now performs the necessary USB protocol handling to transmit the data.

Appendix A:

Function Listings:

```
void HardwareSetup(void)
void USBPreInitSetup(void)
__interrupt(vect=104) void INT_EXIRQ0_USB(void)
void HandleClockOK(void)
void HandleVBus(void)
void HandleBusReset(void)
void HandleSetupCmd(void)
void ReadSetupPacket(void)
BOOL DecodeSetupPacket(void)
BOOL DecodeStandardSetupPacket(void)
BOOL GetDescriptorString(void)
BOOL DecodeClassSetupPacket(void)

__interrupt(vect=18) void INT_IRQ2(void)
__interrupt(vect=20) void INT_IRQ4(void)
__interrupt(vect=23) void INT_IRQ7(void)
void TransmitData(void)
void WriteBULKINPacket(void)

void HandleEP0oTS(void)
void SetRTC(unsigned char *TimeData)
void ResetRTC(void)
void RunRTC(void)
#pragma interrupt INT_IRQ7 void INT_IRQ7 (void)
void WriteEP3INData(void)
```

```
void HardwareSetup(void)
{
    // Clock Pulse Generator
    // HW Manual says to Bypass PLL
    LPWCR.BIT.STC = 0x03;

    // Set interrupt priorities prior to setting Interrupt Mode 2
    INTC.IPRA.BIT._IRQ0 = 1;    // IRQ0
    INTC.IPRA.BIT._IRQ1 = 1;    // IRQ1

    INTC.IPRB.BIT._IRQ23 = 1;   // IRQ2,IRQ3
    INTC.IPRB.BIT._IRQ45 = 1;   // IRQ4,IRQ5

    INTC.IPRC.BIT._IRQ67 = 1;   // IRQ6 (USB Suspend/ Resume), IRQ7

    INTC.IPRD.BIT._WDT = 1;     // WDT0

    INTC.IPRE.BIT._AD = 1;      // A/D

    INTC.IPRF.BIT._TPU0 = 1;    // TPU0
    INTC.IPRF.BIT._TPU1 = 1;    // TPU1

    INTC.IPRG.BIT._TPU2 = 1;    // TPU2

    INTC.IPRJ.BIT._DMAC = 1;    // DMAC
    INTC.IPRJ.BIT._SCI0 = 1;    // SCI0

    INTC.IPRK.BIT._SCI2 = 1;    // SCI2

    INTC.IPRM.BIT._USB = 2;     // USB

    // Set Interrupt Mode 2
    SYSCR.BIT.INTM = 0x02;

    // Set pattern on LED's to show power up.
    PE.DDR = 0xFF;                   // Output port
    PE.DR.BYTE = 0x55;               //(Active Low)

    // Enable IRQs for the switches
    INTC.ISCR.BIT.IRQ2SC = 0x01;    // Falling edge
    INTC.ISCR.BIT.IRQ4SC = 0x01;    // Falling edge
    INTC.ISCR.BIT.IRQ7SC = 0x01;    // Falling edge
    INTC.IER.BIT.IRQ2E = 1;         // SW1
    INTC.IER.BIT.IRQ4E = 1;         // SW2
    INTC.IER.BIT.IRQ7E = 1;         // SW3
}
```

```
void USBPreInitSetup(void)
{
    // Ensure that the Addr / Data BUS is configured correctly
    BSC.ABWCR.BIT.ABW6 = 1;     // 0: 16 bit access
                                // 1: 8 bit access

    BSC.ASTCR.BIT.AST6 = 1;     // 0: 2 state access
                                // 1: 3 state access

    BSC.WCR.BIT.W6 = 0;         // 0: 0 Wait states
           // 1: 1 Wait state
           // 2: 2 Wait states
           // 3: 3 Wait states

    // P36 controlls the USB D+
    // Set as output
    P3.DR.BIT.B6 = 0;           // Disable D+ Pull-up until after VBUS
           // interrupt
    P3.DDR = 0x40;

    // USB
    // Cancel USB Extended Module Stop1
    EXMDLSTP.BIT.USBSTOP1 = 0; // Can now access UCTLR & UIER3 registers









    // Doubling 24MHz main clock generates the USB 48MHz clock.
    USB.UCTLR.BIT.UCKS = 0x06; // 0: USB operation stops (both 48MHz & PLL )
           // 1 - 5: Reserved
           // 6: Clock = 24MHz x 2.
           // 7: Clock = 16MHz x 3.
           // 8 – 15: Reserved

    // Cancel USB module stop
    MSTP.CRB.BIT._USB = 0;      // Can now access the remaining USB registers

    // Now wait for the USB Clock stabilisation interrupt, which is enabled by
    // default.
}
```

```
__interrupt(vect=104) void INT_EXIRQ0_USB(void)
{
    if((USB.UIFR3.BYTE & 0x80) && (USB.UIER3.BYTE & 0x80)) {
        HandleClockOK();        // USB Operating Clock Stabilisation Detected
    }
    else if(USB.UIFR3.BIT.VBUSi) {
        HandleVBus();           // VBUS State change (USB Cable (Dis)Connect)
    }
    else if(USB.UIFR0.BIT.BRST) {
        HandleBusReset();       // Bus Reset signal detected
    }
    else if(USB.UIFR0.BIT.SetupTS) {
        HandleSetupCmd();       // Setup command receive complete
    }
    else if(USB.UIFR0.BIT.EP0iTS) {
        HandleEP0iTS();         // EP0i has transmitted OK
    }
    else if(USB.UIFR0.BIT.EP0oTS) {
        HandleEP0oTS();         // EP0o Receive Complete
    }
        else if(USB.UIFR0.BIT.EP0iTR) {
        HandleEP0iTR();         // EP0i Transfer Request
    }
}
```

```
void HandleClockOK(void)
{
    //Temp
    //Set pattern on LEDs to show progress.
    PE.DR.BYTE = ~0x01; //(Active Low)

    //Cancel USB Interface SW Reset
    USB.UCTLR.BIT.UIFRST = 0;

    //Clear the Interrupt Flag
    //CLEAR_CK48READY
    USB.UIFR3.BYTE = ~0x80;

    //Enable Interrupts
    USB.UIER0.BIT.BRSTE = 1;      // BUS Reset
    USB.UIER0.BIT.EP0iTSE = 1;    // End Point 0 - IN Transmit Complete
              // (CONTROL IN)
    USB.UIER0.BIT.EP0oTSE = 1;    // End Point 0 - OUT Receive Complete
              // (CONTROL OUT)
    USB.UIER0.BIT.SetupTSE = 1;  // Setup Command Receive Complete

    USB.UIER1.BIT.EP2READYE = 1; // End Point 2 Data Ready (BULK OUT)

    USB.UIER3.BIT.VBUSiE = 1;     // VBUS - Cable Connect / Disconnect

    // Enable Stall Cancellation mode for all endpoints so FW doesn't have to
    USB.UESTL1.BIT.SCME = 1;

    // Select Interrupt
    USB.UISR0.BYTE = 0x00;      // EXIRQ0
    USB.UISR1.BYTE = 0xFF;      // EXIRQ1 Keep BULK interrupts separate
    USB.UISR3.BYTE = 0x00;      // EXIRQ0
}
```

```
void HandleVBus(void)
{
    //Clear the Interrupt Flag
    //CLEAR_VBUSi
    USB.UIFR3.BYTE = ~0x01;

    //Temp
    //Set pattern on LEDs to show progress.
    PE.DR.BYTE = ~0x03; //(Active Low)

    if(USB.UIFR3.BIT.VBUSs) {
        //USB Bus Connected

        //Temp
        //Set pattern on LEDs to show progress.
        PE.DR.BYTE = ~0x07; //(Active Low)

        //Clear all FIFOs
        USB.UFCLR0.BYTE = 0x3E;

        //Enable D+ Pull-Up
        P3.DR.BIT.B6 = 1;

        //Cancel UDC core reset
        USB.UCTLR.BIT.UDCRST = 0;
    }
    else {
        //USB Bus Disconnected

        //Temp
        //Set pattern on LEDs to show progress.
        PE.DR.BYTE = ~0xFF; //(Active Low)

        //Disable D+ Pull-Up
        P3.DR.BIT.B6 = 0;

        //UDC core reset
        USB.UCTLR.BIT.UDCRST = 1;
    }
}
```

```
void HandleBusReset(void)
{
    //Clear the Interrupt Flag
    //CLEAR_BRST
    USB.UIFR0.BYTE = ~0x80;

    //Clear FIFOS
    USB.UFCLR0.BYTE = 0x3E;

    //Clear all stalls
    USB.UESTL0.BYTE = 0x00;

    //Temp
    //Set pattern on LEDs to show progress.
    PE.DR.BYTE = ~0x0F; //(Active Low)
}
```

```
void HandleSetupCmd(void)
{
    //Clear the Interrupt Flag
    //CLEAR_SetupTS
    USB.UIFR0.BYTE = ~0x01;

    //Register is not readable - so this will affect all bits!
    //Clear EP0i and EP0o FIFO
    USB.UFCLR0.BIT.EP0oCLR = 1;
    USB.UFCLR0.BIT.EP0iCLR = 1;

    //Read 8 byte Command into oUSBData
    ReadSetupPacket();

    if( DecodeSetupPacket() ) {
        //Is this a Data IN or OUT
        if( SetupData.access0Type.bmRequest.bitVal.d7 ) {
            // ----------IN----------

            // Disable EP0i TR Interrupt as only needed for Status stage
            // of control OUT
            USB.UIER0.BIT.EP0iTRE = 0;

            //Write a packet to the data register
            WriteControlINPacket();
        }
        else {
            // ----------OUT---------

            // Enable EP0i TR Interrupt needed for Status stage
            // of control OUT
            USB.UIER0.BIT.EP0iTRE = 1;
        }
    }
    else {
        // -- Can't handle this command so do a EP0 STALL --
        USB.UESTL0.BIT.EP0STL = 1;

        // Disable EP0i TR Interrupt as only needed for Status stage
        // of control OUT
        USB.UIER0.BIT.EP0iTRE = 0;
    }
}
```

```
void ReadSetupPacket(void)
{
    unsigned short wLength;

    //Longword transfer is allowed from byte register UEDR0s.
    SetupData.longVal[0] = USB.UEDR0s.LONG;
    SetupData.longVal[1] = USB.UEDR0s.LONG;

    //Correct length for endian issue
    //wLength isn't right because of endian issue - so swap bytes
    wLength = 0x00FF & (SetupData.access0Type.wLength >> 8);
    wLength |= 0xFF00 & (SetupData.access0Type.wLength << 8);
    SetupData.access0Type.wLength = wLength;
}
```

```
BOOL DecodeSetupPacket(void)
{
    BOOL bReturn;
    if(SetupData.access0Type.bmRequest.bitVal.d65 == REQUEST_STANDARD) {
        //Standard Type
        bReturn = DecodeStandardSetupPacket();
    }
    else if(SetupData.access0Type.bmRequest.bitVal.d65 == REQUEST_CLASS) {
        //Class
        bReturn = DecodeClassSetupPacket();
    }
    else if(SetupData.access0Type.bmRequest.bitVal.d65 == REQUEST_VENDOR) {
        //Vendor
        bReturn = 0;
    }

    if(bReturn) {
        //Check length of data
        if(SetupData.access0Type.wLength <
            (ControlDataIN.pEnd – ControlDataIN.pStart + 1)) {
        //Only send as much data as has been requested even though we have more
        ControlDataIN.pEnd
            = ControlDataIN.pStart + SetupData.access0Type.wLength - 1;
        }
    }

    //Set EP0 Read Complete
    USB.UTRG0.BIT.EP0sRDFN = 1;

    return bReturn;
}
```

```
BOOL DecodeStandardSetupPacket(void)
{
    BOOL bReturn = 0;

    switch(SetupData.access0Type.bRequest)
    {
        case GET_DESCRIPTOR:
        {
            switch(SetupData.access0Type.wValue & 0x00FF)
            {
            case DEVICE:
            {
                ControlDataIN.pStart = &DeviceDescriptor.pucData[0];
                ControlDataIN.pEnd =
                     &DeviceDescriptor.pucData[DeviceDescriptor.length-1];
                bReturn = 1;
                break;
            }
            case CONFIGURATION:
            {
                ControlDataIN.pStart = &ConfigurationDescriptor.pucData[0];
                ControlDataIN.pEnd =
            &ConfigurationDescriptor.pucData[ConfigurationDescriptor.length-1];
                bReturn = 1;
                break;
            }

            case STRING:
            {
                bReturn = GetDescriptorString();
                break;
            }
        }
        break;
        }
    }
return bReturn;
}
```

```
BOOL GetDescriptorString(void)
{
    BOOL bReturn = 0;

    // wIndex is the language ID.
    // If wIndex==0 then this is a request to get the language ids that we
    // support
    // The LowByte (==HighByte for us being big endian) is the string
    // descriptor index required.
    if(!SetupData.access0Type.wIndex) {
        //Language IDs
        ControlDataIN.pStart = &StringDescriptorLanguageIDs.pucData[0];
        ControlDataIN.pEnd =
&StringDescriptorLanguageIDs.pucData[StringDescriptorLanguageIDs.length-1];

        bReturn = 1;
    }
    else {
        //What string index is being requested
        switch((SetupData.access0Type.wValue >> 8) & 0x00FF)
        {
            case STRING_iMANUFACTURER:
            {
                ControlDataIN.pStart = &StringDescriptorManufacturer.pucData[0];
                ControlDataIN.pEnd =
&StringDescriptorManufacturer.pucData[StringDescriptorManufacturer.length-1];

                bReturn = 1;
                break;
            }
            case STRING_iPRODUCT:
            {
                ControlDataIN.pStart = &StringDescriptorProduct.pucData[0];
                ControlDataIN.pEnd =
&StringDescriptorProduct.pucData[StringDescriptorProduct.length-1];

                bReturn = 1;
                break;
            }
        }
    }
    return bReturn;
}
```

```
BOOL DecodeClassSetupPacket(void)
{
    BOOL bReturn = 0;

    switch(SetupData.access0Type.bRequest)
    {
        case GET_LINE_CODING:
        {
            ControlDataIN.pStart = &LineCodeing[0];
            ControlDataIN.pEnd = &LineCodeing[LINE_CODING_DATA_SIZE - 1];
            bReturn = 1;
            break;
        }
        case SET_LINE_CODING: //(Required for hyperterminal)
        case SET_CONTROL_LINE_STATE:
        {
            //No action required for this request.
            bReturn = 1;
            break;
        }
    }

    return bReturn;
}
```

```
void HandleEP2Ready(void)
{
    unsigned char Index = 0;
    unsigned char Data;

    BULKDataOUT.ReceivedNumber = USB.UESZ2 & 0x7f;

    while(USB.UESZ2 & 0x7f)                 // UESZ2 indicates the number of
                                            // bytes to be received from
    {                                       // the host.
        Data = USB.UEDR2.BYTE;              // Read data
        BULKDataOUT.Data[Index++] = Data;
        PE.DR.BYTE = ~Data;
    }

    //Read Complete
    USB.UTRG0.BIT.EP2RDFN = 1;

    // Software Flag to indicate that we have received data
    BULKDataOUT.ReceivedFlag = 1;
}
```

```
__interrupt(vect=18) void INT_IRQ2(void)
{
    unsigned long debounce, dummy;
    for( debounce=0; debounce<50000; debounce++ ) {
        dummy = INTC.ISR.BIT.IRQ2F;
    }

    INTC.ISR.BIT.IRQ2F = 0; // Clear Interrupt Flag - SW1

    //Set text to output
    BULKDataIN.pStart = szSwitch1;
    BULKDataIN.pEnd = BULKDataIN.pStart + strlen(szSwitch1) - 1;

    TransmitData();
}
```

```
__interrupt(vect=20) void INT_IRQ4(void)
{
    unsigned long debounce, dummy;
    for( debounce=0; debounce<50000; debounce++ ) {
        dummy = INTC.ISR.BIT.IRQ4F;
    }

    INTC.ISR.BIT.IRQ4F = 0; // Clear Interrupt Flag - SW2

    //Set text to output
    BULKDataIN.pStart = szSwitch2;
    BULKDataIN.pEnd = BULKDataIN.pStart + strlen(szSwitch2) - 1;

    TransmitData();
}
```

```
__interrupt(vect=23) void INT_IRQ7(void)
{
    unsigned long debounce, dummy;

    for( debounce=0; debounce<50000; debounce++ ) {
        dummy = INTC.ISR.BIT.IRQ7F;
    }

    INTC.ISR.BIT.IRQ7F = 0; // Clear Interrupt Flag - SW3

    //Set text to output
    BULKDataIN.pStart = szSwitch3;
    BULKDataIN.pEnd = BULKDataIN.pStart + strlen(szSwitch3) - 1;

    TransmitData();
}
```

```
void TransmitData(void)
{
    // Enable BULK IN FIFO empty interrupt -
    // When we handle this interrupt we send data to host.

    USB.UIER1.BIT.EP1EMPTYE = 1;           // End Point 1 FIFO Empty (BULK IN)
}
```

```
void WriteBULKINPacket(void)
{
    int iCount = 0;

    // Write data to the IN FIFO until we have written a full packet or we
    // have no more data to write
    while((iCount < BULK_IN_PACKET_SIZE) &&
                                (BULKDataIN.pStart <= BULKDataIN.pEnd))
    {
        USB.UEDR1.BYTE = *BULKDataIN.pStart;
        BULKDataIN.pStart++;
        iCount++;
    }

    // Packet Enable
    USB.UTRG0.BIT.EP1PKTE = 1;

    // Disable this interrupt if we have written all data
    if(BULKDataIN.pStart > BULKDataIN.pEnd) {
        USB.UIER1.BIT.EP1EMPTYE = 0;
    }
}
```

```
void HandleEP0oTS(void)
{
    unsigned char ucBytesReceived;

    // Clear the Interrupt Flag
    USB.UIFR0.BYTE = ~0x08;

    // Read from the receive data size register (UESZ0o)
    // If there is no data then this is a Status Stage Control IN
    ucBytesReceived = USB.UESZ0o & 0x7F;
    if(!ucBytesReceived) {
        //This is Status Stage Control IN so no data to read
    }
    else {
        // Data Stage Control Out
        unsigned char ucBytesRead = 0;

        // Assumption that this is an output report OK?
        // (Because we accepted SET_REPORT)
        // Read Data into g_OutputReport
        while( (ucBytesRead < ucBytesReceived) &&
                                (ucBytesRead < OUTPUT_REPORT_SIZE) ) {
            g_OutputReport[ucBytesRead] = USB.UEDR0o.BYTE;
            ucBytesRead++;
        }

        USBDataReceivedFlag = 1;
    }

    // EP0o Read complete
    USB.UTRG0.BIT.EP0oRDFN = 1;
}
```

```
#pragma interrupt INT_TGI0A
void INT_TGI0A( void )
{
    static unsigned char TickCount0 = 0;
    static unsigned char TickCount1 = 0;
    static unsigned char TickCount2 = 0;
    static unsigned char TickCount3 = 0;

    TPU0.TSR.BIT.TGFA = 0;                  // Clear flag

    if(FreqLED0 == 0xff){       // min value, so stop the LED
        PE.DR.BIT.B0 = 1;
    }
    else if(TickCount0 == FreqLED0){
        PE.DR.BIT.B0 = 1 - PE.DR.BIT.B0;
        TickCount0 = 0;
    }

    if(FreqLED1 == 0xff){       // min value, so stop the LED
        PE.DR.BIT.B1 = 1;
    }
    else if(TickCount1 == FreqLED1){
        PE.DR.BIT.B1 = 1 - PE.DR.BIT.B1;
        TickCount1 = 0;
    }

    if(FreqLED2 == 0xff){       // min value, so stop the LED
        PE.DR.BIT.B2 = 1;
    }
    else if(TickCount2 == FreqLED2){
        PE.DR.BIT.B2 = 1 - PE.DR.BIT.B2;
        TickCount2 = 0;
    }

    if(FreqLED3 == 0xff){       // min value, so stop the LED
        PE.DR.BIT.B3 = 1;
    }
    else if(TickCount3 == FreqLED3){
        PE.DR.BIT.B3 = 1 - PE.DR.BIT.B3;
        TickCount3 = 0;
    }

    TickCount0++;
    TickCount1++;
    TickCount2++;
    TickCount3++;
}
```

```
void SetRTC(unsigned char *TimeData)
{
    unsigned char Blank;

    ResetRTC();

    RTC.RTCCSR.BIT.CKSI = 8;    // 0 - clk/8 Free Running Counter operation
                                // 1 - clk/32 Free Running Counter operation
                                // 2 - clk/128 Free Running Counter operation
                                // 3 - clk/256 Free Running Counter operation
                                // 4 - clk/512 Free Running Counter operation
                                // 5 - clk/2048 Free Running Counter operation
                                // 6 - clk/4096 Free Running Counter operation
                                // 7 - clk/8192 Free Running Counter operation
                                // 8 - 32.768 kHz RTC Operation

    RTC.RTCCR1.BIT.MD = 1;      // 0 - 12 Hour Mode
                                // 1 - 24 Hour Mode

    // Disable all RTC Interrupts
    RTC.RTCCR2.BYTE = 0;

    // Enable / Disable Individual Interrupts
    // 1 - Enabled
    // 0 - Disabled
    RTC.RTCCR2.BIT.FOIE = 0;    // Free Running Counter Overflow
    RTC.RTCCR2.BIT.WKIE = 0;    // Week Periodic Interrupt
    RTC.RTCCR2.BIT.DYIE = 0;    // Day Periodic Interrupt
    RTC.RTCCR2.BIT.HRIE = 0;    // Hour Periodic Interrupt
    RTC.RTCCR2.BIT.MNIE = 0;    // Minute Periodic Interrupt
    RTC.RTCCR2.BIT.SEIE = 0;    // Second Periodic Interrupt


    Blank = *TimeData;
    TimeData++;                 // Increment pointer

    // Seconds Data
    while(RTC.RSECDR.BIT.BSY == 1);
    RTC.RSECDR.BIT.SC1 = *TimeData / 10;    // 10's
    RTC.RSECDR.BIT.SC0 = *TimeData % 10;    // 0's
    TimeData++;                             // Increment pointer

    // Minutes Data
    while(RTC.RMINDR.BIT.BSY == 1);
    RTC.RMINDR.BIT.MN1 = *TimeData / 10;    // 10's
    RTC.RMINDR.BIT.MN0 = *TimeData % 10;    // 0's
    TimeData++;                             // Increment pointer
```

```
    // Hours Data
    while(RTC.RHRDR.BIT.BSY == 1);
    RTC.RHRDR.BIT.HR1 = *TimeData / 10;    // 10's
    RTC.RHRDR.BIT.HR0 = *TimeData % 10;    // 0's
    TimeData++;                            // Increment pointer

    // Day of Week
    while(RTC.RWKDR.BIT.BSY == 1);
    RTC.RWKDR.BIT.WK = *TimeData % 10;

    RunRTC();
}
```

```
void ResetRTC(void)
{
    RTC.RTCCR1.BIT.RST = 1;
    RTC.RTCCR1.BIT.RST = 0;
}
```

```
void RunRTC(void)
{
    RTC.RTCCR1.BIT.RUN = 1;
}
```

```
#pragma interrupt INT_IRQ7
void INT_IRQ7 (void)
{
    union{
        unsigned short Word;
        struct{
            unsigned short HByte:8;
            unsigned short LByte:8;
        }Byte;
    }AdcResult;

    INTC.ISR.BIT.IRQ7F = 0;                 // Clear Flag

    if(AD.ADCSR.BIT.ADST == 1)              // ADC running
    {
        AdcResult.Word = AD.ADDRA >> 6;

        g_InputReport[0] = AdcResult.Byte.HByte;
        g_InputReport[1] = AdcResult.Byte.LByte;

        // Read seconds
        while(RTC.RSECDR.BIT.BSY == 1);
        g_InputReport[2] = (RTC.RSECDR.BIT.SC1 * 10) + RTC.RSECDR.BIT.SC0;

        // Read Minutes
        while(RTC.RMINDR.BIT.BSY == 1);
        g_InputReport[3] = (RTC.RMINDR.BIT.MN1 * 10) + RTC.RMINDR.BIT.MN0;

        // Read Hours
        while(RTC.RHRDR.BIT.BSY == 1);
        g_InputReport[4] = (RTC.RHRDR.BIT.HR1 * 10) + RTC.RHRDR.BIT.HR0;

        // Read Day of Week
        while(RTC.RWKDR.BIT.BSY == 1);
        g_InputReport[5] = RTC.RWKDR.BIT.WK;

        WriteEP3INData();
    }
}
```

```
void WriteEP3INData(void)
{
    unsigned char Index;

    //Check EP3 is empty before write more data?
    if(USB.UDSR.BIT.EP3DE == 0){
        for(Index=0; Index<INPUT_REPORT_SIZE; Index++) {
            USB.UEDR3.BYTE = g_InputReport[Index];
        }
        //Packet Enable
        USB.UTRG0.BIT.EP3PKTE = 1;
    }
}
```

Appendix B:

File Listings:

`USBDescriptors.h`	For HID Class Application
`USBDescriptors.c`	For HID Class Application
`USBDescriptors.h`	For COM Class Application
`USBDescriptors.c`	For COM Class Application

```
//
// $Workfile: usbdescriptors.h $
// $Revision: 1 $
//
// Description: USB Descriptors for H8S2218, HID class.
//
// (c) 2004 Renesas Technology Europe Limited, All Rights Reserved



#ifndef FILENAME_USBDESCRIPTORS_H
#define FILENAME_USBDESCRIPTORS_H

#include "USB.h"

#define CONTROL_IN_PACKET_SIZE 0x40
#define HID_REPORT_DESCRIPTOR_SIZE 24

#define START_INDEX_OF_HID_WITHIN_CONFIG_DESC 18
#define END_INDEX_OF_HID_WITHIN_CONFIG_DESC   26

#define STRING_iMANUFACTURER   1
#define STRING_iPRODUCT                   2

// Device Descriptor
extern const DiscriptorType gDeviceDescriptor;

// Configuration, Interface, HID and Endpoint Descriptor
extern const DiscriptorType gConfigurationDescriptor;

// Hid Report Descriptor
extern const DiscriptorType gHIDReportDescriptor;

// String descriptors
extern const DiscriptorType gStringDescriptorLanguageIDs;
extern const DiscriptorType gStringDescriptorManufacturer;
extern const DiscriptorType gStringDescriptorProduct;

#endif //FILENAME_USBDESCRIPTORS_H
```

```
//
// $Workfile: USBDescriptors.c $
// $Revision: 1 $
//
// Description: USB Descriptors for H8S2218, HID class.
//            This describes an input report of 2 bytes and an output report of
2 bytes.
//
//
// (c) 2004 Renesas Technology Europe Limited, All Rights Reserved

#include "USBDescriptors.h"

//  Device Descriptor
//  ""Attention"" Please use your company Vendor ID
const unsigned char gDeviceDescriptorData[] =
{
    0x12,                       // Size of this descriptor
    0x01,                       // Device Descriptor
    0x10,0x01,                  // USB Version 1.10
    0x00,                       // Class Code - None as HID is defined in the
                                // Interface Descriptor
    0x00,                       // Subclass Code
    0x00,                       // Protocol Code
    CONTROL_IN_PACKET_SIZE,     // Max Packet Size for endpoint 0
    0x5B,0x04,                  // Vendor ID = 0x045B
    0x10,0x00,                  // Product ID = 0x0010
    0x00,0x01,                  // Device Release Number
    STRING_iMANUFACTURER,       // Manufacturer String Descriptor
    STRING_iPRODUCT,            // Product String Descriptor
    0x00,                       // No Serial Number String Descriptor
    0x01                        // Number of Configurations supported
};

const DiscriptorType           gDeviceDescriptor =
{
    18, gDeviceDescriptorData
};

// Configuration Descriptor
// For HID this includes Interface Descriptors, HID descriptor and endpoint
// descriptor.
// Ensure START_INDEX_OF_HID_WITHIN_CONFIG_DESC and
// END_INDEX_OF_HID_WITHIN_CONFIG_DESC are defined
const unsigned char gConfigurationDescriptorData[] =
{
    0x09,      // Size of this descriptor (Just the configuration part)
    0x02,      // Configuration Descriptor
    0x22,0x00, // Combined length of all descriptors that make this up h'0022
    0x01,      // Number of interfaces
    0x01,      // This Interface Value
    0x00,      // No String Descriptor for this configuration
    0x80,      // bmAttributes - Bus Powered, No Remote Wakeup
    0x1E,      // h'1E * 2 mA* = 60mA

    0x09,      // Size of this descriptor
    0x04,      // INTERFACE Descriptor
```

```
    0x00,      // Index of Interface
    0x00,      // bAlternateSetting
    0x01,      // Number of Endpoints
    0x03,      // HID Class code
    0x00,      // No Subclass
    0x00,      // No Protocol
    0x00,      // No String Descriptor for this interface

    0x09,      // Size of this descriptor
    0x21,      // HID Descriptor
    0x11,0x01, // HID Class Specification Release Number 1.11
    0x00,      // No Target Country
    0x01,      // Number of HID Descriptors
    0x22,      // Type of HID Descriptor = "Report"
    HID_REPORT_DESCRIPTOR_SIZE,00, // Length Of HID Report Descriptor

    0x07,      // Size of this descriptor
    0x05,      // ENDPOINT Descriptor
    0x83,      // bEndpointAddress - IN endpoint, endpoint number = 3
    0x03,      // Endpoint Type is Interrupt
    0x08,0x00, // Max Packet Size
    0x0A       // Polling Interval in mS
};


const DiscriptorType gConfigurationDescriptor =
{
    34, gConfigurationDescriptorData
};

// Report Descriptor
// NOTE The size of this must be HID_REPORT_DESCRIPTOR_SIZE
const unsigned char gHIDReportDescriptorData[] =
{
    0x06, 0x00, 0xFF,          // Usage Page Vendor
    0x09, 0x00,                // Usage something (other than 0 is required if
                               // using undefined page)
    0xA1, 0x01,                // Collection App (seem to need a collection?)
// Input Report
    0x09, 0x00,                // Usage Undefined
    0x75, 0x08,                // Data received is 8 Bits in Size
    0x95, INPUT_REPORT_SIZE,   // Report Count
    0x25, 0xFF,                // Max 255
    0x15, 0x00,                // Min 0
    0x81, 0x02,                // Input
// Output Report
    0x09, 0x00,                // Usage Undefined
    0x91, OUTPUT_REPORT_SIZE,  // Report Count

    0xC0                       //End collection
};

const DiscriptorType gHIDReportDescriptor =
{
    HID_REPORT_DESCRIPTOR_SIZE, // Must match value specified in HID descriptor!
    gHIDReportDescriptorData
};
```

```
//String Descriptors

#define STRING_LANGUAGE_IDS_SIZE 0x04
const unsigned char gStringDescriptorLanguageIDsData[] =
{
    STRING_LANGUAGE_IDS_SIZE,   //Length of this
    0x03,                       //Descriptor Type = STRING
    // NOT LIKED BY USB VIEW ?! 0x09, 0x08 //0x0809 == English UK
    0x09, 0x04                  //0x0409 == English USA
};

const DiscriptorType  gStringDescriptorLanguageIDs =
{
    STRING_LANGUAGE_IDS_SIZE,
    gStringDescriptorLanguageIDsData
};

#define STRING_MANUFACTURER_SIZE 16
const unsigned char gStringDescriptorManufacturerData[] = //"Renesas"
{
    STRING_MANUFACTURER_SIZE,   // Length of this
    0x03,                       // Descriptor Type = STRING
    'R', 0x00,                  // Unicode
    'E', 0x00,
    'N', 0x00,
    'E', 0x00,
    'S', 0x00,
    'A', 0x00,
    'S', 0x00
};

const DiscriptorType  gStringDescriptorManufacturer =
{
    STRING_MANUFACTURER_SIZE,
    gStringDescriptorManufacturerData
};

#define STRING_PRODUCT_SIZE 44
const unsigned char gStringDescriptorProductData[] = // "HID USB Demonstration"
{
    STRING_PRODUCT_SIZE,        // Length of this
    0x03,                       // Descriptor Type = STRING
    'H', 0x00,                  // Unicode
    'I', 0x00,
    'D', 0x00,
    ' ', 0x00,
    'U', 0x00,
    'S', 0x00,
    'B', 0x00,
    ' ', 0x00,
    'D', 0x00,
    'e', 0x00,
    'm', 0x00,
    'o', 0x00,
    'n', 0x00,
    's', 0x00,
```

```
    't', 0x00,
    'r', 0x00,
    'a', 0x00,
    't', 0x00,
    'i', 0x00,
    'o', 0x00,
    'n', 0x00
};

const DiscriptorType  gStringDescriptorProduct =
{
    STRING_PRODUCT_SIZE,
    gStringDescriptorProductData
};
```

```
//*****************************************************************************
// file: USBDescriptors.c
//
// Description: USB Descriptors for H8S2218, CDC class.
//
// (c) 2004 Renesas Technology Europe Limited, All Rights Reserved
// ****************************************************************************

#include "USBDescriptors.h"

//  Device Descriptor
//  ATTENTION: Please use your company Vendor ID
unsigned char DeviceDescriptorData[] =
{
    0x12,                       // Size of this descriptor
    0x01,                       // Device Descriptor
    0x10,0x01,                  // USB Version 1.10
    0x02,                       // Class Code - CDC
    0x00,                       // Subclass Code
    0x00,                       // Protocol Code
    CONTROL_IN_PACKET_SIZE,     // Max Packet Size for endpoint 0
    0x5B,0x04,                  // Vendor ID = 0x045B
    0x11,0x00,                  // Product ID = 0x0011
    0x00,0x01,                  // Device Release Number
    STRING_iMANUFACTURER,       // Manufacturer String Descriptor
    STRING_iPRODUCT,            // Product String Descriptor
    0x00,                       // No Serial Number String Descriptor
    0x01                        // Number of Configurations supported
};

DiscriptorTypeDeviceDescriptor = {        //  This part is
                                          // deviceDiscriptorGVar[0]
    18, DeviceDescriptorData
};

//Configuration Descriptor
unsigned char ConfigurationDescriptorData[] =
{
    0x09,      //Size of this descriptor (Just the configuration part)
    0x02,      //Configuration Descriptor
    0x43,0x00, //Combined length of all descriptors
    0x02,      //Number of interfaces
    0x01,      //This Interface Value
    0x00,      //No String Descriptor for this configuration
    0x80,      //bmAttributes - Bus Powered, No Remote Wakeup
    0x1E,      //h'1E * 2 mA* = 60mA

    // Communication Class Interface Descriptor
    0x09,      //Size of this descriptor
    0x04,      //INTERFACE Descriptor
    0x00,      //Index of Interface
    0x00,      //bAlternateSetting
    0x01,      //Number of Endpoints
    0x02,      //Class code = Communication
    0x20,      //Subclass = Abstract Control Model
    0x00,      //No Protocol
    0x00,      //No String Descriptor for this interface
```

```
//Header Functional Descriptor
0x05,      //bFunctionalLength
0x24,      //bDescriptorType = CS_INTERFACE
0x00,      //bDescriptor Subtype = Header
0x10,0x01, //bcdCDC 1.1

//ACM Functional Descriptor
0x04,      //bFunctionalLength
0x24,      //bDescriptorType = CS_INTERFACE
0x02,      //bDescriptor Subtype = Abstract Control Management
0x02,      //bmCapabilities GET_LINE_CODING etc supported

//Union Functional Descriptor
0x05,      //bFunctionalLength
0x24,      //bDescriptorType = CS_INTERFACE
0x06,      //bDescriptor Subtype = Union
0x00,      //bMasterInterface = Communication Class Interface
0x01,      //bSlaveInterface = Data Class Interface

//Call Management Functional Descriptor
0x05,      //bFunctionalLength
0x24,      //bDescriptorType = CS_INTERFACE
0x01,      //bDescriptor Subtype = Call Management
0x00,      //bmCapabilities
0x01,      //bDataInterface: Data Class Interface = 1

//Interrupt Endpoint
0x07,      //Size of this descriptor
0x05,      //ENDPOINT Descriptor
0x83,      //bEndpointAddress - IN endpoint, endpoint number = 3
0x03,      //Endpoint Type is Interrupt
0x08,0x00, //Max Packet Size
0xFF,      //Polling Interval in mS

// DATA Class Interface Descriptor
0x09,      //Size of this descriptor
0x04,      //INTERFACE Descriptor
0x01,      //Index of Interface
0x00,      //bAlternateSetting
0x02,      //Number of Endpoints
0x0A,      //Class code = Data Interface
0x00,      //Subclass = Abstract Control Model
0x00,      //No Protocol
0x00,      //No String Descriptor for this interface

//Endpoint Bulk OUT
0x07,      //Size of this descriptor
0x05,      //ENDPOINT Descriptor
0x02,      //bEndpointAddress - OUT endpoint, endpoint number = 2
0x02,      //Endpoint Type is BULK
64,0x00,   //Max Packet Size
0x00,      //Polling Interval in mS - IGNORED FOR BULK

//Endpoint Bulk IN
0x07,      //Size of this descriptor
0x05,      //ENDPOINT Descriptor
```

```
    0x81,      //bEndpointAddress - IN endpoint, endpoint number = 1
    0x02,      //Endpoint Type is BULK
    64,0x00,   //Max Packet Size
    0x00       //Polling Interval in mS - IGNORED FOR BULK
};

DiscriptorType ConfigurationDescriptor = {
    0x43, ConfigurationDescriptorData
};

//String Descriptors

#define STRING_LANGUAGE_IDS_SIZE 0x04
unsigned char StringDescriptorLanguageIDsData[] =
{
    STRING_LANGUAGE_IDS_SIZE,  // Length of this
    0x03,                      //Descriptor Type = STRING
    // NOT LIKED BY USB VIEW ?! 0x09, 0x08 //0x0809 == English UK
    0x09, 0x04                             //0x0409 == English USA
};

DiscriptorType  StringDescriptorLanguageIDs =
{
    STRING_LANGUAGE_IDS_SIZE,
    StringDescriptorLanguageIDsData
};

#define STRING_MANUFACTURER_SIZE 16
unsigned char StringDescriptorManufacturerData[] = //"Renesas"
{
    STRING_MANUFACTURER_SIZE,  // Length of this
    0x03,                      // Descriptor Type = STRING
    'R', 0x00,
    'E', 0x00,
    'N', 0x00,
    'E', 0x00,
    'S', 0x00,
    'A', 0x00,
    'S', 0x00                  //Unicode
};

DiscriptorType  StringDescriptorManufacturer =
{
    STRING_MANUFACTURER_SIZE,
    StringDescriptorManufacturerData
};

#define STRING_PRODUCT_SIZE 44
unsigned char StringDescriptorProductData[] = //"CDC USB Demonstration"
 {
    STRING_PRODUCT_SIZE,       //Length of this
    0x03,                      //Descriptor Type = STRING
    'C', 0x00, 'D', 0x00, 'C', 0x00, ' ', 0x00,
    'U', 0x00, 'S', 0x00, 'B', 0x00, ' ', 0x00,
    'D', 0x00,
    'e', 0x00,
    'm', 0x00,
```

```
    'o', 0x00,
    'n', 0x00,
    's', 0x00,
    't', 0x00,
    'r', 0x00,
    'a', 0x00,
    't', 0x00,
    'i', 0x00,
    'o', 0x00,
    'n', 0x00                   //Unicode
};

DiscriptorType  StringDescriptorProduct =
{
    STRING_PRODUCT_SIZE,
    StringDescriptorProductData
};
```

```
//*****************************************************************************
// file: USBDescriptors.h
//
// Description: USB Descriptors for H8S2218, CDC class.
//
// (c) 2004 Renesas Technology Europe Limited, All Rights Reserved
// *****************************************************************************/

#ifndef FILENAME_USBDESCRIPTORS_H
#define FILENAME_USBDESCRIPTORS_H

#include "USB.h"

#define CONTROL_IN_PACKET_SIZE            0x40
#define BULK_IN_PACKET_SIZE               0x40

#define STRING_iMANUFACTURER              1
#define STRING_iPRODUCT                   2

//Device Descriptor
extern DiscriptorType DeviceDescriptor;

//Configuration, Interface and Endpoint Descriptor
extern DiscriptorType ConfigurationDescriptor;

//String descriptors
extern DiscriptorType StringDescriptorLanguageIDs;
extern DiscriptorType StringDescriptorManufacturer;
extern DiscriptorType StringDescriptorProduct;

#endif //FILENAME_USBDESCRIPTORS_H
```

Website and Support

Renesas Technology Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry

Notes regarding these materials

1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
2. Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (http://www.renesas.com)
5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guaranties regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
8. Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below:
   (1) artificial life support devices or systems
   (2) surgical implantations
   (3) healthcare intervention (e.g., excision, administration of medication, etc.)
   (4) any other purposes that pose a direct threat to human life

   Renesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.
9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.