

APPENDIX A

Sample Source Code for the TMS320C3x/4x DSP Processors

A.1 Sample Code

The sample code on the following pages provides an example of the documentation and implementation techniques for the source code of this product. Source licenses are available for this product.


```

;      Register use
;      Used:          FP SP
;                    R0 R1 R2
;                    AR0 AR1 AR2 AR3
;                    IR0 IR1
;                    RC (RS RE)
;
;      Restored:     FP
;
; ISSUES (C40)
;
;-
      .global _vadd
FP      .set      AR3          ; Frame Pointer Assigned to AR3

_vadd          ; Entry Point

      PUSH      FP          ; Save Frame Pointer
      LDI       SP, FP      ; Load Stack Pointer to Frame Pointer

; The following instructions
; initialize the register set to
; Casel, which models vectors where
; all strides are equal. This is
; logical as this will be the most
; frequent case. The register set is
; initialized as follows:
;
; a[i] = b[j] = c[k] where i,j,k
; can equal 1 or greater.
;
; Not that this case also covers:
;
; a[i] = b[j] != c[k] where i and j
; must be equal. k may be different
; than i,j but does not have to be.

      .if .REGPARAM == 0

; STACK PASSING MODEL
      LDI       *-FP(2), AR0 ; Address of input vector [a]
      LDI       *-FP(3), IR0 ; Vector [a] element stride
      LDI       *-FP(4), AR1 ; Address of input vector [b]
      LDI       *-FP(5), R3  ; Vector [b] element stride
      LDI       *-FP(6), AR2 ; Address of output scalar [c]
      LDI       *-FP(7), IR1 ; Vector [c] element stride
      LDI       *-FP(8), RC  ; Element Count -> Repeat Counter

      .else

; REGISTER PASSING MODEL
      LDI       AR2, AR0    ; Address of input vector [a]
      LDI       R2, IR0    ; Vector [a] element stride
      LDI       R3, AR1    ; Address of input vector [b]
      LDI       RC, R3     ; Vector [b] element stride

```

```

LDI      RS, AR2          ; Address of output scalar [c]
LDI      RE, IR1         ; Vector [c] element stride
LDI      *-FP(2), RC     ; Element Count -> Repeat Counter

.endif

ADDI     -2, RC, RC      ; DECREMENT REPEAT COUNTER
;
; R1 holds b[j] after initial loading
;
; R0 holds results of a[i] + b[j]
; before final storage into c[k]
;
; R2 is used in case2 to hold the
; stride k of vector [c]
;
CMPI     R3, IR0         ; Compare Stride [b] to [a]
BNE      case2          ; Branch to case2 if A(x) != B(x)
NOP      ; No Operation Due to Branch
NOP      ; No Operation Due to Branch
NOP      ; No Operation Due to Branch
;
; case1 Variables
;
; [a]      =      AR0
; stride i =      IR0
;
; [b]      =      AR1      +[i]->R1
; stride j =      IR0
;
; [c]      =      AR2
; stride k =      IR1
;
; R0      =      Temp Hold for [b]
; R1      =      Temp Hold or [a]+[b]
; RC      =      Repeat Counter

case1    ADDF      *AR0++(IR0), *AR1++(IR0), R0
; prime the instruction pipeline
RPTB     loop1          ; Loop for case1
LDF      *AR1++(IR0), R1
; Load value of [b] into R1 for
; upcoming parallel add.
;
loop1    ADDF3     *AR0++(IR0), R1, R0
||      STF       R0, *AR2++(IR1)
; Add [a] to R1 and place in R0,
; while storing previous R0 value
; in vector [c]; signify end of
; loop for case 1
;
BUD      end           ; Branch to "end" (of program) (1)
STF      R0, *AR2++(IR1)
; Empty the instruction "pipeline" (2)
NOP      ; No Operation (3)
NOP      ; Last Instruct to before Branch (4)
;
; START OF CASE 2

```

```

;
; The following instructions
; initialize the register set to
; Case2, which models vectors where
; the strides are not equal. The

; a[i] != b[j] or a[i] != c[k] where
; i,j,k can all be different.

; Not that case1 covered b[j] != c[k]
; but this portion of the routine can
; also be used to compute this case,
; albeit a little slower.

; For case2, the register set is
; initialized as follows:

; case2 Variables
;
; [a]      =   AR0
; stride i =   IR0
;
; [b]      =   AR1
; stride j =   IR1 (was IR0 earlier)
;
; [c]      =   AR2
; stride k =   R2 (was IR1 earlier)
;
; R0       =   Temp Hold or [a]+[b]
; R2       =   Holds stride c[k]
; RC       =   Repeat Counter
;
case2  ADDI    1, RC          ; Add 1 to Repeat Counter to Re-
                                ; initialize Repeat Block Counter
      LDI    IR1, R2        ; Load stride of j of [b] into R2
      LDI    R3, IR1        ; Change [IR0] to [IR1] from initial
                                ; assignment at start of program
      RPTB   loop2          ; Loop for case2

      ADDF3  *AR0++(IR0), *AR1++(IR1), R0
                                ; Load [a] + [b] and place in R0

      STF    R0, *AR2        ; Store results of [a]+[b] in [c]
loop2  ADDI    R2, AR2        ; Increment Stride factor k
                                ; for Vector [c]
end    POP    FP            ; Restore Registers

      RETS
      .end

```

