Simple HID Keyboard device on Atmels AT90USB128 using AT90USBKey and CodeVision AVR C-Compiler

Step1: Ignore errors and confusing statements in the AT90USB128 Datasheet

Page 270: *Clearing EPNUMS*

I couldn't find any Information that this Register even exists at all

Page 253: *The SPDCONF bits can be set by software*

The same, no information that they exist and what they are use for

Page 258: *UDSS register allows to select …. UDSS should be….*

Another not existing bit…

Page 274: *CONTROL endpoints should not be managed by interrupts, but only by polling the status bits*

Why???

Page 264: *OTGPADE: OTG Pad Enable*

Set to enable the OTG pad. Clear to disable the OTG pad.

In this datasheet I couldn't even find an explanation what OTG is, I found it in the USB spec sheet and I understood it as some Device to Device transfer (like printing from some Digital still camera without PC), but without this bit set the USB controller on this chip doesn't work at all…

And many more….

Step2: Making a Project with CodeWizzardAVR

First of all I have to say that the CodeWizzard in this case is not much helpful, because in USB initialisation some events and bit sets have to follow each other…

Select the AT90USB128, Clock speed to 8MHz, Divider: 1, Type Application

Port B as input and with pull-ups , Port D as output and zero

Don't select any USB features

Make the Project

Step3: Setting up USB Controller

Make sure that Interrupts are enabled #asm("sei");

First of all set the Detach bit — `UDCON = 0x01;`

..and the Freeze clock bit — `USBCON = 0x20;`

Clear some Registers

```
OTGIEN = 0;
UDIEN  = 0;
UDINT  = 0;
UHIEN  = 0;
UEIENX = 0;
UPIENX = 0;
```

Set the Device Mode — `UHWCON = 0x80;`

Clear the Endpoints

```
for(a=0;a<8;a++){
        UENUM   = a;
        UEINTX  = 0x00;
        UECONX  = 0x00;
        UECFG1X = 0x00;
}
```

After this stop and clear procedure we can set up our device

Enable Pad — `UHWCON = 0x81;`

Make sure you take 2 several steps for Device Mode and Pad enable

Enable Makro and OTG Pad — `USBCON = 0xB0;`

After OTG Pad enable wait a bit… #asm ("nop");

It seems to work fine if you take 3 of these nops, the datasheet only says "Power-On USB pads regulator" and "Wait USB pads regulator ready state" (Page 267), but there is no information about how to know about this ready state…

Enable Clock USBCON = 0x90;

Keep Freeze Clock USBCON = 0xB0;

Enable WakeUp and End of Reset Interrupt

UDIEN = 0x18;

Attach the Device UDCON = 0;

Now our Device is Ready to be detected by the Host…and any event after this is driven by Interrupts even if the datasheet says that I shouldn't do it this way…

After this Initialisation do some endless loop…(or see Step 7)

Step4: Interrupt handling (Part 1)

The USB general interrupt… in this ISR we have to manage 2 cases, the WakeUp and the End of Reset interrupt, after attaching the device to a Host, the WakeUp INT is the first one to handle

Check if the interrupt is a WakeUp if (UDINT & 0x10){

Disable WakeUp INT UDIEN = 0x08;

Set PLL Prescaler PLLCSR = 0x0C;

Enable PLL PLLCSR = 0x0E;

Wait till PLL ready while(!(PLLCSR & 0x01));

Enable Clock USBCON = 0x90;

Delete INT Flag UDINT &= 0xEF;

This was the first part of the USB general interrupt, now our device is attached and the PLL is locked to the USB bus, the next event to come is the End of Reset

Check if its End of Reset INT }else if (UDINT & 0x08){

Now that the End of Reset INT occurred its time to set up the Control Endpoint

Select Endpoint 0 UENUM = 0;

Enable this Endpoint (EP) UECONX = 0x01;

Type: Control EP UECFG0X = 0x00;

Size: 64, 1 Bank UECFG1X = 0x30;

Allocate Memory UECFG1X = 0x32;

Some Error Condition if (!(UESTA0X & 0x80)){led2rt = 1;}

In this minimal HID Interface I don't handle any Errors, they are only shown by setting the second LED in red state, I never had this error, so I suggest to ignore this line

Delete INT Flag UDINT &= 0xF7;

Enable the Receive Setup INT UEIENX = 0x0C;

And a line of code that normally never should be called

}else{ led2rt=1;}

This was the first Part of interrupt handling…

Step5: Interrupt handling (Part 2)

The USB endpoint interrupt…again there are 2 cases, traffic on Endpoint 0 (Configuration) and traffic on Endpoint 1 (the interrupt endpoint used for transmitting Keyboard data to Host)

Check for INT on EP0 if (UEINT & 1){

Select EP0 UENUM = 0;

Check if its a setup packet if (UEINTX & 0x08){

Call a function DeviceRequest();

```
If not                                  }else{
Acknowledge it                              UEINTX &    = 0xFB;
                                        }
```

And now EP1:

```
Check for INT on EP1            }else if (UEINT & 2){
Select EP1                          UENUM = 1;
Check for Data                      if (UEINTX & 0x01){
And Ack                                 UEINTX      &= 0xF3;
It with                                 UEINTX      &= 0xBF;
A zero Byte                             UEDATX      = 0x00;
Send the zero                           UEINTX      &= 0x7F;
                                    }
```

And this time again some Error condition that should never appear in normal operation, because there are only 2 EPs activated….

```
                                }else{
                                    led2rt = 1;
                                }
```

Step6: Interrupt handling (Part 3): The function DeviceReauest()

Although everyone who reads this should know how to declare a function that with no variables, here is the declaration:

```
                    void DeviceRequest(void){
Some variables      unsigned char a, Data[8];
```

The first step is to store the 8 Bytes of Data (that make a Device Request, see Section 9.4 of USB spec) in Data

```
                    for (a=0;a<8;a++)
                        Data[a] = UEDATX;
Delete the INT Flag UEINTX      &= 0xF7;
Check if it's a Std Request if ((Data[0] & 0x60) == 0x00){
```

There a three kinds of Requests for USB devices declared by the USB spec:

- *-Standard Requests*
- *-Class Requests*
- *-Vendor Requests*

For this simple HID interface, we only need some Standard and some Class Requests

```
Check for Set Address Cmd       if (Data[1] == 5){
```

The second Standard Request that is done is the Set Address Command, at this point the device gets a unique address on USB bus, its range is from 1 to 127 (0 is used for unaddressed state, where the device is until this Command is received) This is called Enumeration…

```
Set Address                         UDADDR      = Data[2];
Ack Address                         UEINTX      &= 0xFE;
Wait for Handshake                  while(!(UEINTX & 0x01));
Activate Address                    UDADDR      |= 0x80;
Check for Get Desc Cmd          }else if(Data[1] == 6){
```

The first Std Req (on Windows, don't know on other OS) is the Get Descriptor Command, this Command can ask for several Descriptors, the ones handled by this device are:

- *the Device Descriptor*
- *the Configuration Descriptor*
- *the Device Qualifier*

- *and the HID descriptor (which is only classified as a Standard Descriptor, but is not listed in the USB spec for information on it have a look at HID spec)*

for Descriptor types see Table 9-5 of USB spec

Device Descriptor

Data to Buffer

Submit Data

Configuration Descriptor

The Configuration Descriptor is a bit tricky… there are two types of this Descriptor, the first time the Get Configuration Descriptor Command is sent, the Host (or only Windows???) requests only 9Bytes, this means only the Configuration Descriptor must be send…And the Real Configuration is meant in a Second request, the Real Configuration is the Configuration Descriptor, the Interface Descriptor, in HID case the HID Descriptor and the Endpoint Descriptors of all other Endpoints (in this case only EP1)

Check length

Only Config

Complete

Submit

Device Qualifier

Submit

HID Report Descriptor

The HID Report Descriptor is the last descriptor requested, after the request is handled, a variable is set with a device ready flag and the green LED is the signal that the device is ready to use

End of Collection Byte

Submit

Set the Device ready Flag

LED 1 green

```
if (Data[3] == 1){
        for (a=0;a<18;a++)
                UEDATX = DEV_Desc[a];
        UEINTX  &= 0xFE;
}else if (Data[3] == 2){
        if (Data[6] == 9 & Data[7] == 0){
                for (a=0;a<9;a++)
                        UEDATX =
                        CONF_Desc[a];
        }else{
                for (a=0;a<9;a++)
                        UEDATX =
                        CONF_Desc[a];
                for (a=0;a<9;a++)
                        UEDATX =
                        INTF_Desc[a];
                for (a=0;a<9;a++)
                        UEDATX =
                        HID_Desc[a];
                for (a=0;a<7;a++)
                        UEDATX =
                        EP1_Desc[a];
        }
        UEINTX &= 0xFE;
}else if (Data[3] == 6){
        for (a=0;a<10;a++)
                UEDATX =
                QUAL_Desc[a];
        UEINTX &= 0xFE;
}else if (Data[3] == 34){
        for (a=0;a<58;a++)
                UEDATX = HID_Rep[a];
        UEDATX = 0xC0;
        UEINTX &= 0xFE;
        dev_ready = 1;
        led1gr=1;
}else{
```

Again there is a handling of all other (unsupported) requests

Ack
Red LED
```
                                        UEINTX &= 0xFE;
                                        led2rt = 1;
                                    }
```
Set Configuration Request
```
                                }else if(Data[1] == 9){
```
At the Set Configuration Request, the device activates additional Endpoints (in tis case only EP1)

Ack
EP1 select
EP enable
Type INT, in
Size: 8, 1 Bank
Allocate Memory
Enable Interrupts
```
                                        UEINTX &= 0xFE;
                                        UENUM   = 1;
                                        UECONX  = 0x01;
                                        UECFG0X = 0xC1;
                                        UECFG1X = 0x00;
                                        UECFG1X = 0x02;
                                        UEIENX  = 0x0C;
                                    }else{
```
And again some Error Condition on all other events
```
                                        led2rt = 1;
                                        UEINTX &= 0xFE;
                                    }
```
Vendor Request
```
                                }else if ((Data[0] & 0x60) == 0x40){
```
Some Error Condition on Vendor Request
```
                                        UEINTX &= 0xFE;
                                        led2rt = 1;
```
Class Request
```
                                }else{
```
The "handling" or should I say the not handling of some Class Requests…

Set Idle Mode
```
                                        if(Data[1] == 10){
                                                UEINTX &= 0xFE;
```
Set Report
```
                                        }else if(Data[1] == 9){
                                                UEINTX &= 0xFE;
                                        }else{
                                                led2rt = 1;
                                        }
                                }
```
And finally the end of this function:
```
                        }
```

Step7: The main thing of this Exercise

Now that the device is properly set up, we can send some keys
```
                        if (PINB.5 == 0){
                                led1rt = 1;
                                sendstring("ABCDEFGHIJKLMNOPQRSTUVWXYZ
                                        012345789-abcdefghijklmnopqrstuvwxyz");
```
sendstring() is a function that converts Ascii into Keycodes
```
                                while(PINB.5 == 0){;}
                        } else {
                                led1rt = 0;
                        }
```

Step8: HID Keycodes

Let's convert some Ascii Characters to HID Keycodes
```
                        void sendkey(unsigned char key) {
```

```
                    unsigned char code, mod, y;
Small letters               if ((key >= 97) & (key <= 122)){
                                mod = 0;
                                code = key - 93;
Big letters                 }else if ((key >= 65) & (key <= 90)){
                                mod = 2;
                                code = key-61;
Numbers 1-9                 }else if ((key >= 49) & (key <= 57)){
                                mod = 0;
                                code = key-19;
Zero                        }else if (key == 48){
                                mod = 0;
                                code = 39;
Minus                       }else if (key == 45){
                                mod = 0;
                                code = 86;
Everything else             }else{
                                mod = 0;
                                key = 0;
                            }
```

The Submission of every single Key to the Host, this id done in two steps, first the Key press is submitted and after a delay (By the way this delay is part of the delay.h which you have to include) the Key release is transmitted by sending zeros

```
EP1                         UENUM = 1;
                            UEDATX = mod;
                            UEDATX = 0;
                            UEDATX = code;
                            UEDATX = 0;
                            UEDATX = 0;
                            UEDATX = 0;
                            UEDATX = 0;
                            UEDATX = 0;
                            UEINTX &= 0x7B;
                            delay_ms(8);
                            UENUM = 1;
                            for(y=0;y<8;y++)
                                UEDATX = 0;
                            UEINTX &= 0x7B;
```

And end of function:

```
                    }
```

A second function makes it possible to transmit whole strings (include string.h for strlen)

```
                    void sendstring(unsigned char flash *str){
                    unsigned int lang, y;
                        lang = strlenf(str);
                        for (y=0;y<lang;y++){
                            sendkey(str[y]);
                            delay_ms(8);
                        }
                    }
```

Now you're done

Fore more information refer to
Atmel AT90USB128 datasheet
USB whitepaper
HID whitepaper
HID Usage Descriptor whitepaper

If you should find some Errors or have any suggestions feel free to contact me at
mail@michael-gerber.net

Check the latest Version of this document at www.michael-gerber.net