7-9. USB HID Demo Cortex-M3 1권

2015.09.27. 15:34

http://blog.naver.com/nicepants/220493122704

PC에 설치하는 윈도우용 프로그램 : USB hID Demonstrator
STM32 USB 예제 프로그램을 설치하면 둘 간에 연동이 가능하다.

-
- 연결되면 윈도우에서는 HID 준수장치 + USB 휴먼 인터페이스가 1쌍 생긴다.
- 윈도우에서 USB 장치를 보면 장치 이름 + VID + PID가 보인다.
- usb_desc.c 파일에서 정의한 CustomHID_StringProduct + idVendor + idProduct 이다.

보드에서 버튼을 누르면 윈도우 프로그램에서 어느 버튼이 눌러졌는지 표시됨

- 버튼을 누른다 >> 인터럽트가 걸린다. >> 핸들러가 키값을 확인하고 USB를 통해 PC로 보낸다.

PC에서 LED를 설정하면 보드의 해당 LED가 켜짐

- PC에서 보내면 Callback 함수가 호출된다. >> LED를 켠다.

수신한 데이터를 가져오는 함수를 보자.

```
/*******************************************************************************
* Function Name  : PMAToUserBufferCopy
* Description    : Copy a buffer from user memory area to packet memory area (PMA)
* Input          : - pbUsrBuf    = pointer to user memory area. // PMA로 들어와 있는 데이터를 user memory area의 *pbUsrBuf 로 카피
*                  - wPMABufAddr = address into PMA. // PMA 내의 버퍼 주소
*                  - wNBytes     = no. of bytes to be copied.
* Output         : None.
* Return         : None.
*******************************************************************************/
void PMAToUserBufferCopy(uint8_t *pbUsrBuf, uint16_t wPMABufAddr, uint16_t wNBytes)
{
  uint32_t n = (wNBytes + 1) >> 1;/* /2*/
  uint32_t i;
  uint32_t *pdwVal;
  pdwVal = (uint32_t *)(wPMABufAddr * 2 + PMAAddr); //  PMA의 BASE에서 USB 레지스터값의 2배만큼 OFFSET 함을 주목
                                                    // 나중에 자세히 다룰 듯
  for (i = n; i != 0; i--)
  {
    *(uint16_t*)pbUsrBuf++ = *pdwVal++;
    pbUsrBuf++;
  }
}
```

USB와 STM32는 3군데 연결되어 있다.

- **USB_DM** - GPIO Port A 11번핀
- **USB_DP** - GPIO Port A 12번핀
- **USB_DISCONNECT** - GPIO Port A 8번핀

- USB_DISCONNECT 는 GPIO_Init()을 통해 초기화 설정을 하지만, 즉 이 GPIO를 이런 용도로 쓰겠다 말하지만
- USB_DM, USB_DP는 USB 전용으로 쓰이기에 별도의 GPIO_Init() 이 필요없다. (Alternate Function으로 사용)

외부 인터럽트 0번-15번은 각각 대응되는 GPIO 핀과 EXTI 번호가 있다.
외부 인터럽트 16, 17, 18은 특별한 용도로 쓰이는데 외부 인터럽트 18(EXTI line 18)이 바로 USB Wakeup Event에 쓰인다.
USBWakeUp_IRQn 은 42이다. WWGD(Window WatchDog Interrupt) 에서 부터의 Offset 값

USBWakeUp_IRQn 는 USBWakeUp_IRQHandler()를 부르고
다시 EXTI_ClearITPendingBit() 를 부른다.
Cortex-M3는 WFI, WFE = Wait for Interrupt, Wait for Event 라는 명령이 있는데
EXTI_ClearITPendingBit() 하는 역할은 바로 이 _WFI()를 불러놓고 기다리는 걸 해제만 해주는 것이다.

USB는 512 byte를 가지는데 위 그림의 Packet Buffer Interface 라 되어 있는 부분.
USB와 CAN이 공유하는데 동시 사용은 안된다.

Endpoint Register는 양방향 통신이 가능한 녀석으로 8개 있다.
단방향이라면 16개 가능.

메모리에서
0x4000 6000 - 0x4000 63FF : Packet buffer = PMA(Packet Memory Area)
0x4000 5C00 - 0x4000 5FFF : 레지스터 영역

그런데 PMA가 512 byte라고 했는데 위의 말대로라면 1024 byte이다.
왜냐하면 32비트에서 상위 16비트는 사용하지 않고, 하위 16비트만 사용하기 때문
APB1 bus는 32비트로 접근하고 USB는 16비트로 접근하기 때문에 이런 구조인듯함.
항상 접근할때는 사용하는 16비트만 이용하도록 구현해야 함.

레지스터는 셋으로 나뉜다.
일반 레지스터 : 인터럽트와 컨트롤
Endpoint 레지스터 : Endpoint에 대한 설정 및 상태정보
Buffer Descriptor Table : data buffer를 위치시키는데 사용되는 packet memory의 위치

레지스터 Base Address인 0x4000 5C00 에서의 OFFSET으로 레지스터를 구분하는데
Buffer descriptor table은 USB_BTABLE을 기준으로 OFFSET으로 구한다.

APB1 bridge가 word(=32비트) 단위로 접근해야기에
레지스터가 16비트만 사용하고 16비트는 비워둔다고 하였다.

따라서 USB 레지스터의 OFFSET 값에서 2배만큼 해줘야 원하는 값을 가져올 수 있다.

USB 레지스터들에 대해 상세 분석은 않겠음. ST에서 제공하는 Reference Manual 참조.

실제 레지스터 사용 용례 보기

```
void CustomHID_Reset(void)
{
 /* Set Joystick_DEVICE as not configured */
 pInformation->Current_Configuration = 0;
 pInformation->Current_Interface = 0;/*the default Interface*/

 /* Current Feature initialization */
 pInformation->Current_Feature = CustomHID_ConfigDescriptor[7];

 SetBTABLE(BTABLE_ADDRESS);

 /* Initialize Endpoint 0 */
 SetEPType(ENDP0, EP_CONTROL);
 SetEPTxStatus(ENDP0, EP_TX_STALL);
 SetEPRxAddr(ENDP0, ENDP0_RXADDR);
 SetEPTxAddr(ENDP0, ENDP0_TXADDR);
 Clear_Status_Out(ENDP0);
 SetEPRxCount(ENDP0, Device_Property.MaxPacketSize);  // MaxPacketSize는 0x40 = 64로 설정되어 있음
 SetEPRxValid(ENDP0);

 /* Initialize Endpoint 1 */
 SetEPType(ENDP1, EP_INTERRUPT);
 SetEPTxAddr(ENDP1, ENDP1_TXADDR);
 SetEPRxAddr(ENDP1, ENDP1_RXADDR);
 SetEPTxCount(ENDP1, 2);
 SetEPRxCount(ENDP1, 2);
 SetEPRxStatus(ENDP1, EP_RX_VALID);
 SetEPTxStatus(ENDP1, EP_TX_NAK);

 bDeviceState = ATTACHED;

 /* Set this device to response on default address */
 SetDeviceAddress(0);
}
```

```
// SetEPRxCount() 를 파고 들어가면 아래 함수를 만난다.
// 참고로 이 함수는 ENDP0의 RX관련 Count를 설정하고 있다.

#define _SetEPCountRxReg(dwReg,wCount)  {\
   uint16_t wNBlocks;\
   if(wCount > 62){_BlocksOf32(dwReg,wCount,wNBlocks);}\
   else {_BlocksOf2(dwReg,wCount,wNBlocks);}\
 }/* _SetEPCountRxReg */
```

```
// *pdwReg는 packet buffer 주소와 관련된 듯. 분석 중지. @_@

#define _BlocksOf32(dwReg,wCount,wNBlocks) {\
   wNBlocks = wCount >> 5;\
   if((wCount & 0x1f) == 0)\
     wNBlocks--;\
   *pdwReg = (uint32_t)((wNBlocks << 10) | 0x8000);\
 }/* _BlocksOf32 */

#define _BlocksOf2(dwReg,wCount,wNBlocks) {\
   wNBlocks = wCount >> 1;\
   if((wCount & 0x1) != 0)\
     wNBlocks++;\
   *pdwReg = (uint32_t)(wNBlocks << 10);\
 }/* _BlocksOf2 */
```

*충분히 이해는 되지 못했다.

1. USB의 메모리는 레지스터와 메모리로 나뉜다.
2. 레지스터 영역은 여러 설정과 상태를 다루는데 쓰인다.
3. 그런데 레지스터 영역 마지막 레지스터인 USB_BTABLE 은 buffer allocation table의 시작주소를 가리킨다.
   "버퍼들이 어디있는지 모아놓은 테이블의 위치는 여기입니다"

4. 아래 그림의 왼쪽이 바로 테이블이다. **Buffer Description Table.**
   - 테이블들은 PMA(Packet Memory Area)에 위치해있다.
   - Endpoint Register가 8개 있고
   - 각각의 레지스터가 데이터를 읽고 쓸 수 있는 버퍼공간이 최대 8개까지 있으며
   - 그 버퍼가 어디에(address) 얼마만큼의 크기로(Count) 있는지를 담은 정보가 이 테이블에 있다.

아래는 각각의 테이블 주소값을 확인하는 수식으로 보면 되겠다.

```
#define RegBase  (0x40005C00L)  /* USB_IP Peripheral Registers base address */
#define PMAAddr  (0x40006000L)  /* USB_IP Packet Memory Area base address   */

#define _pEPTxAddr(bEpNum) ((uint32_t *)((_GetBTABLE()+bEpNum*8  )*2 + PMAAddr))
#define _pEPTxCount(bEpNum) ((uint32_t *)((_GetBTABLE()+bEpNum*8+2)*2 + PMAAddr))
#define _pEPRxAddr(bEpNum) ((uint32_t *)((_GetBTABLE()+bEpNum*8+4)*2 + PMAAddr))
#define _pEPRxCount(bEpNum) ((uint32_t *)((_GetBTABLE()+bEpNum*8+6)*2 + PMAAddr))
```