TMS320 DSP
Number 26
DESIGNER'S NOTEBOOK
TEXAS INSTRUMENTS

A Novel Way of Using TMS320C40 Cache

Contributed by Keith Larson

Design Problem How can I place any value into the TMS320C40 cache?

Solution A usual approach to loading the cache is to unfreeze the cache and let it always be filled, hoping for a looped block of code. By freezing the cache at the end of time-sensitive routines, a little more performance can be expected since the cache does not always have to be filled from external memory on the first pass through. However, the cache may not always fill completely due to code dependencies or conditional branching. In this case, it would be desirable to load the contents of any address into the cache.

The routine shown in Figure 1 will poke opcodes from an arbitrary address into the cache using a feature of the interrupt processor. In this case, when the RETI opcode is executed, writing PGIE to GIE, one opcode following the RETI is protected from interrupts and is always fetched (and executed). By properly controlling the value of TOS, it is possible to load any external address pointed to by TOS into the cache! In this case, an interrupt vector is used to loop the cache loader back to itself each time an opcode is loaded into the cache.

Caution: Since any opcode can be executed in any order, it is important to control the potential action of all opcodes fetched in this manner. For example, if an opcode is supposed to write data to a location pointed to by an auxiliary register, it would make sense to make sure that all the auxiliary registers point to a safe "dummy" location. Likewise, adequate controls should be placed on the loader to ensure that the correct status is always loaded back into the CPU after each cache load.

Also note that DATA values can be poked into the cache. Since all opcodes going into the cache are executed, unpredictable results may occur when loading such a value. If a DATA value is loaded into the cache, that value is NOT accessible as data from the cache since the DDATA bus cannot be connected to the cache for a transfer. IE-only program fetching is allowed from the cache.

Please note: The routine shown in Figure 1 does not include a full save and restore, nor does it control the values of the data pointers (DP and ARs). It is the programmer's responsibility to add the code necessary to provide the context save routines and other error checking.

Since the 'C40 cache is filled on 32 words boundary, sometimes the program address alignment is needed in order to put the maximum length of the program into the cache.

```
;_________________________;
; void lcache(*ptr, len);                      ;
;                                              ;
; loads the program pointed to by ptr into     ;
; the cache from external memory               ;
;_________________________;
        .global start,test,FLAG_0,_lcache
        .global dec,inc,more,RST,NMI,TINT_0
        .text
RST     .word   $               ;set up temporary IVTP in
; need to align at 512 word boundary
NMI     .word   $+1             ;external RAM
TINT_0  .word   _lcache         ;
        ;___________-;
start:  ldp     RST             ;set up a new vector table
        ldi     @RST,R0         ;
        ldpe    R0,IVTP         ;
        ldi     @stack,SP       ;set up a runtime stack
        ldi     @a_test,R0      ;subroutine to load is “test”
        sti     R0,@APC         ;
        ldi     16,R0           ;load 16 cache locations
        sti     R0,@CNT         ;
        sti     IIF,@FLAG       ;keep original IIF
        and     0E3FFh,ST       ;clear, thaw and enable cache
        or      5800h,ST        ;
        call    $+1             ;a way to push PC on stack
        pop     R0              ;takes care of first dummy pop
        addi    4,R0            ;
        push    R0              ;
        call    _lcache         ;call the cache loader
        or      00C00h,ST       ;freeze and enable cache
        ldi     @FLAG,IIF       ;restore IIF
        ;___________-;
test    ldi     15,R0           ;Test code to cram into the
                                ;cache
dec     subi    1,R0            ;with conditional branches
        bnn     dec             ;
        ldi     -15,R0          ;
inc     addi    1,R0            ;
        bn      inc             ;
        bud     test            ;
        nop                     ;
        nop                     ;
        nop                     ;
        ;___________-;
_lcache ldp     APC             ;
        ldi     @CNT,R1         ;
        subi    1,R1            ;
        bnz     more            ;
        pop     R0              ;pop junk address
        rets                    ;return to original caller
        ;___________
```

Figure 1.

(continued on next page)

```
more    sti     R1,@CNT         ;
        pop     R1              ;pop junk address
        ldi     @APC,R1         ;create "new" return address
        addi    1,R1            ;
        sti     R1,@APC         ;
        push    R1              ;
        ldi     @TINT0,IIF      ;turn on TINT0
        ldi     1,IIE           ;enable TINT0
        reti                    ;after return, fetch 1 opcode
        ;_____________-;
        .global  FLAG,APC,CNT
        .global  a_test,stack
FLAG    .word   0               ;Original IIF register
APC     .word   0               ;Auxiliary Program Counter
CNT     .word   0               ;length to load
TINT0   .word   01000000h       ;
a_test  .word   test-1          ;
stack   .word   $               ;reserve stack locations
        .end
```

Figure 1. (continued)