Using UART0 on AN2131-DK001 “Cypress USB Full-Speed Development Kit”

Introduction

This paper discusses how to transfer data over Endpoint 2 then out EZ-USB UART0 (labeled SIO-0 on DK) at 9600,8,N,1 using EZ-USB Control Panel. This implementation uses SCON.1 (TI) bit polling.

We’ll start on the basis that you have already received your development kit and installed the software that came on the CDROM. Hopefully you didn’t miss out on the tutorial under EZ-USB Control Panel **Help | Contents and Tutorial** menu.

Default EZ-USB UART Configuration

EZ-USB implements two UARTS. By default the Keil dScope monitor debugger uses UART1 (and Timer 1), leaving UART0 for your application.

When you plug your development board into the USB, Windows loads the default *ezmon.sys* file, based on the default *VID(0x547) & PID(0x80)* values, which then downloads *mon-ext-sio1.hex* to the DK at program memory location *0xE000* and above. Once this download occurs *ezmon.sys* unloads and the greeen “Monitor” LED comes ON.

The EZ-USB Control Panel program has the ability to load the Keil monitor for use with EZ-USB UART0 by selecting the “LoadMon” button. This selection is determined under **Options | Properties | Path** dialog box. By default the Keil monitor runs at 19200 baud rate.

Keil dScope Configuration

After launching dScope, select “*mon51.dll*” from the drop down menu. At this point you may be presented with a **“NO TARGET SYSTEM FOUND!”** dialog box prompting you to enter the proper configuration for your PC. Match the proper port and baudrate then press **“Try Again”**. If you are having problems connecting to the Keil monitor then see the following check list,

`www.keil.com/support/docs/214.htm`

As a final configuration step, check the “**Use serial interrupt**” box under the **Peripherals | Configuration** dialog box from within dScope.

If the default setup is working, then you are ready to download an example to verify dScope operation. Choose **File** | **Load Object File** | search for *dev_io.hex* under the examples directory, then select **GO** under the debug window. The seven segment LED should display “0” and buttons *F3* and *F2* should count up/down, respectively.

EZ-USB UART0 Setup

To start, let’s look at what is needed for simple terminal-based serial communications (Tx, Rx, and GND).

EZ-USB UART0 uses PCO/RxD0 and PC1/TxD0 pins. EZ-USB UART1 uses PB2/RxD1 and PB3/TxD1 pins. On power-on reset, these pins default to IOs and not their alternate function. So, the first step is to set PORTnCFG=1 so that these pins, EZ-USB UART0 only, assume their alternate function.

This is a good time to point out Table B-5 “Special Function Registers” of *EZ-USB TRM v1.8*. This table lists all the SFRs in EZ-USB. The *Dallas High Speed Microcontroller User’s Guide* is a good source for referencing the specific register bits. We don’t need to utilize all of these registers for simple RS-232 communications. Below is a list of the specific registers we need to manipulate for this implementation:

- SCON0 - UART0 Port Control Register
- SBUF0 - Serial Data Register
- T2CON -Timer 2 Control Register
- RCAP2L - Timer 2 Reload Register
- RCAP2H - Timer 2 Reload Register

This specific EZ-USB UART0 implementation uses the following resources:

- UART0,
- Timer 2 in baud rate generator mode
- PC0/RxD0 and PC1/TxD0 port pins

For now we’ll leave the monitor debugger port pins, EZ-USB UART1, defaulted.

```
PORTCCFG |= 0x03; // UART0, using TXD0 & RXD0
```

Typically, this “C” code is added to the function *TD_Init()* in your frameworks-based EZ-USB firmware.

Since, *mon-ext-sio1.hex* uses Timer 1 then we’ll implement EZ-USB UART0 baud rates using Timer 2. In addition, Timer 2 has a 16-bit preload for more accurate baud rate generator values.

The following formula is used to determine EZ-USB Timer 2 reload values for a given baudrate:

```
RCAP2H, RCAP2L = 65536 - ( 24MHz / ( 32 * BaudRate ) )
```

EZ-USB UART0 register initialization “C” code in *TD_Init()*:

```
RCAP2H = TH2 = 0xFF; // auto - reload, 9600
RCAP2L = TL2 = 0xB2; // auto - reload, 9600
T2CON = 0x34; // overflow clk, start timer
SCON0 |= 0x52; // Mode 1, Rx enabled, set TI
```

Polling EZ-USB UART0 transmit bit “C” code in *TD_Poll()*:

```
while (!TI); // check if UART0 Tx is ready
TI=0; // clear UART0 Tx flag
SBUF0 = Tx0_Data; // Tx the data
```

At this point, we know what registers to manipulate and the code necessary to transmit data over EZ-USB UART0.

Let's see what is needed to add RS-232 functionality to EZ-USB frameworks based application so that we can transmit bytes over Endpoint 2 to the target and out the EZ-USB UART0 TxD0 pin using the EZ-USB Control Panel program.This final firmware example will look similar to the EP_PAIR example. In fact, we can just edit this as UART0 in the exam-ples directory. The main difference being that we aren't looping back Endpoint 2 out data to Endpoint 2 in. Instead our goalis to transmit this data out EZ-USB UART0 TxD0 pin at 9600,8,N,1.We need to add our application specific firmware to *TD_Init()* and *TD_Poll()* routines, as typically done.

Listing 1 contains the rewritten *TD_Init() & TD_Poll()* routines for our EZ-USB UART0 example:

Listing 1

```
 1  void TD_Init(void)
 2  { // Called once at startup
 3    // uart0 application specific SIO-0 register initialization
 4    PORTCCFG |= 0x03;       // using UART0, TxD0 & RxD0 pins
 5    RCAP2H = TH2 = 0xFF;    // auto-reload for 9600 baud
 6    RCAP2L = TL2 = 0xB2;    // auto-reload for 9600 baud
 7    T2CON = 0x34;           // clk overflow, start timer2 ticks
 8    SCON0 |= 0x52;          // mode1, rx enable, set TI0
 9  // terminal settings: 9600,8,N,1, no hardware flow control
10  // uart0 application specific USB register initialization
11    IN07VAL = bmEP2;        // using endpoint 2 in
12    OUT07VAL = bmEP2;       // using endpoint 2 out
13
14  // arm endpoint 2 out to receive data
15    EPIO[OUT2BUF_ID].bytes = 0x00;
16
17  // setup breakpoint to trigger each time TD_Poll() gets called
18    BPADDR = (WORD)TD_Poll;
19    USBBAV |= bmBPEN;       // enable breakpoint
20    USBBAV &= ~bmBPPULSE;
21
22    Rwuen = TRUE;            // Enable remote-wakeup
23  }

24  void TD_Poll(void)
25  { // Called repeatedly while the device is idle
26    BYTE i = 0x00;
27
28      if( !( EPIO[ OUT2BUF_ID ].cntrl & bmEPBUSY ) )
29      { // there's data in OUT2BUF
30              for( i = 0x00; i < EPIO[ OUT2BUF_ID ].bytes; ++i )
31              {
32                      while( !TI );              // poll tx flag
33                      TI = 0;                    // clr it
34                      SBUF0 = OUT2BUF[ i ];      // tx byte using uart0
35              }
36              EPIO[ OUT2BUF_ID ].bytes = 0x00;
37      }
38    // toggle green breakpoint/monitor LED for visual
39    EZUSB_Delay( 100 );  // to see the green LED ON
40    USBBAV |= bmBREAK;
41    EZUSB_Delay( 100 );  // to see the green LED OFF
42  }
```

We can use the same *dscr.a51* file as EP_PAIR for this application.

Getting Mon51 and EZ-USB UART0 working

Now that we have completed our code modifications how do we get this firmware working with Mon51?

Launch dScope and load *mon51.dll* then **File** | **Load Object File** | select *uart0*, then click **GO** from the debug window. Notice that the green monitor LED is flashing. Launch *Hyperterminal* with the proper settings for EZ-USB UART0. Launch the EZ-USB Control Panel program and click on the following buttons: **GetDev**, **GetPipes**, **GetString** to confirm that *uart0.hex* is running on the target. Select **Bulktrans Pipe 1:Endpoint 2 OUT** then load **HexBytes** with *45* and **Length** with *64* and click the **BulkTrans** button. Switch back over to *Hyperterminal* and 64 “E”s should be displayed.

Conclusion

EZ-USB has two UARTs available to your design / development. Typically, one is used for dScope monitor debugging and the other is free for use by your application.