

OVERVIEW

The AnchorChips EZ-USB chips support a feature known as AutoVectoring. This feature lets multiple USB interrupt sources share a single interrupt vector. The examples provided by AnchorChips show you how to implement your programs to use this feature in assembly. The following Application Note shows you how to write C code that supports AutoVectoring.

AUTOVECTOR REQUIREMENTS

Normally, when an interrupt on the 8051 occurs, the program counter is changed and execution begins at the interrupt vector (which usually contains an LJMP to the interrupt service routine). AutoVectoring on the EZ-USB chips works nearly the same way. However, the byte at offset two is replaced by 0, 4, 8, 12, 16, 20, and so on depending on the USB interrupt source. Then, when the interrupt occurs and the program counter changes, program execution LJMPs into another table that LJMPs to the final program code. More information about AutoVectoring may be obtained in the AnchorChips documentation.

To support AutoVectoring in your applications, you must:

1. Include a small assembly file,
2. Carefully declare the AutoVector interrupt service routines,
3. Enable AutoVectoring.

For the examples presented here, we assume that the AutoVector table is stored starting at address 4000h.

ASSEMBLY CODE

The following assembly code must be included in your application. Its purpose is to set an LJMP into the AutoVector table at address 4000h.

```
cseg    at    43h
        ljmp  4000h
end
```

AUTOVECTOR INTERRUPT SERVICE ROUTINES

The AutoVector ISRs may be written entirely in C if you observe a few rules.

1. Set the Interrupt Vector Interval to 4 bytes,
2. Set the Interrupt Vector to start at 4000h (for this example).

The following C code may be included in a file or files separate from other interrupt code. It must be separated because the interrupt vector addresses and intervals are setup differently.

```
#pragma intvector (0x3FFD) /* Start Interrupts at 4000h */
#pragma interval (4) /* Interrupt Vectors: 4000, 4004, 4008, ... */

static void SUDAV_ISR (void) interrupt 0
{
}

static void SOF_ISR (void) interrupt 1
{
}

static void SUTOK_ISR (void) interrupt 2
{
}

static void SUSP_ISR (void) interrupt 3
{
}

static void URES_ISR (void) interrupt 4
{
}

static void SPARE_ISR (void) interrupt 5
{
}

static void EP0IN_ISR (void) interrupt 6
{
}

static void EP0OUT_ISR (void) interrupt 7
{
}

static void EP1IN_ISR (void) interrupt 8
{
}

static void EP1OUT_ISR (void) interrupt 9
{
}

static void EP2IN_ISR (void) interrupt 10
{
}

static void EP2OUT_ISR (void) interrupt 11
{
}
```

```
static void EP3IN_ISR (void) interrupt 12
{
}

static void EP3OUT_ISR (void) interrupt 13
{
}

static void EP4IN_ISR (void) interrupt 14
{
}

static void EP4OUT_ISR (void) interrupt 15
{
}

static void EP5IN_ISR (void) interrupt 16
{
}

static void EP5OUT_ISR (void) interrupt 17
{
}

static void EP6IN_ISR (void) interrupt 18
{
}

static void EP6OUT_ISR (void) interrupt 19
{
}

static void EP7IN_ISR (void) interrupt 20
{
}

static void EP7OUT_ISR (void) interrupt 21
{
}
```

CONCLUSION

All you have to do is to fill-in the interrupt function stubs with code for each interrupt, enable the interrupt, and enable AutoVectoring. You may want to leave all USB interrupts in a single source file or split them out into multiple files. If you do split them out, remember to include the interval and intvector pragmas at the beginning of each source file.

Copyright © 1998 Keil Software, Inc. All rights reserved.

In the USA:
Keil Software, Inc.
16990 Dallas Parkway, Suite 120
Dallas, TX 75248-1903
USA

Sales: 800-348-8051
Phone: 972-735-8052
FAX: 972-735-8055

E-mail: sales.us@keil.com
support.us@keil.com

Internet: <http://www.keil.com/>

In Europe:
Keil Elektronik GmbH
Bretonischer Ring 15
D-85630 Grasbrunn b. Munchen
Germany

Phone: (49) (089) 45 60 40 - 0
FAX: (49) (089) 46 81 62

E-mail: sales.intl@keil.com
support.intl@keil.com