TEXAS INSTRUMENTS

Interfacing TMS320C5000 DSP to MSP430 Mixed Signal Microcontroller

Muhammad Haque

Digital Signal Processing Solutions

ABSTRACT

The TMS320C5000 family of digital signal processors (DSPs) features Host Port Interface Controllers (HPI) and Direct Memory Access Controllers (DMAC) for efficient data movement without any CPU involvement. The HPI enables the DSP to interface to host processors (typically microcontrollers) bidirectionally with minimal or no external interface logic. This application report presents a hardware interface and a software protocol to communicate between the TMS320VC5402 DSP and the MSP430x33x mixed signal microcontroller. A set of simple arithmetic functions are implemented on the DSP, under the supervision of the microcontroller.

Contents

1   Introduction

Typical high performance digital signal processing applications require a DSP to handle signal processing in real time and a microcontroller unit (MCU) to process all other general purpose tasks. The HPI allows a host MCU with separate or multiplexed address and data buses to access the DSP's internal memory without any CPU involvement. The HPI also allows a host to interrupt the DSP and vice versa. The interface requires no additional logic circuitry and thus saves board space, reduces system cost, and cuts down valuable development time.

2 Enhanced Host Port Interface (HPI-8)

The Enhanced Host Port Interface, also referred to as the HPI-8, connects to a host as a peripheral. The host acts as the master and accesses the DSP memory through three dedicated HPI registers. These registers are HPI address (HPIA), data (HPID), and control (HPIC) registers. The host can access all three registers, but the DSP can access the HPIC register only.

The interface consists of an 8-bit bi-directional data bus (HD0 – HD7) and various control signals. Due to the 16-bit word structure of the C5000 DSPs, all the host data transfers are accomplished in two consecutive 8-bit operations. The logic state of the byte identification input (HBIL) pin indicates whether the first or the second byte is being transferred. The byte-order bit (BOB) of the HPIC register determines the placement of the two transferred bytes. If BOB=1, the first transferred byte is the least significant byte; if BOB=0, the first transferred byte is the most significant byte.

The HPIA register serves as a pointer to the DSP's internal memory. The HPID register is used to move data to and from the address pointed to by the HPIA register. The HPIC register controls and monitors the HPI-8 operations, such as, transferred byte placement, interrupt control, etc.

Two host control (HCNTL0, and HCNTL1) input pins indicate which HPI internal register is being accessed, and the type of access. Table 1 describes the HCNTL0/1 pin functions.

Table 1. HPI-8 Input Control Signals and Function Selection

HCNTL1	HCNTL0	Description
0	0	Host can read from and write to the HPI control register, HPIC.
0	1	Host can read from and write to HPI data latches. HPIA is automatically post-incremented during each read, and pre-incremented during each write.
1	0	Host can read from and write to HPI address register, HPIA. This register points to the DSP on-chip RAM
1	1	Host can read from and write to HPI data latches. HPIA is not affected.

Two data strobe (HDS1 and HDS2) input pins control data transfer during the host access cycles. Hosts with separate read and write strobes connect to HDS1 and HDS2. The data strobes are internally exclusive-NORed, so hosts with a single data strobe can connect to either HDSx pin and have the other HDSx pin connected to a logic high or low depending on the polarity of host data strobe. For example, if the host data strobe is a true logic high then the unconnected HDSx pin should be tied to a logic low.

The host address strobe (HAS) input pin is connected to a logic high if the host has a separate address and data bus. Hosts with multiplexed address and data bus should connect HAS to their address latch enable (ALE) pin or equivalent.

The host read/write (HRW) input pin is used to indicate whether the host is performing a read or write access. The HRW pin should stay high during the read cycle and stay low during the write cycle.

The host chip select (HCS) input pin enables the HPI and must stay low during an HPI access.

The HPI ready (HRDY) output pin indicates if the HPI is ready for a transfer or not. When the HRDY pin is high and the HCS pin is low, the HPI is ready for a transfer. The HRDY pin can be connected to the host READY or an equivalent (if available) pin to generate host wait-states. The HRDY pin is always high when the HPI is not enabled (HCS pin logic high).

The HPI-8 supports two interrupts, these interrupts are controlled by the DSPINT and the HINT bits of the HPIC register. Writing a 1 to the DSPINT bit by the host sends an interrupt to the DSP. When the DSP writes a 1 to HINT bit, the HINT pin signal level transitions from high to low. This pin can be used to interrupt the host. Writing a 1 to the HINT bit by the host clears this bit to 0 and drives the HINT pin output to high again.

3   MSP430 Mixed Signal Microcontroller

The Texas Instruments MSP430 series is an ultra low power 16-bit RISC microcontroller family with a rich set of built-in hardware features, like LCD driver, A to D converter, timers with compare/ capture unit, watch dog timer, serial communication interface, and dedicated programmable I/Os. MSP430 family currently consists of three sub-families with different sets of integrated features targeting various applications. Among the three sub-families MSP430x33x is most suitable for interfacing to a C5000 DSP through HPI-8 because of its additional dedicated I/O ports.

4   Signal Connections

Figure 1 shows the pin connections between the two devices. The HPI-8 connects to the MSP430x33x general purpose I/O port P1 and P2. The HPI-8 data bus (HD0 – HD7) interfaces with the port P1 and the HPI control signals are driven by the port P2. The MSP430 port P1 and P2 are 8-bit I/O ports with features to individually select the function of each pin as input or output and as an interrupt source. The port P1 is configured as input prior to an HPI read cycle and as output prior to an HPI write cycle. Pins P2.0 – P2.4 are configured as outputs to drive the HPI control pins, pins P2.6 and P2.7 are configured as inputs for the host ready (HRDY) and the host interrupt (HINT) signals.
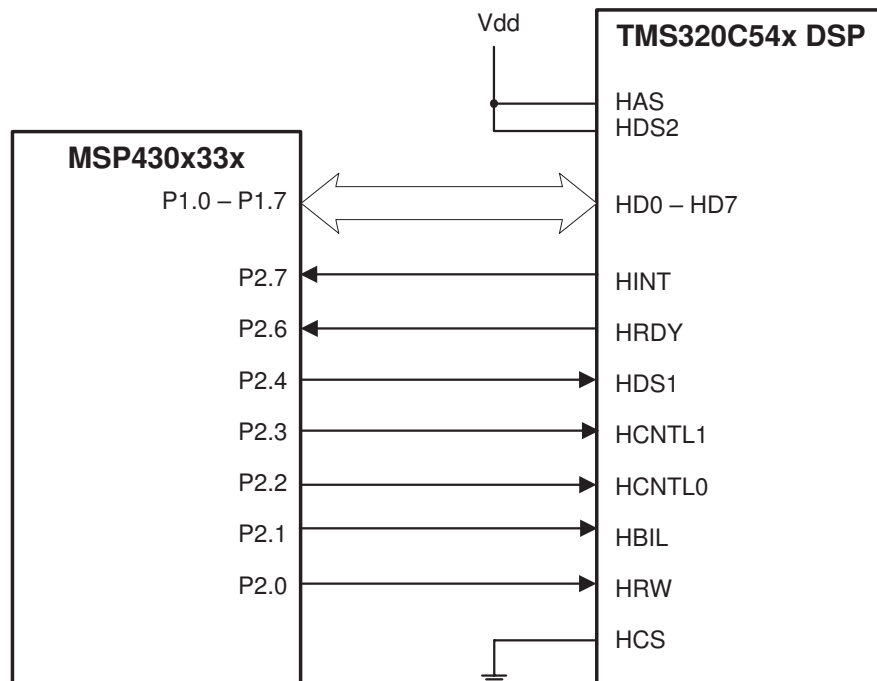

Figure 1. HPI-8 and MSP430x33x Pin Connections

5 Software Description

The interfacing software components consist of two parts:

- Host routine running on MSP430x33x.
- DSP routines running on TMS320VC5402.

5.1 Host Routine

The host routines initiate the communication between the two processors. The host routines begin by configuring the port P1 and P2 pin functions and initializing the HPIC register. After initializing HPIC register, the host loads the DSP internal data memory address at 60h through 62h with the host command word (host_cmd) and two 16-bit signed numbers (num1 and num2), respectively. Then the host routine interrupts the DSP by setting the DSPINT bit of the HPIC register to 1. At this point, the host idles and waits for the DSP to interrupt it by changing the HINT pin output from logic high to low. When the HINT pin drives the host P2.7 pin low, the host branches to the associated interrupt subroutine. In the interrupt subroutine, the host clears the HINT bit of the HPIC register by writing a 1 to this bit and then reads the result from the DSP internal data memory address 63h. Normally, the host routine should poll the HRDY pin before initiating an HPI read or write cycle. Polling the HRDY pin is not necessary in this case because the DSP runs at a considerably faster speed (x25) than the host processor. See Appendix A for the host routine source code.

5.2 DSP Routines

The DSP routines initialize the DSP runtime environment, take the DSP to an idle loop, and wait for the DSPINT interrupt from the host. Upon arrival of the interrupt, the DSP reads the host command (host_cmd) from its internal data memory address 60h.

- If bit0 of the host command is 1, the DSP adds the contents (num1 and num2) of the data memory addresses 61h and 62h and stores the result at address 63h.

- If bit1 is set to 1, the DSP subtracts the contents (num1 from num2) of the data memory addresses 61h and 62h and stores the result at address 63h.

- If bit2 is set to 1, the DSP multiplies the contents (num1 times num2) of the data memory addresses 61h and 62h and stores the result at address 63h.

- After storing the result, the DSP interrupts the host by setting the HINT bit of the HPIC register and returns from the DSPINT interrupt subroutine. See Appendix B for the DSP routine source code.

6 Summary

This application report describes the hardware interface between a 16-bit microcontroller and the enhanced 8-bit Host Port Interface (HPI-8) of C5000 DSP family by using a TMS320VC5402 DSP and a MSP430x33x mixed signal microcontroller. The application report also provides the software routines used for the communication between these two devices.

7 References

1. *TMS320C54x DSP Reference Set, Volume 1: CPU and Peripherals*, SPRU131.
2. *TMS320C54x DSP Reference Set, Volume 5: Enhanced Peripherals*, SPRU302.
3. *TMS320C54x Assembly Language Tools User's Guide*, SPRU102.
4. *TMS320C54x Optimizing C Compiler User's Guide*, SPRU103.
5. *TMS320C54x C Source Debugger User's Guide*, SPRU099.
6. *TMS320VC5402 Fixed-Point Digital Signal Processor*, SPRS097.
7. *MSP430 Architecture Guide and Module Library*, SLAUE10.
8. *MSP430 Assembly Language Tools User's Guide*, SLAUE12.
9. *MSP430 Software User's Guide*, SLAUE11.
10. *MSP430x33x Mixed Signal Microcontrollers*, SLAS163.

Appendix A   Host Routine

```
*************************************************************************
; File Name: Host_430.asm
;
; Target System: MSP430 EVK330
;
; Description:
;
;   Code initialize C5000 Enhanced 8-bit Host Port
;   Interface and controls data exchange with DSP
;   internal data RAM.
;
;   Load the code on EVK330, change the memory content
;   of host_cmd, num1, and num2 with test values.  Run
;   the program, MSP430 will send the host_cmd, num1,
;   num2 to DSP, and read and store outcome of DSP operation
;   on num1 and num2 in result.
;
*************************************************************************



*** Set this variable to '1' for the use on the Simulator ***
SIM        .set   0              ;1 = Simulator
                                 ;0 = STK/EVK

RAM_orig  .set   0240h       ; Free Memory startadress
SP_orig   .set   05DEh       ; stackpointer

***************************************************************
* HCNTL1 & HCNTL0 value
***************************************************************
HPI_CtrlReg           .set 00h
HPI_DatReg_w_AddrInc  .set 04h
HPI_AddrReg           .set 08h
HPI_DatReg            .set 0Ch


***************************************************************
* HPIC config value
***************************************************************

SET_BOB     .set 0101h   ;config HPI for low byte first
CLR_HINT    .set 0909h   ;clear DSP asserted host interrupt
SET_DSPINT  .set 0505h   ;Interrupt DSP


***************************************************************
* Control register definitions
***************************************************************
IE1               .equ    00h
IE2               .equ    01h
IFG1              .equ    02h
IFG2              .equ    03h
ME1               .equ    04h
ME2               .equ    05h
```

```
WDTCTL          .equ    0120h
WDTHold         .equ    080h
WDT_wrkey       .equ    05A00h


GIE             .equ    08h


*************************************************************
* DIGITAL I/O Port1/2
*************************************************************

P1IN   .equ   020h
P1OUT  .equ   021h
P1DIR  .equ   022h
P1IFG  .equ   023h
P1IES  .equ   024h
P1IE   .equ   025h
P1SEL  .equ   026h


P2IN   .equ   028h
P2OUT  .equ   029h
P2DIR  .equ   02Ah
P2IFG  .equ   02Bh
P2IES  .equ   02Ch
P2IE   .equ   02Dh
P2SEL  .equ   02Eh


*************************************************************
* host command definition
*************************************************************


_ADD   .equ   01h
_SUB   .equ   02h
_MUL   .equ   04h


*************************************************************
* RAM allocation
*************************************************************

        .bss      host_cmd,2,220h  ; host command, 0x01 = add,
                                    ; 0x02 = sub, 0x04 = mul
        .bss      num1,2            ; 1st num to send to dsp
        .bss      num2,2            ; 2nd num to send to dsp
        .bss      result,2          ; result returned from dsp
        .bss      tempReg0,2        ; 16 bit temp reg


;***********************************************************************
; Reset : Initialize processor
;***********************************************************************
        .sect "MAIN",RAM_orig
RESET
        MOV #SP_orig,SP                     ; initialize stackpointer
        MOV #(WDTHold+WDT_wrkey),&WDTCTL ; Stop Watchdog Timer
```

```
;****************************************************************************
; enable monitor and clear  special function registers
;****************************************************************************

        MOV.B  #08h,IE1 ; ! Monitor !
        CLR.B  IE2
        CLR.B  IFG1
        CLR.B  IFG2



;****************************************************************************
; Init dig i/o port P2 for HPI control signals
;****************************************************************************

        MOV.B  #010h, P2OUT  ;output level:P2.4 = hi, rest low
        MOV.B  #03Fh, P2DIR  ;P2.0 - P2.5 output, P2.6 - P2.7 input
        MOV.B  #080h, P2IES  ;HINT=P2.7->hi to lo
        MOV.B  #080h, P2IE   ;P2.7 interrupt enable

;****************************************************************************
; Init HPIC with 0x0101 (low byte first, high byte second mode)
;****************************************************************************

        MOV.W  #SET_BOB, &tempReg0 ;tempReg0 = 0101h
        MOV.W  #tempReg0, R6
        MOV.B  #HPI_CtrlReg, R5    ;/HCS=hi
        CALL   #HPI_WRITE           ; write 0000h to HPIC

;****************************************************************************
; Init HPIA with 0x0060
;****************************************************************************

        MOV.W  #60h, &tempReg0     ;tempReg0 = 00h
        MOV.W  #tempReg0, R6
        MOV.B  #HPI_AddrReg, R5    ;/HCS=hi
        CALL   #HPI_WRITE           ; write 0000h to HPIC

;****************************************************************************
; Load host_cmd word in DSP memory (0x60)
;****************************************************************************

        MOV.W  #_ADD, &host_cmd    ;host_cmd = 01h
        MOV.W  #host_cmd, R6
        MOV.B  #HPI_DatReg, R5     ;R5=HPID access with addr inc bit orientation
        CALL   #HPI_WRITE           ; write 0000h to HPIC

;****************************************************************************
; Increment HPIA and Load num1 in DSP memory (0x61)
;****************************************************************************

        MOV.W  #num1, R6
        MOV.B  #HPI_DatReg_w_AddrInc, R5 ;R5=HPID access with addr inc bit orientation
        CALL   #HPI_WRITE                 ; write 0000h to HPIC
```

```
;*************************************************************************
; Increment HPIA and Load num2 in DSP memory (0x62)
;*************************************************************************

        MOV.W   #num2, R6
        MOV.B   #HPI_DatReg_w_AddrInc, R5   ;R5=HPID access bit orientation
        CALL    #HPI_WRITE                  ; write 0000h to HPIC

;*************************************************************************;
; Set DSPINT bit of HPIC register to indicate that new data have been
; transferred from the host
;*************************************************************************

        CALL    #INTRPT_DSP
        JMP     $                           ; Endless Loop

;*************************************************************************
; Subroutine: HPI_WRITE
;
; Description: Writes a 16-bit value pointed by the R6 to HPIC,
;   HPIA, or HPID depending on the content of R5
;
; Input: R6 = Addr of 16-bit value to be sent to HPI
;   R5 = HPI access Control Signal bit orientation
;*************************************************************************

HPI_WRITE
        MOV.B   #0FFh, &P1DIR       ;configure P1.0 - P2.7 as output

;   ============ write low byte ================
        MOV.B   @R6+, P1OUT         ; put 8 LSB of data on the bus
        BIS.B   #10h, R5            ;set /HDS1 bit hi
        MOV.B   R5, P2OUT           ;put HPI control signals out on P2
        BIC.B   #10h, P2OUT         ;/HDS1=lo, start HPI cycle
        BIS.B   #10h, P2OUT         ;pull /HDS1 hi, end HPI cycle

;   ============ write high byte ================
        MOV.B   @R6, P1OUT          ;put 8 MSB of data on the bus
        BIS.B   #12h, R5            ;/HBIL=hi, HDS1=hi
        MOV.B   R5, P2OUT           ; put HPI control signals out on P2
        BIC.B   #10h, P2OUT         ;/HDS1=lo, start HPI cycle
        BIS.B   #10h, P2OUT         ;pull /HDS1 hi, end HPI cycle
        RET

;*************************************************************************
; Subroutine: HPI_READ
;
; Description: Reads a 16-bit value from HPIC, HPIA, or HPID
;   depending on the content of R5 and stores it in
;   memory location pointed by R6
;
; Input: R5 = HPI access Control Signal bit orientation
; Output: R6 = Store addr for 16-bit value read from HPI
;
;*************************************************************************
```

```
HPI_READ
        MOV.B  000h, P1DIR      ;P1.0 - P2.7 input

;  ============ read low byte ================
        BIS.B  #11h, R5         ; /HCS=hi, HR/~W=hi
        MOV.B  R5, P2OUT        ; put HPI control signals out on P2
        BIC.B  #10h, P2OUT      ; /HDS1=lo, start HPI cycle
        MOV.B  P1IN, 0(R6)      ; read 8 LSB data from the bus
        BIS.B  #11h, P2OUT      ; pull /HDS1 hi, end HPI cycle

;  ============ read high byte ================
        BIS.B  #13h, R5         ;/HBIL=hi, HCS=hi, HR/~W=hi
        MOV.B  R5, P2OUT        ; put HPI control signals out on P2
        BIC.B  #10h, P2OUT      ;/HDS1=lo, start HPI cycle
        MOV.B  P1IN, 1(R6)      ;read 8 MSB data from the bus
        BIS.B  #10h, P2OUT      ;pull /HDS1 hi, end HPI cycle
        RET

;****************************************************************************
; Subroutine: INTRPT_DSP
;
; This subroutine sets DSPINT bit of HPIC register to indicate that
; new data have been transferred from the host
;
; Input: None
; output: None
;
;****************************************************************************

INTRPT_DSP
        MOV.W  #SET_DSPINT, &tempReg0   ; tempReg0 = bit value to load on HPIC
        MOV.W  #tempReg0, R6
        MOV.B  #HPI_CtrlReg, R5              ; R5=HPIC access bit orientation
        CALL   #HPI_WRITE                    ; set DSPINT bit of HPIC
        RET                                   ; return from subroutine


;****************************************************************************
; Interrupt Subroutine: HPI_ISR
;
; Description:
;
; This subroutine reads the result of arithmetic operation done by DSP
; on two 16-bit numbers and stores the value in memory location allocated
; for variable "result".
;
; Input: None
; Registers Used: R6 (Store addr (result) for 16-bit value read from HPI)
;    and R5
;
;****************************************************************************

HPI_ISR
        PUSH   R5            ; save R5
        PUSH   R6            ; save R6
```

```
;======  pull HINT signal high by writing 1 to HINT bit field of HPIC  =======
        MOV.W  #CLR_HINT, &tempReg0       ; tempReg0 = bit value to load on HPIC
        MOV.W  #tempReg0, R6
        MOV.B  #HPI_CtrlReg, R5           ; R5=HPIC access bit orientation
        CALL   #HPI_WRITE                 ; write 0909h to HPIC
;    ======  clear P2IFG reg and set GIE  =======

        MOV.B  &P2IFG, R5          ; read int flag
        BIC.B  R5, &P2IFG          ; clear int flag
        EINT                       ; set GIE to enable Monitor and nested int


;    ======  load HPIA with address (063h) of var result   =======

        MOV.W  #63h, &tempReg0     ; tempReg0 = addr of result in DSP memory
        MOV.W  #tempReg0, R6
        MOV.   #HPI_AddrReg, R5    ; R5=HPIA access bit orientation
        CALL   #HPI_WRITE          ; write 0063h to HPIA
;    ======  read result from HPID   =======

        MOV.W  #result, R6         ; R6=address of result
        MOV.B1 #HPI_DatReg, R5     ; R5=HPID access bit orientation
        CALL   #HPI_READ           ; read result from DSP memory and store in result
        POP  R6                    ; restore R6
        POP  R5                    ; restore R5
        RETI                       ; return with interrupt enabled
;*************************************************************************
;    other interrupts
;*************************************************************************

Int_unused            ; return from interrupt if unused
        RETI


;*************************************************************************
;    Interrupt Vector Table
;*************************************************************************
                .sect     "Int_Vect",05E0h
                .word     Int_unused        ; P0.2-P0.7 Int
                .word     Int_unused        ; Basic Timer Int
                .word     Int_unused        ; I/O Port1 Int
                .word     HPI_ISR           ; I/O Port2 Int
                .word     Int_unused        ; Timer/Port Int
                .word     Int_unused        ; Reserved
                .word     Int_unused        ; UART Xmit Int
                .word     Int_unused        ; UART Rec Int
                .word     Int_unused        ; TimerA Time Int
                .word     Int_unused        ; TimerA CAP/CMP Int
                .word     Int_unused        ; WD Overflow Int
                .word     Int_unused        ; Reserved
                .word     Int_unused        ; P0.1 Int
                .word     Int_unused        ; P0.0 Int
                .word     RESET             ; NMI, Osc. fault
                .word     RESET             ; POR, ext. Reset, Watchdog
                .end
```

Appendix B   DSP Routines

```
**************************************************************************
; File Name: slv_5402.asm
;
; Target System: TMS320VC5402 EVM
;
; Description:
;
;   Code to initialize 5402 DSP runtime environment and
;   process data sent by the host microcontroller.  Host
;   writes data at DSP internal data memory 60h to 62h
;   and asserts DSPINT. Upon arrival of the interrupt the
;   DSP processes the data, stores the result at 63h and
;   interrupts the host to indicate the completion of data
;   processing
;
**************************************************************************


    .mmregs
    .def START, DSP_ISR
    .ref INIT5402

***************************************************************************
*  Assign variables to values sent by host
***************************************************************************

command  .set  0060h   ;Host command word
num1     .set  0061h   ;Host command word
num2     .set  0062h   ;Host command word
result   .set  0063h   ;Host command word


***************************************************************************
*   Code starts from here
***************************************************************************

    .text

START
    SSBX  INTM          ; mask interrupts
    STM   #280h, SP     ; initialize stack pointer
    CALL  INIT5402      ; initialize DSP runtime environment
    SSBX  OVM           ; overflow mode enabled
    SSBX  SXM           ; sign extension mode enabled
    STM   #0FFFFh, IFR  ; clear pending interrupts (if any)
    STM   #0200h, IMR   ; select host to DSP interrupt
    RSBX  INTM          ; enable maskable interrupts

WAIT:  NOP       ; wait for host interrupt
       NOP
       B  WAIT
```

```
;************************************************************************
; Interrupt Subroutine: DSP_ISR
;
; Description:
;
; This subroutine tests the host command and process data sent by host
; accordingly.  After processing the data DSP stores the result at 63h
; and interrupts the host for attention.  If invalid host command is
; detected the DSP loads 0x0bad in result.
;
;************************************************************************

DSP_ISR:

    STM  #command, AR2 ; pointer to host command
    STM  #num1, AR3    ; points @ 1st number
    STM  #num2, AR4    ; points @ 2nd number
    STM  #result, AR5  ; destination pointer


*************************************************************************
*   Bit test host command and process num1 and num2 accordingly
*************************************************************************

CHK_BIT0:

    BIT  *AR2, 15-0          ; test bit0 of host command
    BC   CHK_BIT1, NTC       ; if bit0=0 check bit1
    ADD  *AR3, *AR4, A       ; else add num1 and num2
    STH  A, *AR5             ; store result
    B    EXIT

CHK_BIT1:

    BIT  *AR2, 15-1          ; test bit0 of host command
    BC   CHK_BIT2, NTC       ; if bit1=0 check bit2
    SUB  *AR3, *AR4, A       ; else subtract num1 and num2
    STH  A, *AR5             ; store result
    B    EXIT

CHK_BIT2:
    BIT  *AR2, 15-2          ; test bit0 of host command
    BC   BAD_CMD, NTC        ; if bit2=0 then bad host command
    MPY  *AR3, *AR4, A       ; else multiply num1 and num2
    STL  A, *AR5             ; store result
    B    EXIT

BAD_CMD:

    STM  0BADh, *AR5     ; load 0x0bad in result to indicate
                         ; invalid host command received
```

TEXAS
INSTRUMENTS

```
***************************************************************************
*   DSP_ISR exit point
***************************************************************************


EXIT:

   STM  #08h, HPIC               ; Interrupt Host for attention
   STM #0FFFFh, IFR              ; clear interrupt flags
   RETED                         ; enable interrupt and return with delay
   NOP
   NOP



***************************************************************************
; File Name:  init5402.asm
;
; Target System: TMS320VC5402 EVM
;
; Description: Code to initialize 5402 DSP runtime environment,
;   configures PMST, BSCR and SWWSR to known states

***************************************************************************

                 .mmregs
                 .def     INIT5402


***************************************************************************
*   Register initialization values
***************************************************************************


PMST_VAL  .set  00E3h   ; Intrrupt vec on page 1, reset reset val
BSCR_VAL  .set  0800h   ; 64k mem bank, extra 1 cycle between
                        ; consecutive prog and data read.
SWWSR_VAL .set 2000h    ; I/O wait state 2 clock cycle
***************************************************************************
*   Start of initialization code
***************************************************************************
                 .text
INIT5402:

   STM  #PMST_VAL, PMST      ; Init Processor Mode Status Register
   STM  #BSCR_VAL, BSCR      ; Set Wait States for Bank Switching
;  STM  #SWWSR_VAL, SWWSR    ; Init S/W Wait State Register
                            ; (commented out: power-up default value used)

   STM #10h, TCR   ; Stop on-chip timer
   RETD     ; return from subroutine
   NOP
   NOP
```

```
;*********************************************************************
; File Name:  c5402vec.asm
;
; Target System: TMS320VC5402 EVM
;
; Description: TMS320VC5402 interrupt vector initialization table
;
;*********************************************************************

            .ref    START, DSP_ISR     ; subroutines defined in slv_5402.asm

            .sect  ".vectors"

RESET: BD START                         ; RESET vector
       NOP
       NOP
NMI:   BD NMI                           ; ~NMI
       NOP
       NOP


****************************************************************************
*   S/W Interrupts
****************************************************************************
SINT17 BD SINT17
       NOP
       NOP
SINT18 BD SINT18
       NOP
       NOP
SINT19 BD SINT19
       NOP
       NOP
SINT20 BD SINT20
       NOP
       NOP
SINT21 BD SINT21
       NOP
       NOP
SINT22 BD SINT22
       NOP
       NOP
SINT23 BD SINT23
       NOP
       NOP
SINT24 BD SINT24
       NOP
       NOP
SINT25 BD SINT25
       NOP
       NOP
SINT26 BD SINT26
       NOP
       NOP
```

```
SINT27 BD SINT27
        NOP
        NOP
SINT28 BD SINT28
        NOP
        NOP
SINT29 BD SINT29
        NOP
        NOP
SINT30 BD SINT30
        NOP
        NOP


*************************************************************************
*    Rest of the Interrupts
*************************************************************************
*
INT0:   BD INT0
        NOP
        NOP

INT1:   BD INT1
        NOP
        NOP

INT2:   BD INT2
        NOP
        NOP

TINT0:  BD TINT0
        NOP
        NOP
BRINT0: BD BRINT0
        NOP
        NOP

BXINT0: BD BXINT0
        NOP
        NOP

DMAC0:  BD DMAC0
        NOP
        NOP

TINT1:  BD TINT1
        NOP
        NOP

INT3:   BD INT3
        NOP
        NOP
```

```
HPINT:  BD DSP_ISR
        NOP
        NOP
BRINT1: BD BRINT1
        NOP
        NOP
BXINT1: BD BXINT1
        NOP
        NOP

DMAC4:  BD DMAC4
        NOP
        NOP

DMAC5:  BD DMAC5
        NOP
        NOP




/*****************************************************/
/*      Linker Command File            */
/*****************************************************/

C5402vec.obj
slv_5402.obj
init5402.obj

-o slv_5402.out
-m slv_5402.map

MEMORY
  {
     PAGE 0:                /* Pgm space */
      VECS  : origin = 00080h, length = 007fh    /* Vector */
      PROG  : origin = 01800h, length = 1000h  /* Ext.Pgm.area */

    PAGE 1:                /* Data space */
      RAM0  : origin = 0060h, length = 0020h
      RAM1  : origin = 0180h, length = 0680h
}

SECTIONS
{
   .vectors : {} > VECS    PAGE 0    /* Interupt Vector table */
   .text    : {} > PROG    PAGE 0    /* Program code goes here */
   .bss     : {} > RAM1    PAGE 1    /* uninitialized variables */
   .data    : {} > RAM1    PAGE 1
   stack    : {} > RAM1    PAGE 1    /* uninitialized section */
}
```

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE ("CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.