Bulk Performance Analysis of the Cypress AN2131

Introduction

The Cypress EZ-USB series of USB chips incorporates features to speed the transfer of data to and from the Universal Serial Bus. These include:

1. A fast 8051 core (4 clocks per instruction cycle, 24-MHz clock).
2. An auto-incrementing data pointer that gives FIFO-like access to RAM.
3. A fast transfer mode that transfers outside data directly in and out of endpoint FIFOS.
4. Endpoint pairing, which allows double-buffered bulk transfers.

This note presents 8051 code that achieves fast transfer of data from a USB peripheral device to a USB host over a bulk endpoint. This 8051 code continuously transfers data from outside the AN2131 to the endpoint 2 IN buffer using the fast transfer mode.

The **Anchor** Control Panel (supplied with the EZ-USB Developer Kit) was used to initiate long bulk IN transfers, and the bus traffic was recorded and analyzed using a CATC USB bus analyzer. From this analysis it was possible to study USB bus utilization, and measure the maximum number of bytes per frame that the AN2131 can transfer over a bulk endpoint.

USB Data Transfer

The EZ-USB architecture transfers data from *internal or external* sources using the same instruction, namely 'movx'. The AN2131 automatically generates a read strobe (Fast Read, FRD#) for every 'movx' instruction when the fast transfer mode is enabled. The 'movx' instruction requires two 8051 cycles, or eight 24-MHz clocks. Thus a byte can be transferred into or out of an endpoint buffer in 333 nanoseconds.

The AutoPointer

The AN2131 provides bulk endpoint data in 64-byte RAM buffers. Normally, a transfer into one of these buffers would require a 'movx @ dptr,a' instruction, where the data pointer 'dptr' contains the memory address of the bulk endpoint buffer. For every 'movx' transfer, an 'inc dptr' instruction would also be required to advance the address pointer to the next byte in the endpoint buffer. To eliminate this overhead, the AN2131 provides a special hardware pointer called the Autopointer. The 8051 loads a 16-bit address into two registers called AUTOPTRH and AUTOPTRL, and then reads or writes a single register called AUTODATA to retrieve data pointed to by AUTOPTR/H-L. Each read or write of the AUTODATA register automatically increments the 16-bit address in AUTOPTR/H-L. Thus the random-access data in a bulk endpoint buffer can be addressed sequentially through a single address (like a FIFO), saving the 'inc dptr' instruction for every byte transferred.

Double Buffering (Endpoint Pairing)

When the USB host acknowledges an IN transfer for a particular endpoint, the AN2131 clears the endpoint's BUSY bit. The 8051 is free to load the endpoint buffer only when the BUSY bit is LOW. The 8051 takes a finite amount of time to transfer data into the endpoint buffer, during which the USB host may request another IN transfer. If the 8051 has not finished loading data into the endpoint buffer when the next IN token arrives, the AN2131 sends a NAK handshake in response to the IN token, indicating that the host should issue an IN token at a later time. NAK means "I'm busy, try again."

For best transfer speed, the 8051 should load the *next* packet of bulk data while the *previous* packet is being transferred over USB. This is accomplished by double buffering, or in AN2131 terms, "endpoint pairing." By setting a control bit, the even-numbered bulk endpoints can be paired with the next sequential endpoint to implement double buffering. In the code example, endpoint 2 IN is paired with endpoint 3 IN. Note that once a pairing bit is set, the endpoint should be accessed only through the even endpoint—in this example, endpoint 2 IN. Endpoint 3 should not be accessed in any manner (the endpoint 3 buffer is used by the AN2131 hardware for the 'other' buffer).

Results

The AN2131 transferred seventeen 64-byte packets of bulk data per frame. This translates to 8.704 Mbits/sec if the bulk data were to be transferred on a sustained basis, and if no other USB devices were on the bus. This accounts for a bus utilization percentage of 73%

Using the programs described in this note, the 8051 core transferred data from the outside world into the endpoint 2 IN buffer quickly enough to insure that no NAK handshakes were generated by the AN2131. **Thus the achieved bandwidth was not determined by the AN2131, but rather by the USB host.** *Table 1* shows the results of the CATC USB Bus Analyzer capture for one frame of USB data. The **Anchor** Control Panel was set to transfer 4096 bytes of bulk data over endpoint 2 IN, and the CATC was set to trigger on any IN transfer. Then a "full" frame (in the middle of the transfer) was captured for analysis.

Table 1 shows the bus idle times for each of the packets in the frame. The frame starts with an SOF, followed by 33 idle bit times. Then each of the seventeen IN transfers consists of an IN token, a DATA token, 64 bytes of data, and finally an ACK token. Notice that the total idle time of 1822 USB clocks accounts for 15% of the bus overhead, and the token overhead accounts for 12% of the bus overhead.

Conclusion

This data was taken to confirm that the AN2131 is speedy enough to get outside world data into a bulk endpoint buffer without incurring 'wait' states by NAK'ing any of the host's IN transfers. The transfer "bandwidth" of 8.7 Megabits per sec-

Table 1. Bit Times for One Frame of USB Data, Transferring Data from Endpoint 2 IN from the AN2131

	IDLE Times (in USB times)																	Totals	
SOF	33																	33	
Packet #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17		
IN	6	6	6	6	6	6	6	6	6	8	8	8	6	6	6	6	6	108	
DATA0/1	7	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	135	
ACK	62	165	62	62	62	63	62	62	62	62	62	62	62	62	62	64	448	1546	
														Total IDLE time				1822	
Total Frame Time																		11984	
Payload time = 17 packets * 64 bytes * 8 bits																		8704	73%
Total Idle Time																		1822	15%
Overhead (tokens) time																		1458	12%

ond should not be interpreted as a typical maximum bandwidth for bulk transfers. In practice there will be other USB devices plugged into the bus. Other bus traffic will subtract from the available bulk bandwidth.

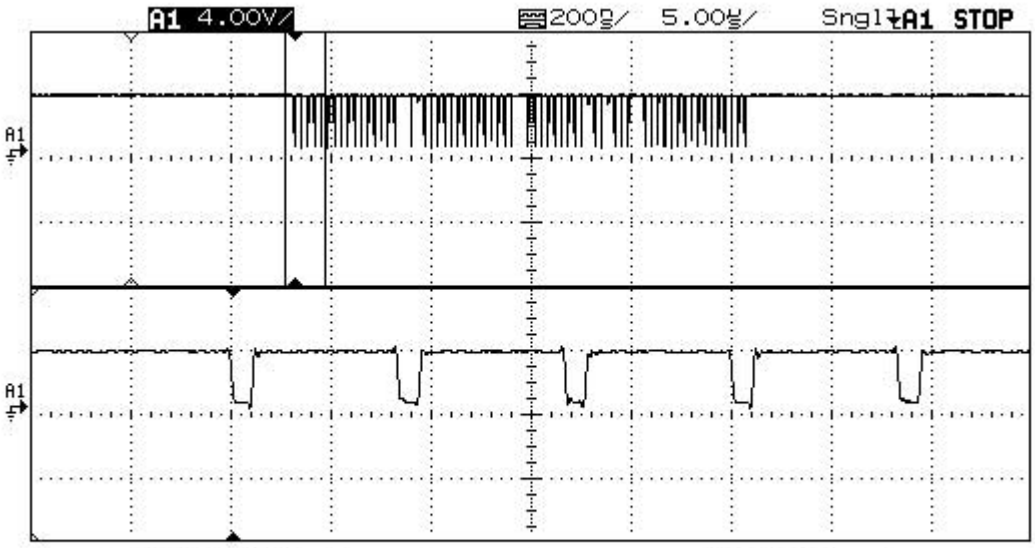
These tests were run using two Dell computers, a 200-MHz Dimension XPS Pro with a Symbios OHCI controller, and a 300-MHz Dimension XPS with a UHCI controller. Results were identical—the PC managed to issue seventeen IN tokens per USB frame, and the AN2131 fed the IN endpoints quickly enough not to incur any NAKs.

The AN2131 achieves its "no-NAK" performance of 8.7 Megabits per second regardless of where the 8051 program code is located (internal or external memory), and regardless of the data source, internal or external to the AN2131.

Code Details

Two 8051 code examples are shown below. The first example polls the EP2IN busy bit and transfers 64 bytes into the IN2BUF buffer whenever the busy bit goes LOW (endpoint not busy). The 64-byte transfer is accomplished by in-line coding a loop of sixteen 'movx' instructions, and executing the loop four times. The scope photo below shows the FRD# (Fast Read) strobe timing for both code examples. Notice that the code looping after every sixteen byte transfers introduces a small gap in the FRD# strobe timing.

The second example places the transfer code into an endpoint 2 ISR (Interrupt Service Routine). This was done to add the overhead of responding to an interrupt, to insure that the 8051 still has plenty of time to load data into the endpoint 2 buffer. The test results were identical, 17 packets per frame with no NAKs.



Top: An2131Q FRD# pulses, showing four groups of 16 byte transfers.
Bot: Expanded scale of Top, showing FRD# pulses every 333 nanoseconds.

One programming subtlety is worth mentioning. When endpoint pairing is used, the 8051 achieves fastest operation by loading the endpoint 2 IN buffer and arming the EP2-IN transfers (by loading IN2BC) twice before entering the background program. EP2-IN interrupts occur whenever an IN packet is received and acknowledged by the host. By pre-loading two transfers, the ISR does not need to check the endpoint's busy bit to determine whether to load one or two packets.

For reference, when endpoint pairing is in effect, the busy bit acts as follows:

- Initially the busy bit is 0.
- After the first IN2BC load the busy bit is still 0 (one buffer filled).
- After the second IN2BC load the busy bit goes to 1 (both buffers filled).
- After an IN transfer successfully completes it goes to 0. The 1-0 transition causes an EP2 interrupt request.
- If another IN transfer occurs before the 8051 loads another buffer full of data, a second interrupt request occurs even though the busy bit remains at 0.
- After the third IN2BC load (in the ISR) it goes to 1 (both buffers filled).

In the second code example a full USB ISR jump table is provided at the end of the code, but only the 'IN2_ISR' label is used. This is a convenient template for any 8051 code that uses AN2131 autovectoring. As interrupt service routines are written, the corresponding labels in the jump table are commented out.

8051 Code—Polled Example

```
;------------------------------------------------------------
; bulkfxf.A51  8-21-98   LTH
;
; Program for App Note demonstrating bulk rate performance of AN2131.
;
; Send data over EP2IN as fast as possible, using the fast transfer mode
; and paired endpoint EP2-3IN.  This demonstrates how many bulk packets
; the EZ-USB chip can send in a frame.
;
;------------------------------------------------------------
$NOMOD51                            ; disable predefined 8051 registers
;$nolist
$INCLUDE (..\REG320.INC)
$include (..\ezregs.inc)            ; ez-USB register assignments
;$list
;
NAME           bulkfxfr
;
               ISEG AT 80H          ; stack
stack:                 ds     20
;
               CSEG   AT     0      ; absolute Segment at Address 0
               LJMP   start         ; Jump over the interrupt vectors
; ------------------------------------------------------------------

               org    200H
start:         mov    SP,#STACK-1   ; set stack
;
; Set the stretch value to 0. Don't disturb bits 7-3.
;
               mov    a,CKCON
               anl    a,#11111000b  ; fastest MOVX cycles--b2b1b0=000
               mov    CKCON,a
;
; Select the alternate function for PA5--FRD "Fast Read" Strobes
;
               mov    dptr,#PORTACFG
               movx   a,@dptr
               orl    a,#00100000b  ; set bit 5 for PA5 alt fn
               movx   @dptr,a
;
; Enable fast BULK transfers
;
               mov    dptr,#FASTXFR
               mov    a,#01000000b  ; b7   0      no fast ISO
                                    ; b6   1      enable fast BULK
                                    ; b5   0      FRD# active low
                                    ; b43  00     1 clk FRD# strobe
                                    ; b2   0      FWR# active low
               movx   @dptr,a
;
; Select endpoint pairing for EP2IN
;
               mov    dptr,#USBPAIR
               mov    a,#1          ; EP2IN paired
               movx   @dptr,a
;
; The AUTOPTR high byte needs to be loaded only once, since when we transfer
; 64 bytes only the low autopointer byte increments.
;
               mov    dptr,#AUTOPTRH
               mov    a,#HIGH(IN2BUF)
```

```
                movx    @dptr,a
;
                mov     dptr,#IN2BC     ; arm the first transfer
                mov     a,#64
                movx    @dptr,a
;---------------------------------------
loop:           mov     dptr,#IN2CS     ; check EP2IN busy bit
loo2:           movx    a,@dptr
                jb      acc.1,loo2      ; endpoint is busy
;
service_IN2:    mov     dptr,#AUTOPTRL ; initialize the autopointer to point to IN2BUF
                mov     a,#LOW(IN2BUF)
                movx    @dptr,a
                mov     dptr,#AUTODATA ; This register will now access IN2BUF as a FIFO
                mov     r7,#4           ; (4) 16-byte loops
;
; In-line code to transfer 64 bytes using four 16 byte chunks into EP2IN buffer
;
inloop16:       movx    @dptr,a         ; generate FRD# strobe and read from D[7..0]
                movx    @dptr,a
                movx    @dptr,a
                movx    @dptr,a
                movx    @dptr,a
                movx    @dptr,a
                movx    @dptr,a
                movx    @dptr,a
                movx    @dptr,a
                movx    @dptr,a
                movx    @dptr,a
                movx    @dptr,a
                movx    @dptr,a
                movx    @dptr,a
                movx    @dptr,a
                movx    @dptr,a
                movx    @dptr,a
                djnz    r7,inloop16
;
                mov     dptr,#IN2BC
                mov     a,#64
                movx    @dptr,a         ; arm the EP2IN transfer
;
                sjmp    loop            ; keep going
;
                END
```

8051 Code—Interrupt Driven Example

```
;------------------------------------------------------------
; ibulkfxf.A51  8-25-98   LTH
; (Interrupt driven version of bulkfxf.a51)
;
; Program for App Note demonstrating bulk rate performance of AN2131.
;
; Send data over EP2IN as fast as possible, using the fast transfer mode
; and paired endpoint EP2-3IN.  This demonstrates how many bulk packets
; the EZ-USB chip can send in a frame.
;
; ---------------------------------------------------------------------
$NOMOD51                            ; disable predefined 8051 registers
$nolist
$INCLUDE (..\REG320.INC)
$include (..\ezregs.inc)            ; ez-USB register assignments
$list
;
NAME           bulkfxfr
;
               ISEG AT 80H          ; stack
stack:         ds     20
;
               CSEG   AT     0      ; absolute Segment at Address 0
               LJMP   start         ; Jump over the interrupt vectors
; -----------------
; Interrupt Vectors
; -----------------
               org    43h           ; int2
               ljmp   USB_Jump_Table ; Autovector will replace byte 45
; ---------------------------------------------------------------------
               org    200H
start:         mov    SP,#STACK-1   ; set stack
;
; Set the stretch value to 0. Don't disturb bits 7-3.
;
               mov    a,CKCON
               anl    a,#11111000b  ; fastest MOVX cycles--b2b1b0=000
               mov    CKCON,a
;
; Select the alternate function for PA5--FRD "Fast Read" Strobes
;
               mov    dptr,#PORTACFG
               movx   a,@dptr
               orl    a,#00100000b  ; set bit 5 for PA5 alt fn
               movx   @dptr,a
;
; Enable fast BULK transfers
;
               mov    dptr,#FASTXFR
               mov    a,#01000000b  ; b7   0      no fast ISO
                                    ; b6   1      enable fast BULK
                                    ; b5   0      FRD# active low
                                    ; b43  00     1 clk FRD# strobe
                                    ; b2   0      FWR# active low
               movx   @dptr,a
;
; Select endpoint pairing for EP2IN
;
               mov    dptr,#USBPAIR
               mov    a,#1          ; EP2IN paired
               movx   @dptr,a
;
```

```
; The AUTOPTR high byte needs to be loaded only once, since when we transfer
; 64 bytes only the low autopointer byte increments.
;
               mov     dptr,#AUTOPTRH
               mov     a,#HIGH(IN2BUF)
               movx    @dptr,a
;
               mov     dptr,#IN2BC     ; arm the first transfer
               mov     a,#64
               movx    @dptr,a
               movx    @dptr,a         ; arm the second transfer
;
               mov     dptr,#USBBAV    ; enable autovectors
               mov     a,#00000001b    ; AVEN bit
               movx    @dptr,a                ; do it
;
               mov     dptr,#IN07IEN   ; enable EP2IN interrupt
               mov     a,#00000100b
               movx    @dptr,a

               setb    EX2             ; enable USB interrupts (INT2)
               setb    EA              ; enable 8051 interrupts
;---------------------------------------------------------------
loop:           sjmp   loop            ; wait for the interrupt
;---------------------------------------------------------------
; Endpoint 2 ISR.  Save registers, clear interrupt request flags,
; then transfer 64 bytes from the outside data bus into EP2IN buffer.
;
IN2_ISR:       push    dps             ; save important regs
               push    dpl
               push    dph
               push    dpl1
               push    dph1
               push    acc
;
               mov     a,EXIF          ; clear the IRQ2(USB) interrupt
               clr     acc.4
               mov     EXIF,a
;
               mov     dptr,#IN07IRQ
               mov     a,#00000100b
               movx    @dptr,a         ; clear EP2IN IRQ
;
               mov     dptr,#AUTOPTRL ; initialize the autopointer to point to IN2BUF
               mov     a,#LOW(IN2BUF)
               movx    @dptr,a
               mov     dptr,#AUTODATA ; This register will now access IN2BUF as a FIFO
               mov     r7,#4          ; (4) 16-byte loops
;
; In-line code to transfer 64 bytes using four 16 byte chunks into EP2IN buffer
;
inloop16:      movx    @dptr,a         ; generate FRD# strobe and read from D[7..0]
               movx    @dptr,a
               movx    @dptr,a
               movx    @dptr,a
               movx    @dptr,a
               movx    @dptr,a
               movx    @dptr,a
               movx    @dptr,a
               movx    @dptr,a
               movx    @dptr,a
               movx    @dptr,a
               movx    @dptr,a
               movx    @dptr,a
               movx    @dptr,a
               movx    @dptr,a
               movx    @dptr,a
```

```
                movx    @dptr,a
                movx    @dptr,a
                movx    @dptr,a
                djnz    r7,inloop16
;
                mov     dptr,#IN2BC
                mov     a,#64
                movx    @dptr,a         ; arm the next EP2IN transfer
;
                pop     acc             ; restore registers
                pop     dph1
                pop     dpl1
                pop     dph
                pop     dpl
                pop     dps
                reti
;-------------------------------------------------------------------------------
                CSEG    AT 400H; autovector jump table
USB_Jump_Table:
                ljmp    SUDAV_ISR       ; Setup Data Avaialble
                db      0               ; make a 4-byte entry
                ljmp    SOF_ISR         ; SOF
                db      0
                ljmp    SUTOK_ISR       ; Setup Data Loading
                db      0
                ljmp    SUSP_ISR        ; Global Suspend
                db      0
                ljmp    URES_ISR        ; USB Reset
                db      0
                ljmp    SPARE_ISR
                db      0
                ljmp    IN0_ISR
                db      0
                ljmp    OUT0_ISR
                db      0
                ljmp    IN1_ISR
                db      0
                ljmp    OUT1_ISR
                db      0
                ljmp    IN2_ISR
                db      0
                ljmp    OUT2_ISR
                db      0
                ljmp    IN3_ISR
                db      0
                ljmp    OUT3_ISR
                db      0
                ljmp    IN4_ISR
                db      0
                ljmp    OUT4_ISR
                db      0
                ljmp    IN5_ISR
                db      0
                ljmp    OUT5_ISR
                db      0
                ljmp    IN6_ISR
                db      0
                ljmp    OUT6_ISR
                db      0
                ljmp    IN7_ISR
                db      0
                ljmp    OUT7_ISR
                db      0
;
```

```
SUDAV_ISR:      reti
SOF_ISR:        reti
SUTOK_ISR:      reti
SUSP_ISR:       reti
URES_ISR:       reti
SPARE_ISR:      reti
IN0_ISR:        reti
OUT0_ISR:       reti
IN1_ISR:        reti
OUT1_ISR:       reti
;IN2_ISR:       reti
OUT2_ISR:       reti
IN3_ISR:        reti
OUT3_ISR:       reti
IN4_ISR:        reti
OUT4_ISR:       reti
IN5_ISR:        reti
OUT5_ISR:       reti
IN6_ISR:        reti
OUT6_ISR:       reti
IN7_ISR:        reti
OUT7_ISR:       reti;-------------------------------------------------------------
                END
```