

```

//-----
// File:      fw.c
// Contents:  Firmware frameworks task dispatcher and device request parser
//           source.
//
// Copyright (c) 1997 AnchorChips, Inc. All rights reserved
//-----
#include "ezusb.h"
#include "ezregs.h"
//-----
// Needed Macros
//-----
#define min(a,b) (((a)<(b))?a:(b))
#define max(a,b) (((a)>(b))?a:(b))
//-----
// Constants
//-----
#define DELAY_COUNT 0x9248*8L      // Delay for 8 sec at 24MHz, 4 sec at 48
//-----
// Global Variables
//-----
volatile BOOL  GotSUD;
BOOL          Rwuen;
BOOL          Selfpwr;
volatile BOOL  Sleep; // Sleep mode enable flag
WORD         pDeviceDscr; // Pointer to Device Descriptor; Descriptors may be moved
WORD         pConfigDscr;
WORD         pStringDscr;

//-----
// Prototypes
//-----
void SetupCommand(void);
void TD_Init(void);
void TD_Poll(void);
BOOL TD_Suspend(void);
BOOL TD_Resume(void);
BOOL DR_GetDescriptor(void);
BOOL DR_SetConfiguration(void);
BOOL DR_GetConfiguration(void);
BOOL DR_SetInterface(void);
BOOL DR_GetInterface(void);
BOOL DR_GetStatus(void);
BOOL DR_ClearFeature(void);
BOOL DR_SetFeature(void);
BOOL DR_VendorCmnd(void);
BOOL DR_ClassRequest(void);

//-----
// Code
//-----
// Task dispatcher
void main(void) {
    DWORD  i;
    WORD   offset;
    DWORD  DevDescrLen;
    DWORD  j=0;
    WORD   IntDescrAddr;
    WORD   ExtDescrAddr;

// Initialize Global States
    Sleep = FALSE;           // Disable sleep mode
    Rwuen = FALSE;          // Disable remote wakeup
    Selfpwr = FALSE;        // Disable self powered
    GotSUD = FALSE;         // Clear "Got setup data" flag

// Initialize user device
    TD_Init();

// The following section of code is used to relocate the descriptor table.
// Since the SUDPTRH and SUDPTL are assigned the address of the descriptor
// table, the descriptor table must be located in on-part memory.
// The 4K demo tools locate all code sections in external memory.
// The descriptor table is relocated by the frameworks ONLY if it is found
// to be located in external memory.
    pDeviceDscr = (WORD)&DeviceDscr;
    pConfigDscr = (WORD)&ConfigDscr;
    pStringDscr = (WORD)&StringDscr;
    if ((WORD)&DeviceDscr & 0xe000) {

```



```

        SUDPTRL = LSB(pDeviceDscr);
        break;
    case GD_CONFIGURATION: // Configuration = 0x02
        if(dscr_ptr = (void *)EZUSB_GetConfigDscr(SETUPDAT[2])) {
            SUDPTRH = MSB(dscr_ptr);
            SUDPTRL = LSB(dscr_ptr);
        }
        else EZUSB_STALL_EP0(); // Stall End Point 0
        break;
    case GD_STRING: // String = 0x03
        if(dscr_ptr = (void *)EZUSB_GetStringDscr(SETUPDAT[2])) {
// Workaround for rev D errata number 8. If you're certain that you will never run on rev D you can just do
// this:
            SUDPTRH = MSB(dscr_ptr);
            SUDPTRL = LSB(dscr_ptr);
// Arm a 0 length packet just in case. There was some reflector traffic about Apple hosts asking for too much
// data.
// This will keep them happy and will not hurt valid hosts because the next SETUP will clear this.
            EZUSB_SET_EP_BYTES(INOBUF_ID,0);
            EPOCS = bmHS; // Clear the HS-nak bit
        }
        else EZUSB_STALL_EP0(); // Stall End Point 0
        break;
    default: // Invalid request
        EZUSB_STALL_EP0(); // Stall End Point 0
    }
    break;
case SC_GET_INTERFACE: // *** Get Interface 0x0a
    DR_GetInterface();
    break;
case SC_SET_INTERFACE: // *** Set Interface 0x0b
    DR_SetInterface();
    break;
case SC_SET_CONFIGURATION: // *** Set Configuration 0x09
    DR_SetConfiguration();
    break;
case SC_GET_CONFIGURATION: // *** Get Configuration 0x08
    DR_GetConfiguration();
    break;
case SC_GET_STATUS: // *** Get Status 0x00
    if(DR_GetStatus())
        switch(SETUPDAT[0]) {
            case GS_DEVICE: // Device 0x80
                INOBUF[0] = ((BYTE)Rwuen << 1) | (BYTE)Selfpwr;
                INOBUF[1] = 0;
                EZUSB_SET_EP_BYTES(INOBUF_ID,2);
                break;
            case GS_INTERFACE: // Interface 0x81
                INOBUF[0] = 0;
                INOBUF[1] = 0;
                EZUSB_SET_EP_BYTES(INOBUF_ID,2);
                break;
            case GS_ENDPOINT: // End Point 0x82
                INOBUF[0] = EPIO[EPID(SETUPDAT[4])].cntrl & bmEPSTALL;
                INOBUF[1] = 0;
                EZUSB_SET_EP_BYTES(INOBUF_ID,2);
                break;
            default: EZUSB_STALL_EP0(); // Invalid Command, Stall End Point 0
        }
    break;
case SC_CLEAR_FEATURE: // *** Clear Feature 0x01
    if(DR_ClearFeature()) switch(SETUPDAT[0]) {
        case FT_DEVICE: // Device 0x00
            if(SETUPDAT[2] == 1) Rwuen = FALSE; // Disable Remote Wakeup
            else EZUSB_STALL_EP0(); // Stall End Point 0
            break;
        case FT_ENDPOINT: // End Point 0x02
            if(SETUPDAT[2] == 0) {
                EZUSB_UNSTALL_EP( EPID(SETUPDAT[4]) );
                EZUSB_RESET_DATA_TOGGLE( SETUPDAT[4] );
            }
            else EZUSB_STALL_EP0(); // Stall End Point 0
            break;
    }
    break;
case SC_SET_FEATURE: // *** Set Feature 0x03
    if(DR_SetFeature())
        switch(SETUPDAT[0]) {
            case FT_DEVICE: // Device 0x00

```

```

        if(SETUPDAT[2] == 1) Rwuen = TRUE;    // Enable Remote Wakeup
        else EZUSB_STALL_EP0();             // Stall End Point 0
        break;
    case FT_ENDPOINT:                       // End Point 0x02
        if(SETUPDAT[2] == 0) EZUSB_STALL_EP( EPID(SETUPDAT[4]) );
        else EZUSB_STALL_EP0();            // Stall End Point 0
        break;
    }
    break;
default:                                    // *** Invalid Command
    EZUSB_STALL_EP0();                     // Stall End Point 0
}
break;
case SETUP_VENDOR_REQUEST:                 //Vendor Request 0x40
    if(DR_VendorCmdnd()) EZUSB_STALL_EP0(); // Stall End Point 0
    break;
case SETUP_CLASS_REQUEST:                 //Class Request 0x20
    if(DR_ClassRequest()) EZUSB_STALL_EP0();// Stall End Point 0
    break;
default: //Reserved or illegal
    EZUSB_STALL_EP0();                     // Stall End Point 0
    break;
}
// Acknowledge handshake phase of device request
// Required for rev C does not effect rev B
EPOCS |= bmBIT1;
}

// Wake-up interrupt handler
void resume_isr(void) interrupt WKUP_VECT {
    EZUSB_CLEAR_RSMIRQ();
}

```

```

.....

#pragma NOIV                               // Do not generate interrupt vectors
//-----
// File:      ButtonsAndLights.c
// Contents:  Hooks required to implement USB peripheral function.
//
// Copyright (c) 2000 CYPRESS SEMICONDUCTORS
//-----
#include "ezusb.h"
#include "ezregs.h"

#define min(a,b) (((a)<(b))?a):(b)

#define GD_HID 0x21
#define GD_REPORT 0x22
#define CR_SET_REPORT 0x09
#define HID_OUTPUT_REPORT 2

extern BOOL GotSUD;           // Received setup data flag
extern BOOL Sleep;

WORD    pHIDDscr;
WORD    pReportDscr;
WORD    pReportDscrEnd;
extern code HIDDscr;
extern code ReportDscr;
extern code ReportDscrEnd;

void TD_Poll(void);

BYTE    Configuration;      // Current configuration
BYTE    AlternateSetting;   // Alternate settings
BYTE    LEDValue;           // LED Mask

//-----
// Task Dispatcher hooks
// The following hooks are called by the task dispatcher.
//-----

void TD_Init(void) {        // Called once at startup
// Enable endpoint 2 in
    IN07VAL |= bmEP1;      // Validate all EP's
    PORTACFG = 0x00;        // Port A pins are I/O
    PORTBCFG = 0x00;        // Port B pins are I/O
}

```

```

OEA = 0x00;           // Port A is an input
OEB = 0xFF;           // Port B is an Output

USBBAV |= 0x01;      // Enable Autovectoring

OUT07IEN &= ~0xFF;   // Disenable all Endpoint Interrupts
IN07IEN = 0x01;      // Enable Endpoint 0 IN
USBIEEN = 0x13;      // Enable resume, Setup data available, and SOF
EIE = 0x01;          // Enable EUSB
IE = 0x80;           // Enable Global Interrupt

LEDValue = 0;        // Initialize LEDValue
USBIRQ = 0x01;       // Set setup data available interrupt request bit
}

void TD_Poll(void) {   // Called repeatedly while the device is idle
}

BOOL TD_Suspend(void) { // Called before the device goes into suspend mode
// Turn off breakpoint light before entering suspend
USBBAV |= bmBREAK;    // Clear the breakpoint
return(TRUE);
}

BOOL TD_Resume(void) { // Called after the device resumes
return(TRUE);
}

//-----
// Device Request hooks
// The following hooks are called by the end point 0 device request parser.
//-----

BOOL DR_ClassRequest(void) {
switch(SETUPDAT[1]) {
case CR_SET_REPORT:
switch (SETUPDAT[3]) {
case HID_OUTPUT_REPORT:
while (EPOCS & 0x04); // Wait for not busy
LEDValue = OUT0BUF[0]; // Save led states
OUTB= LEDValue;        // Output the LED Mask on Port B
OUT0BC = 0;           // Rearm endpoint buffer
return (FALSE);
default:
return (TRUE);
}
default: return(TRUE);
}
}

BOOL DR_GetDescriptor(void) {
BYTE length,i;

pHIDDscr = (WORD)&HIDDscr;
pReportDscr = (WORD)&ReportDscr;
pReportDscrEnd = (WORD)&ReportDscrEnd;

switch (SETUPDAT[3]) {
case GD_HID: //HID Descriptor
SUDPTRH = MSB(pHIDDscr);
SUDPTRL = LSB(pHIDDscr);
return (FALSE);
case GD_REPORT: //Report Descriptor
length = pReportDscrEnd - pReportDscr;

while (length) {
for(i=0; i<min(length,64); i++) *(IN0BUF+i) = *((BYTE xdata *)pReportDscr+i);
}

//set length and arm Endpoint
EZUSB_SET_EP_BYTES(IN0BUF_ID,min(length,64));
length -= min(length,64);

// Wait for it to go out (Rev C and above)
while(EPOCS & 0x04)
;
}
return (FALSE);
default: return(TRUE);
}

```

```

    }
}

BOOL DR_SetConfiguration(void) { // Called when a Set Configuration command is received
    Configuration = SETUPDAT[2];
    return(TRUE); // Handled by user code
}

BOOL DR_GetConfiguration(void){ // Called when a Get Configuration command is received
    IN0BUF[0] = Configuration;
    EZUSB_SET_EP_BYTES(IN0BUF_ID,1);
    return(TRUE); // Handled by user code
}

BOOL DR_SetInterface(void) { // Called when a Set Interface command is received
    AlternateSetting = SETUPDAT[2];
    return(TRUE); // Handled by user code
}

BOOL DR_GetInterface(void) { // Called when a Set Interface command is received
    IN0BUF[0] = AlternateSetting;
    EZUSB_SET_EP_BYTES(IN0BUF_ID,1);
    return(TRUE); // Handled by user code
}

BOOL DR_GetStatus(void) {
    return(TRUE);
}

BOOL DR_ClearFeature(void) {
    return(TRUE);
}

BOOL DR_SetFeature(void) {
    return(TRUE);
}

BOOL DR_VendorCmnd(void) {
    return(TRUE);
}

//-----
// USB Interrupt Handlers
// The following functions are called by the USB interrupt jump table.
//-----

// Setup Data Available Interrupt Handler
void ISR_Sudav(void) interrupt 0 {
    GotSUD = TRUE; // Set flag
    EZUSB_IRQ_CLEAR();
    USBIRQ = bmSUDAV; // Clear SUDAV IRQ
}

// Setup Token Interrupt Handler
void ISR_Sutok(void) interrupt 0 {
    EZUSB_IRQ_CLEAR();
    USBIRQ = bmSUTOK; // Clear SUTOK IRQ
}

void ISR_Sof(void) interrupt 0 {
    if( !(EPIO[IN1BUF_ID].cntrl & bmEPBUSY) ) { // Is the IN2BUF available,
        IN1BUF[0] = PINS_A; // Create a report
        IN1BC = 0x01; // Arm Enpoint 1
    }
    EZUSB_IRQ_CLEAR();
    USBIRQ = bmSOF; // Clear SOF IRQ
}

void ISR_Ures(void) interrupt 0 {
    EZUSB_IRQ_CLEAR();
    USBIRQ = bmURES; // Clear URES IRQ
}

void ISR_IBN(void) interrupt 0 {
    // ISR for the IN Bulk NAK (IBN) interrupt.
}

void ISR_Susp(void) interrupt 0 {
    Sleep = TRUE;
}

```

```
PCON |=0x01;
WRITEDELAY();
EZUSB_IRQ_CLEAR();
USBIRQ = bmSUSP;
}
```

```
void ISR_Ep0in(void) interrupt 0 {
    IN07IRQ |= 0x0100;
}
```

```
void ISR_Ep0out(void) interrupt 0 {
}
```

```
void ISR_Ep1in(void) interrupt 0 {
}
```

```
void ISR_Ep1out(void) interrupt 0 {
}
```

```
void ISR_Ep2in(void) interrupt 0 {
}
```

```
void ISR_Ep2out(void) interrupt 0 {
}
```

```
void ISR_Ep3in(void) interrupt 0 {
}
```

```
void ISR_Ep3out(void) interrupt 0 {
}
```

```
void ISR_Ep4in(void) interrupt 0 {
}
```

```
void ISR_Ep4out(void) interrupt 0 {
}
```

```
void ISR_Ep5in(void) interrupt 0 {
}
```

```
void ISR_Ep5out(void) interrupt 0 {
}
```

```
void ISR_Ep6in(void) interrupt 0 {
}
```

```
void ISR_Ep6out(void) interrupt 0 {
}
```

```
void ISR_Ep7in(void) interrupt 0 {
}
```

```
void ISR_Ep7out(void) interrupt 0 {
}
```