Application Note 604
Fast Memory Transfers with the Ultra High-Speed Microcontroller

www.maxim-ic.com

Note to Reader: \ denotes an active-low signal.

OVERVIEW
Embedded system designers and programmers are always searching for clever ways to push performance boundaries of their 8051 microcontroller designs. If your application is performance-limited by the rate at which the microcontroller can copy external data-memory buffers or read/write memory-mapped peripherals, you might benefit from a scheme that uses more on-chip hardware and less software. This application note introduces the use of a timer/counter as a means to terminate a fixed-length copy routine. Using this method with Dallas' Ultra High-Speed 8051 architecture and its data pointer enhancements, data can be copied at rates in excess of 2MBps, over 20 times faster than a standard 8051 core running at the same clock frequency and equipped with a single data pointer.

XDATA COPY ROUTINE–ORIGINAL 8051
The original 8051 architecture contains only one data pointer (DPTR = DPH + DPL), yet provides two different ways to indirectly access external data memory. The MOVX instruction can either use the 16-bit data pointer (e.g., MOVX @DPTR, A) or an 8-bit working register (e.g., MOVX @R0, A) to access MOVX data space. Note that the latter instruction uses an 8-bit register pointer, thus requiring that Port 2 be written with most significant address byte of the pointer prior to instruction execution. With only one data pointer, managing source and destination pointers when copying data in excess of 256 bytes requires heavy use of the working registers for temporary storage. Below is an example of code typically executed when copying data using a single data pointer.

```
; Original 8051 Copy – Single DPTR
;     R6:R7 control copy length
;     R4:R0 used for source/dest DPTR hi/lo temp storage
loop:                   ; Cycle Count  @ 12clks/cycle
      movx  a,@dptr     ; 2
      inc   dptr        ; 2
      xch   a,r0        ; 1
      xch   a,dpl       ; 1
      xch   a,r0        ; 1
      xch   a,r4        ; 1
      xch   a,dph       ; 1
      xch   a,r4        ; 1
      movx  @dptr,a     ; 2
      inc   dptr        ; 2
      xch   a,r0        ; 1
      xch   a,dpl       ; 1
      xch   a,r0        ; 1
      xch   a,r4        ; 1
      xch   a,dph       ; 1
      xch   a,r4        ; 1
      djnz  r7,loop     ; 2 => 22cycles * length
      djnz  r6,loop     ; 2 => 2cycles * (1 + (length/256))
```

XDATA COPY ROUTINE–ULTRA HIGH-SPEED MICROCONTROLLER

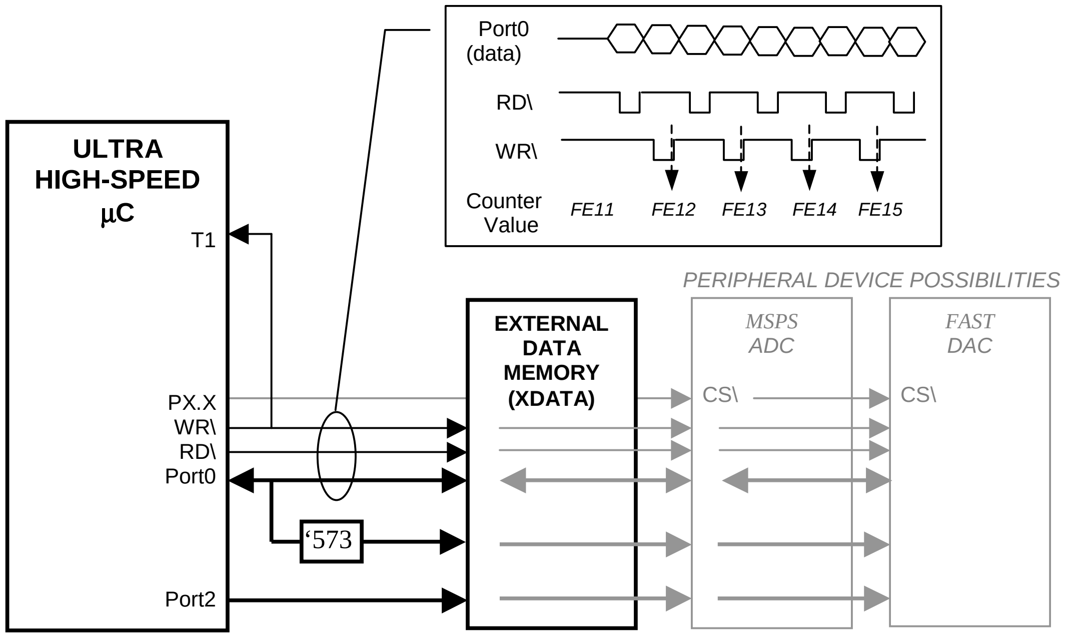
The Ultra High-Speed Microcontroller contains two data pointers (DPTR = DPH + DPL; DPTR1 = DPH1 + DPL1), which allows separate source and destination data pointers. In addition, it implements hardware controls to automatically toggle between the data pointers and automatically increment or decrement the active data pointer in response to certain DPTR-related instructions. Full details of the enhanced dual data pointers can be found in the *DS89C420 Ultra High-Speed Microcontroller User's Guide*. The code below demonstrates how the enhanced dual data pointers simplify the copy routine. Instructions that automatically toggle the active data pointer have been marked with a [T] and those that automatically advance the data pointer have been marked with a [+/-]. For the purpose of comparison with the original 8051 architecture, the standard 8051 external P2, P0 memory bus structure has been assumed, thus giving a minimum MOVX duration of 5 clock cycles (1 clock cycle for MOVX opcode fetch, 4 clock cycles for the data memory access). Although cycle counts for the Page Mode 1 external bus configuration are not reflected in the code example directly below, it should be noted that Page Mode 1 does provide the absolute minimum external MOVX duration (3 clock cycles = 1 clock cycle for the MOVX opcode fetch + 2 clock cycles for the data memory access).

```
; Ultra High-Speed Micro Copy - Enhanced Dual DPTRs
; DPS.4 (AID) = 1; DPS.5 (TSL) = 1;
;     R6:R7 control copy length
loop:                   ; Cycle Count @ 1clk/cycle
      movx  a,@dptr     ; 5 [T][+/-]
      movx  @dptr,a     ; 5 [T][+/-]
      djnz  r7,loop     ; 5 => 15 cycles * length
      djnz  r6,loop     ; 5 => 5 cycles * (1 +(length/256))
```

XDATA COPY ROUTINE–USING A TIMER/COUNTER

To use a timer/counter, the application must make available a timer/counter and its associated input pin for the duration of the copy routine. The basic idea is to use the on-chip counter to track and terminate the copy loop instead of using working registers. The WR\ strobe serves as the input signal to the counter. Figure 1 shows an example hardware configuration.

Figure 1. **HARDWARE DIAGRAM FOR WR\ STROBE COUNTING**



All Ultra High-Speed Microcontroller timer/counter input pins (T0, T1, T2) are able to sample an input frequency equivalent to one-quarter the system clock frequency. This means that the input signal to be sampled must have minimum high and low times of 2 system clock cycles each. Aside from fastest MOVX in 1-cycle or 2-cycle Page Mode 1, the WR\ strobe meets this criteria for all other external MOVX operations. Before entering the copy loop, the 16-bit counter is loaded with the appropriate value (2^{16}- #bytes to copy), the timer/counter interrupt is enabled, and the enhanced data pointers are configured for the most expeditious copy. The last byte copied causes the counter to roll over and generates a timer interrupt, which allows code execution to be returned to the main program.

BENEFITS OF USING THE TIMER/COUNTER METHOD

The primary benefit of using a timer/counter is increased performance while allowing a more forgiving xdata access time. It has already been noted that the fastest copy loop execution time is achieved by configuring the external bus structure to Page Mode 1 (Port2 = multiplexed address MSB/LSB and Port0 = data). Although Page Mode 1 gives the ultimate performance, it also requires the fastest xdata access time ($t_{RLDV} < t_{CLCL}$). The timer/counter method gives nearly the same performance while almost doubling maximum xdata access time.

Moreover, it is not expected that all Ultra High-Speed Microcontroller designs will use the new bus structure. Some system designers may wish to retain the legacy 8051 bus interface or may even be dropping a DS89C420 into an existing socket. Given these circumstances, the counter-terminated copy loop provides a high-performance compromise.

Because the timer/counter-terminated copy loop relies upon internal 16-bit timer hardware to count the number of external read/write events, it does not require separate DJNZ instructions to keep track of a 16-bit loop-control variable. This allows the application to conduct high-speed synchronous transfers for data lengths greater than 256 bytes.

Table 1 compares the copy loop for the original 8051 architecture versus three possible Ultra High-Speed Microcontroller configurations. The counter terminated copy loop has been highlighted. Following the table is an example code listing for a timer/counter-terminated copy. The copy loop has been highlighted in the code for comparison with the previous two examples.

Table 1. **COPY LOOP COMPARISON**

MICROCONTROLLER OPTION	**SPEED (ms)** (1000-byte xdata copy-loop duration)	**FAST XDATA ACCESS TIME** (t_{RLDV})	**SYNCHRONOUS TRANSFER ?**	**SPECIAL FEATURES USED**
Ultra High-Speed Microcontroller[1]	0.441	$< 1 \times t_{CLCL}$	No	Page Mode 1 Bus Structure
Ultra High-Speed Microcontroller[1]	0.480	$< 2 \times t_{CLCL}$	Yes	Timer/Counter and Timer Input Pin
Ultra High-Speed Microcontroller[1]	0.601	$< 2 \times t_{CLCL}$	No	-
Standard 8051[2]	10.568	$< 5 \times t_{CLCL}$	No	-

[1] Machine cycle = 1 x t_{CLCL}; Enhanced Dual Data Pointers with Auto-Inc/Dec, Auto-Toggle
[2] Machine cycle = 12 x t_{CLCL}; Single Data Pointer

COUNTER TERMINATED COPY METHOD (CODE EXAMPLE)

```
;-------------------------------------------------------------
; Demonstrate use of Timer/Counter 1 to terminate copy loop
; Use P3.6 (WR\) as an input to Timer/Counter 1 (P3.5)
; In this example:
; 1) Source, dest, and length defined as constants.
; 2) Code saves only DPTR0 state under the assumption that a
;    single DPTR is normally used and that the second (DPTR1)
;    is enabled only for certain routines (such as this one).
; 3) Code disables all other interrupts during the copy.
; 4) External non-overlapping xdata to xdata transfer
;-------------------------------------------------------------
$include(420.def)                  ; include file w/89C420 SFRs
source      equ   0100h            ; source xdata address
dest        equ   0200h            ; dest xdata address
length      equ   256d             ; #bytes to copy
;-------------------------------------------------------------
org   0h
      ljmp  0100h
;-------------------------------------------------------------
;Timer/Counter 1 interrupt
;-------------------------------------------------------------
org   1bh
      clr   tr1                    ; disable timer/counter 1
      pop   acc                    ; pop 'ajmp' loop addr
      pop   acc                    ;   from the stack
      pop   dps                    ; return pre-transfer
      pop   dph                    ;   DPTR state
      pop   dpl
      pop   ie                     ; return interrupt config
      pop   eie
      reti                         ; back to instruction
                                   ; after "xmemcpy_.." call
;-------------------------------------------------------------
; Main
;-------------------------------------------------------------
org   0100h
      orl   tmod, #50h             ; 16-bit counter
      anl   ckcon, #0F8h           ; fast 2-cycle MOVX
      call  xmemcpy_count          ; call xdata copy code
      sjmp  $
;-------------------------------------------------------------
; 1) Save interrupt enable registers, make only T/C#1 enabled
; 2) Save DPTR0
; 3) Timer/Counter 1 loaded with (2^16-#bytes to copy)
; 4) Configure source/destination pointers
; 5) Execute copy loop
;     - last write rolls the Counter
;     - Timer/Counter 1 Interrupt breaks the loop
;-------------------------------------------------------------
xmemcpy_count:
      push  eie
      push  ie
      mov   eie, #00h              ; disable other ints
      mov   ie, #88h               ; EA=1, ET1=1
      push  dpl                    ;save DPTR state
      push  dph
      push  dps
      mov   th1, #high(-length)    ; copy length
      mov   tl1, #low(-length)
      setb  tr1                    ; enable Timer1
      orl   dps, #30h              ; SEL=0,TSL=1, AID=1
      mov   dptr, #source          ; DPTR0 = source addr
      mov   dptr, #dest            ; DTPR1 = dest addr
transfer:
        movx a, @dptr              ;[5] read from @DPTR0
        movx @dptr, a              ;[5] write to @DPTR1
        ajmp transfer              ;[2] in loop ‘til int
;                                  ----
;                                  [12] total
end
```

USING INTERNAL XRAM FOR HIGHER TRANSFER RATES

The basic principle of using a timer/counter to improve execution efficiency not only applies to external data copy routines, but can also be used for external data reads or writes to/from internal data memory. If, for instance, the application required no more than 1024 bytes be either read or written from external memory (or to/from a parallel peripheral), the internal 1kB data memory present on the DS89C420 could be used to further increase the transfer rate.

Since the internal MOVX operation requires only 2 cycles, the read/write transfer loop, which has been the subject of discussion to this point, could be reduced to 9 cycles for an effective transfer rate of 3.67MBps (@33MHz) and would occupy a mere 4B of code space (code below). Please remember that one of the two MOVX operations, either the read or the write, must be executed on external memory so that the RD\ or WR\ strobe increments the timer/counter, eventually terminating the transfer loop. The first code example below gives cycle counts corresponding to an external MOVX write and the second example shows an external MOVX read. Also note that when the RD\ signal is counted, the counter should be initialized to [2^{16} - (#bytes to copy +1)] in order that the final byte write will occur before the loop is interrupted.

```
loop:                   ; 89C420 Cycle Count    / Byte Count
      movx  a,@dptr     ; 2 (internal MOVX)     / 1
      movx  @dptr,a     ; 5 (external MOVX)     / 1
      ajmp  loop        ; 2                     / 2
                        ; 9 cycles              / 4 bytes

-- OR --

loop:                   ; 89C420 Cycle Count    / Byte Count
      movx  a,@dptr     ; 5 (external MOVX)     / 1
      movx  @dptr,a     ; 2 (internal MOVX)     / 1
      ajmp  loop        ; 2                     / 2
                        ; 9 cycles              / 4 bytes
```