```
#include "Platform.h"
#include "USBAPI.h"
#include "USBDesc.h"

#if defined(USBCON)
#ifdef HID_ENABLED

//#define RAWHID_ENABLED

//  Singletons for mouse and keyboard

Mouse_ Mouse;
Keyboard_ Keyboard;
Joystick_ Joystick;

//================================================================================
//================================================================================

//  HID report descriptor

#define LSB(_x) ((_x) & 0xFF)
#define MSB(_x) ((_x) >> 8)

#define RAWHID_USAGE_PAGE   0xFFC0
#define RAWHID_USAGE        0x0C00
#define RAWHID_TX_SIZE 64
#define RAWHID_RX_SIZE 64
```

```
extern const u8 _hidReportDescriptor[] PROGMEM;
const u8 _hidReportDescriptor[] = {

    //  Mouse
    0x05, 0x01,                    // USAGE_PAGE (Generic Desktop)  // 54
    0x09, 0x02,                    // USAGE (Mouse)
    0xa1, 0x01,                    // COLLECTION (Application)
    0x09, 0x01,                    //   USAGE (Pointer)
    0xa1, 0x00,                    //   COLLECTION (Physical)
    0x85, 0x01,                    //     REPORT_ID (1)
    0x05, 0x09,                    //     USAGE_PAGE (Button)
    0x19, 0x01,                    //     USAGE_MINIMUM (Button 1)
    0x29, 0x03,                    //     USAGE_MAXIMUM (Button 3)
    0x15, 0x00,                    //     LOGICAL_MINIMUM (0)
    0x25, 0x01,                    //     LOGICAL_MAXIMUM (1)
    0x95, 0x03,                    //     REPORT_COUNT (3)
    0x75, 0x01,                    //     REPORT_SIZE (1)
    0x81, 0x02,                    //     INPUT (Data,Var,Abs)
    0x95, 0x01,                    //     REPORT_COUNT (1)
    0x75, 0x05,                    //     REPORT_SIZE (5)
    0x81, 0x03,                    //     INPUT (Cnst,Var,Abs)
    0x05, 0x01,                    //     USAGE_PAGE (Generic Desktop)
    0x09, 0x30,                    //     USAGE (X)
    0x09, 0x31,                    //     USAGE (Y)
    0x09, 0x38,                    //     USAGE (Wheel)
    0x15, 0x80,                    //     LOGICAL_MINIMUM (-128)
    0x25, 0x7f,                    //     LOGICAL_MAXIMUM (127)
    0x75, 0x08,                    //     REPORT_SIZE (8)
    0x95, 0x03,                    //     REPORT_COUNT (3)
    0x81, 0x06,                    //     INPUT (Data,Var,Rel)
    0xc0,                          //   END_COLLECTION
    0xc0,                          // END_COLLECTION

    //  Keyboard
    0x05, 0x01,                    // USAGE_PAGE (Generic Desktop)  // 47
    0x09, 0x06,                    // USAGE (Keyboard)
    0xa1, 0x01,                    // COLLECTION (Application)
    0x85, 0x02,                    //   REPORT_ID (2)
    0x05, 0x07,                    //   USAGE_PAGE (Keyboard)

    0x19, 0xe0,                    //   USAGE_MINIMUM (Keyboard LeftControl)
    0x29, 0xe7,                    //   USAGE_MAXIMUM (Keyboard Right GUI)
    0x15, 0x00,                    //   LOGICAL_MINIMUM (0)
    0x25, 0x01,                    //   LOGICAL_MAXIMUM (1)
```

```
    0x75, 0x01,                    //   REPORT_SIZE (1)

    0x95, 0x08,                    //   REPORT_COUNT (8)
    0x81, 0x02,                    //   INPUT (Data,Var,Abs)
    0x95, 0x01,                    //   REPORT_COUNT (1)
    0x75, 0x08,                    //   REPORT_SIZE (8)
    0x81, 0x03,                    //   INPUT (Cnst,Var,Abs)

    0x95, 0x06,                    //   REPORT_COUNT (6)
    0x75, 0x08,                    //   REPORT_SIZE (8)
    0x15, 0x00,                    //   LOGICAL_MINIMUM (0)
    0x25, 0x65,                    //   LOGICAL_MAXIMUM (101)
    0x05, 0x07,                    //   USAGE_PAGE (Keyboard)

    0x19, 0x00,                    //   USAGE_MINIMUM (Reserved (no event indicated))
    0x29, 0x65,                    //   USAGE_MAXIMUM (Keyboard Application)
    0x81, 0x00,                    //   INPUT (Data,Ary,Abs)
    0xc0,                          // END_COLLECTION

 // Atari JOYSTICK
   0x05, 0x01,                   // USAGE_PAGE (Generic Desktop)
   0x09, 0x04,                   // USAGE (Joystick)
   0xA1, 0x01,                 // Collection (Application)
   0x85, 0x04,                   // REPORT_ID (4)

   0x05, 0x01,                   // USAGE_PAGE (Generic Desktop)
   0x09, 0x01,                   // USAGE (Pointer)
   0xA1, 0x00,                   // Collection (Physical)
   0x09, 0x30,                   // USAGE (x)
   0x09, 0x31,                   // USAGE (y)
   0x15, 0x00,                   // LOGICAL_MINIMUM (0)
   0x26, 0xFF, 0x00,             // LOGICAL_MAXIMUM (255)
   0x75, 0x08,                   // REPORT_SIZE (8)
   0x95, 0x02,                   // REPORT_COUNT (2)
    0x81, 0x02,                   // INPUT (Data,Var,Abs)
    0xC0,                         // END_COLLECTION

    0x05, 0x09,                   // USAGE_PAGE (Button)
    0x19, 0x01,                   // USAGE_MINIMUM (button 1)
    0x29, 0x01,                   // USAGE_MAXIMUM (button 1)
    0x15, 0x00,                   // LOGICAL_MINIMUM (0)
    0x25, 0x01,                   // LOGICAL_MAXIMUM (1)
    0x75, 0x01,                   // REPORT_SIZE (1)
    0x95, 0x08,                   // REPORT_COUNT (8)
    0x55, 0x00,                   // UNIT_EXPONENT (0)
```

```
    0x65, 0x00,                    // UNIT (None)
    0x81, 0x02,                    // Input (Data,Var,Abs)
   0xC0              // end collection

#if RAWHID_ENABLED
    //  RAW HID
    0x06, LSB(RAWHID_USAGE_PAGE), MSB(RAWHID_USAGE_PAGE),   // 30
    0x0A, LSB(RAWHID_USAGE), MSB(RAWHID_USAGE),

    0xA1, 0x01,             // Collection 0x01
    0x85, 0x03,             // REPORT_ID (3)
    0x75, 0x08,             // report size = 8 bits
    0x15, 0x00,             // logical minimum = 0
    0x26, 0xFF, 0x00,       // logical maximum = 255

    0x95, 64,               // report count TX
    0x09, 0x01,             // usage
    0x81, 0x02,             // Input (array)

    0x95, 64,               // report count RX
    0x09, 0x02,             // usage
    0x91, 0x02,             // Output (array)
    0xC0                    // end collection
#endif
};

extern const HIDDescriptor _hidInterface PROGMEM;
const HIDDescriptor _hidInterface =
{
    D_INTERFACE(HID_INTERFACE,1,3,0,0),
    D_HIDREPORT(sizeof(_hidReportDescriptor)),
    D_ENDPOINT(USB_ENDPOINT_IN (HID_ENDPOINT_INT),USB_ENDPOINT_TYPE_INTERRUPT,0x40,0x01)
};

//================================================================================
//================================================================================
//  Driver

u8 _hid_protocol = 1;
u8 _hid_idle = 1;

#define WEAK __attribute__ ((weak))

int WEAK HID_GetInterface(u8* interfaceNum)
{
```

```
    interfaceNum[0] += 1;   // uses 1
    return USB_SendControl(TRANSFER_PGM,&_hidInterface,sizeof(_hidInterface));
}

int WEAK HID_GetDescriptor(int i)
{
    return USB_SendControl(TRANSFER_PGM,_hidReportDescriptor,sizeof(_hidReportDescriptor));
}

void WEAK HID_SendReport(u8 id, const void* data, int len)
{
    USB_Send(HID_TX, &id, 1);
    USB_Send(HID_TX | TRANSFER_RELEASE,data,len);
}

bool WEAK HID_Setup(Setup& setup)
{
    u8 r = setup.bRequest;
    u8 requestType = setup.bmRequestType;
    if (REQUEST_DEVICETOHOST_CLASS_INTERFACE == requestType)
    {
        if (HID_GET_REPORT == r)
        {
            //HID_GetReport();
            return true;
        }
        if (HID_GET_PROTOCOL == r)
        {
            //Send8(_hid_protocol); // TODO
            return true;
        }
    }

    if (REQUEST_HOSTTODEVICE_CLASS_INTERFACE == requestType)
    {
        if (HID_SET_PROTOCOL == r)
        {
            _hid_protocol = setup.wValueL;
            return true;
        }

        if (HID_SET_IDLE == r)
        {
            _hid_idle = setup.wValueL;
            return true;
```

```
        }
    }
    return false;
}

//================================================================================
//================================================================================
//  Mouse

Mouse_::Mouse_(void) : _buttons(0)
{
}

void Mouse_::begin(void)
{
}

void Mouse_::end(void)
{
}

void Mouse_::click(uint8_t b)
{
    _buttons = b;
    move(0,0,0);
    _buttons = 0;
    move(0,0,0);
}

void Mouse_::move(signed char x, signed char y, signed char wheel)
{
    u8 m[4];
    m[0] = _buttons;
    m[1] = x;
    m[2] = y;
    m[3] = wheel;
    HID_SendReport(1,m,4);
}

void Mouse_::buttons(uint8_t b)
{
    if (b != _buttons)
    {
        _buttons = b;
        move(0,0,0);
```

```
    }
}

void Mouse_::press(uint8_t b)
{
    buttons(_buttons | b);
}

void Mouse_::release(uint8_t b)
{
    buttons(_buttons & ~b);
}

bool Mouse_::isPressed(uint8_t b)
{
    if ((b & _buttons) > 0)
        return true;
    return false;
}

//================================================================================
//================================================================================
//  Keyboard

Keyboard_::Keyboard_(void)
{
}

void Keyboard_::begin(void)
{
}

void Keyboard_::end(void)
{
}

void Keyboard_::sendReport(KeyReport* keys)
{
    HID_SendReport(2,keys,sizeof(KeyReport));
}

extern
const uint8_t _asciimap[128] PROGMEM;

#define SHIFT 0x80
```

```
const uint8_t _asciimap[128] =
{
    0x00,             // NUL
    0x00,             // SOH
    0x00,             // STX
    0x00,             // ETX
    0x00,             // EOT
    0x00,             // ENQ
    0x00,             // ACK
    0x00,             // BEL
    0x2a,           // BS   Backspace
    0x2b,           // TAB  Tab
    0x28,           // LF   Enter
    0x00,             // VT
    0x00,             // FF
    0x00,             // CR
    0x00,             // SO
    0x00,             // SI
    0x00,             // DEL
    0x00,             // DC1
    0x00,             // DC2
    0x00,             // DC3
    0x00,             // DC4
    0x00,             // NAK
    0x00,             // SYN
    0x00,             // ETB
    0x00,             // CAN
    0x00,             // EM
    0x00,             // SUB
    0x00,             // ESC
    0x00,             // FS
    0x00,             // GS
    0x00,             // RS
    0x00,             // US

    0x2c,           //  ' '
    0x1e|SHIFT,     // !
    0x34|SHIFT,     // "
    0x20|SHIFT,     // #
    0x21|SHIFT,     // $
    0x22|SHIFT,     // %
    0x24|SHIFT,     // &
    0x34,           // '
    0x26|SHIFT,     // (
    0x27|SHIFT,     // )
```

```
    0x25|SHIFT,    // *
    0x2e|SHIFT,    // +
    0x36,          // ,
    0x2d,          // -
    0x37,          // .
    0x38,          // /
    0x27,          // 0
    0x1e,          // 1
    0x1f,          // 2
    0x20,          // 3
    0x21,          // 4
    0x22,          // 5
    0x23,          // 6
    0x24,          // 7
    0x25,          // 8
    0x26,          // 9
    0x33|SHIFT,      // :
    0x33,          // ;
    0x36|SHIFT,      // <
    0x2e,          // =
    0x37|SHIFT,      // >
    0x38|SHIFT,      // ?
    0x1f|SHIFT,      // @
    0x04|SHIFT,      // A
    0x05|SHIFT,      // B
    0x06|SHIFT,      // C
    0x07|SHIFT,      // D
    0x08|SHIFT,      // E
    0x09|SHIFT,      // F
    0x0a|SHIFT,      // G
    0x0b|SHIFT,      // H
    0x0c|SHIFT,      // I
    0x0d|SHIFT,      // J
    0x0e|SHIFT,      // K
    0x0f|SHIFT,      // L
    0x10|SHIFT,      // M
    0x11|SHIFT,      // N
    0x12|SHIFT,      // O
    0x13|SHIFT,      // P
    0x14|SHIFT,      // Q
    0x15|SHIFT,      // R
    0x16|SHIFT,      // S
    0x17|SHIFT,      // T
    0x18|SHIFT,      // U
    0x19|SHIFT,      // V
```

```
    0x1a|SHIFT,      // W
    0x1b|SHIFT,      // X
    0x1c|SHIFT,      // Y
    0x1d|SHIFT,      // Z
    0x2f,          // [
    0x31,          // bslash
    0x30,          // ]
    0x23|SHIFT,    // ^
    0x2d|SHIFT,    // _
    0x35,          // `
    0x04,          // a
    0x05,          // b
    0x06,          // c
    0x07,          // d
    0x08,          // e
    0x09,          // f
    0x0a,          // g
    0x0b,          // h
    0x0c,          // i
    0x0d,          // j
    0x0e,          // k
    0x0f,          // l
    0x10,          // m
    0x11,          // n
    0x12,          // o
    0x13,          // p
    0x14,          // q
    0x15,          // r
    0x16,          // s
    0x17,          // t
    0x18,          // u
    0x19,          // v
    0x1a,          // w
    0x1b,          // x
    0x1c,          // y
    0x1d,          // z
    0x2f|SHIFT,    //
    0x31|SHIFT,    // |
    0x30|SHIFT,    // }
    0x35|SHIFT,    // ~
    0               // DEL
};

uint8_t USBPutChar(uint8_t c);
```

```
// press() adds the specified key (printing, non-printing, or modifier)
// to the persistent key report and sends the report.  Because of the way
// USB HID works, the host acts like the key remains pressed until we
// call release(), releaseAll(), or otherwise clear the report and resend.
size_t Keyboard_::press(uint8_t k)
{
    uint8_t i;
    if (k >= 136) {         // it's a non-printing key (not a modifier)
        k = k - 136;
    } else if (k >= 128) {  // it's a modifier key
        _keyReport.modifiers |= (1<<(k-128));
        k = 0;
    } else {                // it's a printing key
        k = pgm_read_byte(_asciimap + k);
        if (!k) {
            setWriteError();
            return 0;
        }
        if (k & 0x80) {                     // it's a capital letter or other character reached with shift
            _keyReport.modifiers |= 0x02;   // the left shift modifier
            k &= 0x7F;
        }
    }

    // Add k to the key report only if it's not already present
    // and if there is an empty slot.
    if (_keyReport.keys[0] != k && _keyReport.keys[1] != k &&
        _keyReport.keys[2] != k && _keyReport.keys[3] != k &&
        _keyReport.keys[4] != k && _keyReport.keys[5] != k) {

        for (i=0; i<6; i++) {
            if (_keyReport.keys[i] == 0x00) {
                _keyReport.keys[i] = k;
                break;
            }
        }
        if (i == 6) {
            setWriteError();
            return 0;
        }
    }
    sendReport(&_keyReport);
    return 1;
}
```

```
// release() takes the specified key out of the persistent key report and
// sends the report.  This tells the OS the key is no longer pressed and that
// it shouldn't be repeated any more.
size_t Keyboard_::release(uint8_t k)
{
    uint8_t i;
    if (k >= 136) {         // it's a non-printing key (not a modifier)
        k = k - 136;
    } else if (k >= 128) {  // it's a modifier key
        _keyReport.modifiers &= ~(1<<(k-128));
        k = 0;
    } else {                // it's a printing key
        k = pgm_read_byte(_asciimap + k);
        if (!k) {
            return 0;
        }
        if (k & 0x80) {                         // it's a capital letter or other character reached with shift
            _keyReport.modifiers &= ~(0x02);    // the left shift modifier
            k &= 0x7F;
        }
    }

    // Test the key report to see if k is present.  Clear it if it exists.
    // Check all positions in case the key is present more than once (which it shouldn't be)
    for (i=0; i<6; i++) {
        if (0 != k && _keyReport.keys[i] == k) {
            _keyReport.keys[i] = 0x00;
        }
    }

    sendReport(&_keyReport);
    return 1;
}

void Keyboard_::releaseAll(void)
{
    _keyReport.keys[0] = 0;
    _keyReport.keys[1] = 0;
    _keyReport.keys[2] = 0;
    _keyReport.keys[3] = 0;
    _keyReport.keys[4] = 0;
    _keyReport.keys[5] = 0;
    _keyReport.modifiers = 0;
    sendReport(&_keyReport);
}
```

```
size_t Keyboard_::write(uint8_t c)
{
    uint8_t p = press(c);       // Keydown
    uint8_t r = release(c);     // Keyup
    return (p);                 // just return the result of press() since release() almost always returns 1
}


//================================================================================
//================================================================================
// Joystick
//  Usage: Joystick.move(x, y, throttle, buttons)
//  x & y forward/left = 0, centre = 127, back/right = 255
//  throttle max = 0, min = 255
//  8 buttons packed into 1 byte

Joystick_::Joystick_()
{
}

void Joystick_::move(uint8_t x, uint8_t y, uint8_t buttons)
{
    u8 j[3];
    j[0] = x;
    j[1] = y;
    j[2] = buttons;
    //HID_SendReport(Report number, array of values in same order as HID descriptor, length)
    HID_SendReport(4, j, 3);
}

#endif

#endif /* if defined(USBCON) */
```