Interfacing TMS320C54x DSK-Plus with the TMS28F400

Literature Number: SPRA456
Texas Instruments Europe
June 1998

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.

Contents

List of Figures

List of Tables

Interfacing TMS320C54x DSK-Plus with the TMS28F400

ABSTRACT

This document describes a daughter board that can be plugged onto the TMS320C54x DSK-Plus. It is based on the TMS28F400 4 Megabit bit Flash memory. This application report includes the complete hardware description and the software algorithms that have been successfully tested on the configuration mentioned above.

The debug of this kit has been performed using the standard JTAG emulator XDS510 from Texas Instruments.

1. Hardware description

1.1 The DSK-Plus

The DSK-Plus board is based on the TMS320C542 which is part of the TMS320C54x family. This device is a very powerful 16-bit Fixed-Point machine targeted at middle and high performance applications.

It includes 10K words of internal RAM that can be shared between the DSP program and data memory spaces. The 2K words of on-chip ROM are masked during the production phase and contain :

- A bootloader program that boots from the serial ports, external memory, an I/O port or the host port interface
- A 256-word u-law table
- A 256-word A-law table
- A 256-word sine look-up table
- An interrupt vector table

Refer the TMS320C54x User's guide Volume 1 page 3-11 for more information on the contents of this ROM.

Since the TMS320C542 interface is 16-bit wide, it can address up to 64K*16 bit. Furthermore this device ouputs 3 different strobes ($\overline{PS}$, $\overline{DS}$ and $\overline{IS}$ signals) and it can therefore access a total of up to 3 Megabit of external memory.

A designer who wants to access more than 64K of external program memory can use the TMS320LC548, which is code compatible with the TMS320C542 and which has a 22-bit extended program memory interface.

In order to present a comparable solution using the external data memory, it has been decided to map the TMS28F400 into the DSP data memory. And because the DSP can only access up to 64K words (one each strobe), the external flash memory has been split into 4 different pages, individually selectable by the 2 MSBs A16 and A17 of the memory

device. The active page is then accessed through the standard 16-bit DSP data memory interface.

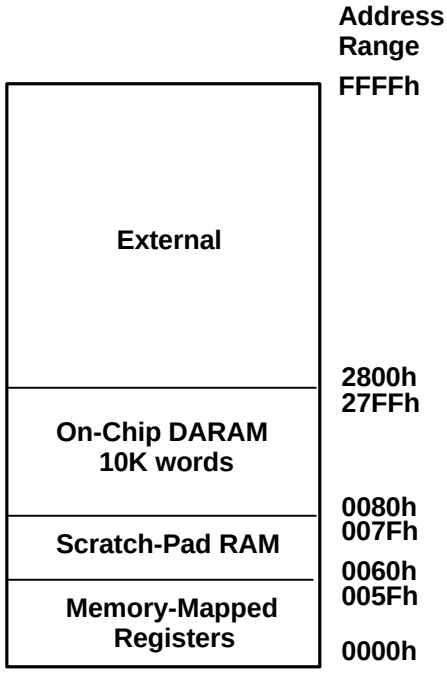


Figure 1: TMS320C542 Data Memory Map

1.2 The TMS28F400ASB

The TMS28F400ASB is a 4 194 304 bit, 256K words x 16, boot block flash memory that can be electrically block erased and reprogrammed. The TMS28F400ASB is segmented in one 8K words protected boot block, two parameter blocks, one 48K words main block, and three 64K words main blocks. The device is available in both top and bottom boot block configuration This application refers to the top boot block device. The boot block is therefore located at the address 0x00000 to 0x01FFFF in the chip memory map.

To address 256K words, 18 address lines (A0-A17) are required, compared to only the 16 address lines of the DSP (A0-A15). In one of the following chapters we shall describe the design that has been implemented to interface the two address buses.

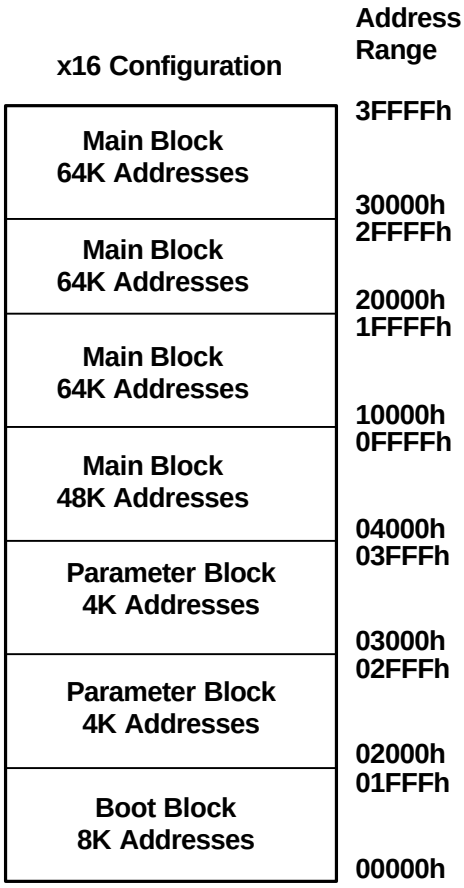


Figure 2: TMS28F400ASB Memory Map

This device needs a dual supply voltage (5 V & 12 V continuous) to be properly reprogrammed. The power supply delivered with the DSK-Plus is not suitable as it provides only a 5 V continuous output voltage. Another external supply kit has been used in order to provide the necessary voltages to the board, and especially the 12 V continuous voltage that must be connected to the Vpp programming pin (pin 1) of the TMS28F400.

1.3 Other Devices

To interface the TMS320C542 with the TMS28F400, four other TI devices have been used :

- One single inverter gate : SN74AHC1G04DBV
- One single 2 input positive OR gate : SN74AHC1G32DBV
- One 8-bit addressable latch register : SN74ALS259D
- A 16-bit buffer/driver with inverting outputs : SN74ABT16240DL

The single gates are used to generate the $\overline{E}$, $\overline{W}$ & $\overline{G}$ signals of the Flash with the following equations :

- $\overline{E}$ = $\overline{MSTRB}$ OR $\overline{DS}$ $\overline{E}$ is the chip select
- $\overline{W}$ = R/$\overline{W}$ $\overline{W}$ is the write enable
- $\overline{G}$ = $\overline{(R/\overline{W})}$ $\overline{G}$ is the output enable

The SN74ABT16240DL and the SN74ALS259D are used to implement the interface between the TMS320C542 and the TMS28F400 address buses. This will be described in detail in the next chapter.

1.4 Interface between the TMS320C542 and the TMS28F400 address bus

Whereas the TMS28F400 is a 256K words flash memory, the TMS320C542 can only address up to 64K words in data space. We thus need to implement a paging system. This is realized by means of an external 8-bit addressable latch register SN74ALS259D that generates the missing address signals for A16 and A17 and selects the active page.

Here is the corresponding schematic :

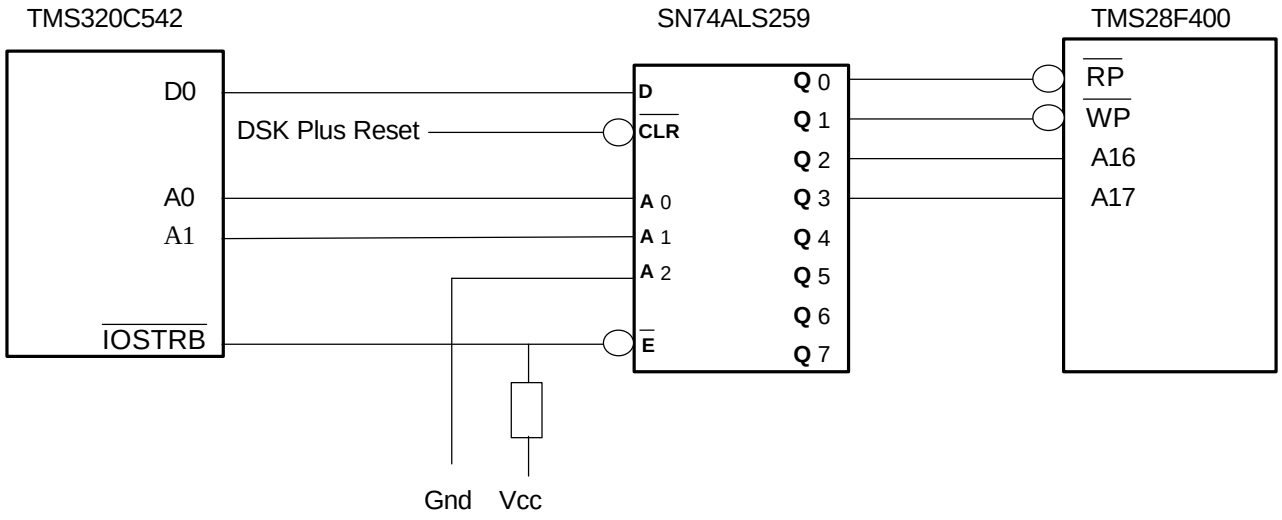


Figure 3: Flash memory Banks access description

This latch register is mapped in I/O space and driven by the $\overline{\text{IOSTRB}}$ signal. It is accessed with the PORTW assembly language instruction, which allows the programmer to write to the I/O page. The description of this instruction is as follows :

```
PORTW    Smem, PA
```

Two parameters are needed :

- Smem is the address of the data that you want to write to the I/O register
- PA is the port address that you want to access

More information about this instruction may be found in *TMS320C54x Mnemonic Instruction Set 1996* page 4-130.

Looking at the schematic of the latch register (see Figure 3 on the previous page), you can see that each port address will access one of the Flash control signals ($\overline{RP}$, $\overline{WP}$, A16 and A17). Table 1 below shows the PA address that needs to be activated in order to generate the corresponding flash signal :

Table 1: PA addressing

PA value	Flash control signal
00	$\overline{RP}$
01	$\overline{WP}$
02	A16
03	A17

The second important point is that the protected boot block of the TMS28F400ASB is located at the address 0x00000 to 0x01FFF within the first 64K words bank (A16=A17=0).

It could be very useful to keep this protected boot block to store the software routines used to program and erase the Flash memory in a safe area.

The TMS320C542 accesses its external memory from 0x2800 to 0xFFFF (See Figure 1 page 5). An easy way to map the flash boot block locations into the DSP external data space consists in inverting the whole DSP address bus. This explains the role of the 16-bit inverting buffer SN74ABT16240DL.

Please note that this device could have been avoided by using a TMS28F400AST on which the boot block is located at the top of the memory map.

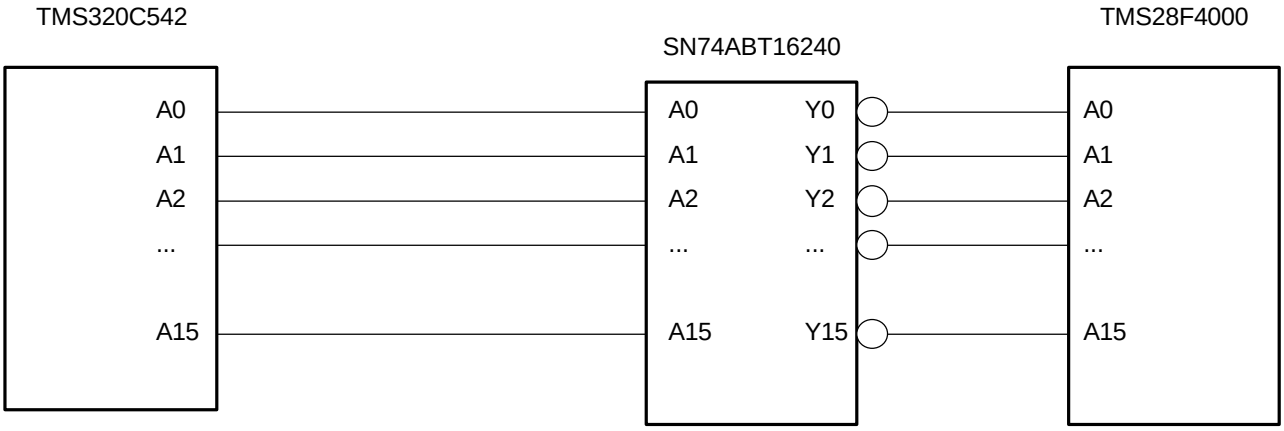


Figure 4: Position of the Flash within the DSP data memory map

1.5 Timings

The Flash device that has been used on the daughter board is the 70 ns version of the TMS28F400ASB. Its complete part number is therefore the TMS28F400ASB70BDBJL.

The most critical timing occurs on a read of the external Flash memory.

On a read, the first timing requirement is the access time T_a(MSTRBL) - read data access from $\overline{\text{MSTRB}}$ low-. The $\overline{\text{MSTRB}}$ signal is delayed by the SN74AHC1G32 for about 5.5 ns. This implies that the data to be read from the flash is stable 70+5.5=75.5 ns after the DSP $\overline{\text{MSTRB}}$ signal becomes active.

Since the TMS320C542 plugged onto the DSK-Plus is a 40 Mips device it runs with a cycle time of 25 ns. At that speed, the DSP reads the external data memory interface no later than 15 ns after the $\overline{\text{MSTRB}}$ signal becomes active. To meet the 75 ns delay needed to access the external flash device, the DSP needs to be slowed down on external read by 3 wait-states.

The second important timing requirement is the access time T_a(A)M - read data access from address valid. Since the SN74ABT16240 delays the address by less than 5 ns and since the T_a(A)M is equal to T_a (MSTRBL), it results in a similar constraint for this second access time requirement.

This validates the previous result: 3 wait-states need to be inserted to interface the TMS320C542-40 with the TMS28F400-70.

The above timing information can been found in the TMS28F400 and the TMS320C542 data sheets.

1.6 PCB Layout generic information

The interface which has been successfully implemented and tested is a four layer board arranged as follows:

- 1 Signal layer (device side)
- 1 Ground layer
- 1 Power layer
- 1 Signal layer (no device)

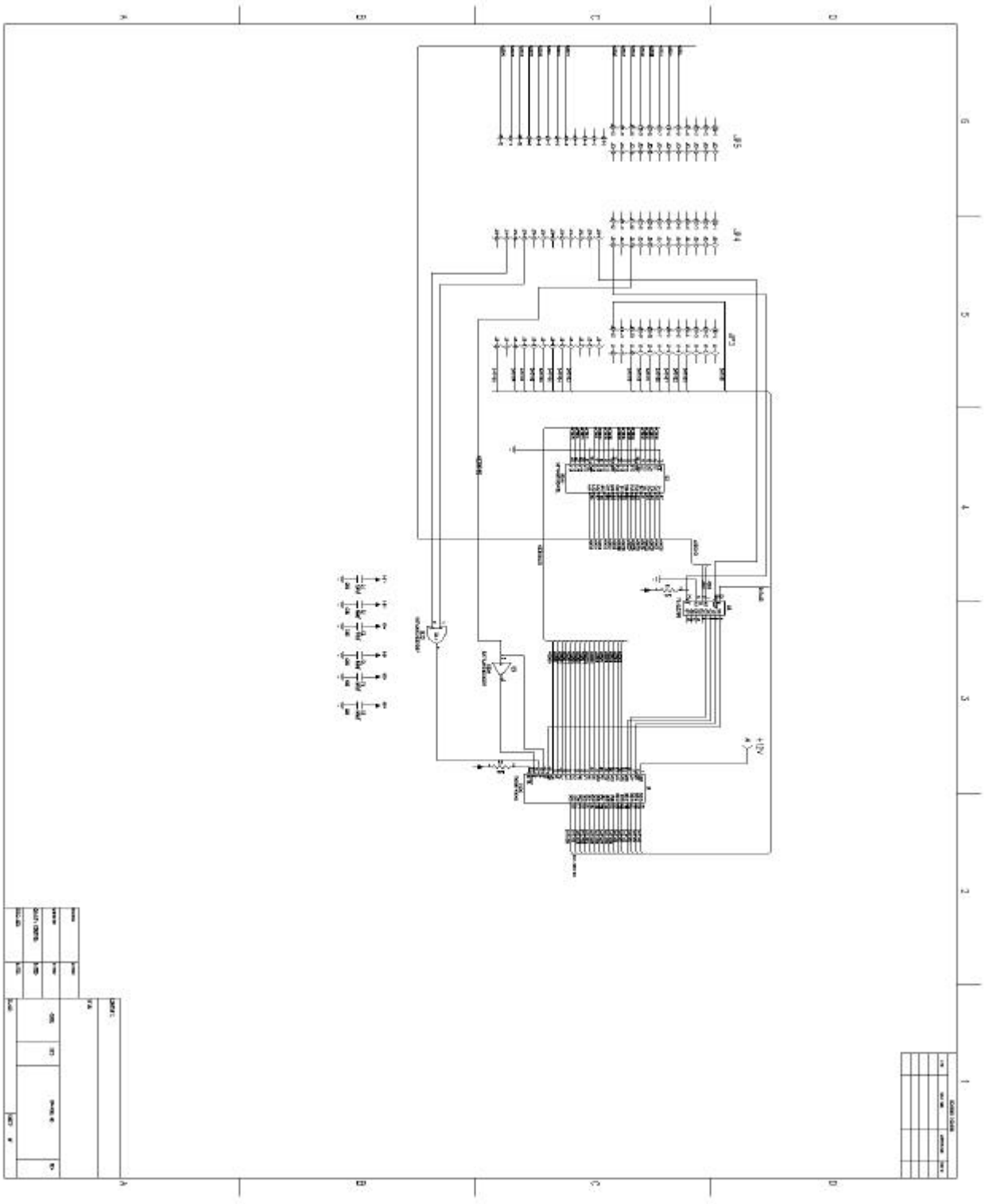
Six 100 nF decoupling capacitors have been soldered to the board as follows:

- 1 to the SN74ALS259
- 4 to the SN74ABT16240
- 1 to the TMS28F400

Finally two 4.7 KΩ resistors have been used to pull-up:

- the $\overline{\text{BYTE}}$ input of the TMS28F400
- and the $\overline{\text{E}}$ input of the SN74ALS259

1.7 Schematics of the board



2. Software Algorithms

2.1 Description of the main routines

In order to provide a complete software solution, 4 routines have been implemented:

- A load program and execute routine
- The program routine
- The erase routine
- A load data routine

The only difference between the first and the last routines is that the *load data routine* transfers data whereas the *load program routine* transfers and executes code.

In order to take as little memory space as possible, all these routines have been written in TMS320C54x mnemonic assembly language, but it would be very easy to call them from a C-program.

At startup the program assumes that the Host port interface has loaded at its base address in DSP RAM (0x1000) the command word that determines the action to be executed.

A bit test is performed, and depending on its result, the main program will branch to the corresponding routine.

Table 2: Main routines

Bit Command word at 0x1000	Action to perform
xxxx xxxx xxxx xxx1	Load program from the Flash and execute
xxxx xxxx xxxx xx1x	Program the Flash
xxxx xxxx xxxx x1xx	Erase the Flash
xxxx xxxx xxxx 1xxx	Load data from the Flash

2.2 Data Table organization in the DSP and external Flash memory

As soon as the main program branches to the routine that needs to be executed, it also expects to find other data that should have been loaded by the HPI right after the command word.

Here is a summary of all the data that should be found at 0x1000 in the DSP internal RAM, and in the TMS28F400.

- Load Program and Execute Routine

Data Words in the HPI RAM	Corresponding Hexadecimal value
Command Word	xxxx xxxx xxxx xxx1
A16 and A17 values	xxxx xxxx xxxx xxA_{16} A_{17}
Flash Offset in the bank defined above	Any from 0x0000 to 0xFFFF

And :

Data Words in the external Flash	Corresponding Hexadecimal value
DSP internal RAM run address	Any from 0x0000 to 0x2800
Program length	Any from 0x0000 to 0x7FFF
Program code	Opcode value

- Program Routine

Data Words in the HPI RAM	Corresponding Hexadecimal value
Command Word	xxxx xxxx xxxx xx1x
A16 and A17 values	xxxx xxxx xxxx xxA_{16} A_{17}
Flash Offset in the bank defined above	Any from 0x0000 to 0xFFFF
DSP internal RAM run address	Any from 0x0000 to 0x2800
Length	Any from 0x0000 to 0x7FFF
Code or Data	Any

- Erase Routine

Data Words in the HPI RAM	Corresponding Hexadecimal value
Command Word	xxxx xxxx xxxx x1xx
A16 and A17 values	xxxx xxxx xxxx xxA_{16} A_{17}
Flash Offset in the bank defined above	Any from 0x0000 to 0xFFFF

- Load Data Routine

Data Words in the HPI RAM	Corresponding Hexadecimal value
Command Word	xxxx xxxx xxxx 1xxx
A16 and A17 values	xxxx xxxx xxxx xxA_{16} A_{17}
Flash Offset in the bank defined above	Any from 0x0000 to 0xFFFF

And :

Data Words in the external Flash	Corresponding Hexadecimal value
DSP internal RAM run address	Any from 0x0000 to 0x2800
Data length	Any from 0x0000 to 0x7FFF
Data values	Any

2.3 Mnemonic assembly language source code

The following full software has been tested on the add-on board through the JTAG emulator XDS510 :

```
.title  "flash.asm"

        .mmregs
HPMEM   .set    1000h
SSTACK  .set    80h

        .global _c_int00

        .sect   "flashcom"
fwrite  .word   040h            ; write command
fboot   .word   000h            ; reset RP\ and WP\
ferase  .word   020h            ; erase command
fconf   .word   0D0h            ; confirm erase
setrp   .word   001h            ; set RP\
status  .word   0               ; buffer for flash status
buf     .word   0               ; 1 word data buffer

        .sect   "vectors"
reset   BD      _c_int00
        STM     #SSTACK,SP
        .space  24*4*16
hpint   BD      _c_int25
        NOP
```

```
        NOP


; pointer assignment:
; AR0 = index register
; AR1 = HPI memory pointer write
; AR2 = HPI memory pointer read
; AR3 = source pointer for flash transfers
; AR4 = destination pointer for flash transfers
; AR5 = pointer to internal data


        .text
_c_int00:
        LD      #SWWSR,DP       ; initialize DP
        STM     #76FFh,SWWSR    ; 3 Wait-states inserted to access data
        LD      #fwrite,DP      ; initialize DP
        STM     #fwrite, AR5    ; set pointer to internal data
        STM     #200h, IMR      ; enable HPI interrupt
        RSBX    INTM            ; enable global int. bit
        B       $               ; branch until HPI interrupt

_c_int25:

        STM     #HPMEM, AR1
        STM     #HPMEM, AR2
        STM     #6, AR0         ; offset to status
        BIT     *AR2,0Eh        ; test bit 1 of the address 0x1000
        BC      progm,TC        ; program flash mem

        BIT     *AR2,0Dh        ; test bit 2 of the address 0x1000
        BC      erase,TC        ; erase flash mem

        BIT     *AR2,0Ch        ; test bit 3 of the address 0x1000
        BC      ldata,TC        ; load flash data

        BIT     *AR2+,0Fh       ; test bit 0 of the address 0x1000
        NOP                     ; pipeline delay
        NOP                     ; pipeline delay
        XC      1, NTC          ; no valid command recognized

        RETE                    ; return to loop

;
; load program from flash to internal RAM and execute
;

        PORTW   *AR2, 2         ; set A16
        LD      *AR2+, -1, A    ; mask A17
        STL     A, buf
        PORTW   buf, 3          ; set A17
        LD      *AR2+,A         ; offset source address in flash
        STLM    A,AR3           ; initialize flash pointer
        PORTW   setrp,0h        ; Needed to access the flash in
```

```
                                ; read mode
        LD      *AR3+,B         ; internal RAM run address
        STLM    B,AR4           ; initialize RAM address pointer
        LD      *AR3+,A         ; length
        SUB     #1,A            ; get # of repetitions
        LD      #0,DP           ; initialize DP for loop count
        RPT     8               ; repeat as specified by AL
        MVDD    *AR3+,*AR4+     ; move flash to internal RAM
        ST      #1111h,*AR1     ; send confirmation to host
        ST      #8h, 02Ch       ; interrupt host
        CALA    B               ; branch to program
        RETE

;
; program flash memory
;

progm
        MAR     *AR2+           ; mandatory for A16 timings
        PORTW   *AR2, 2         ; set A16
        LD      *AR2+, -1, A    ; mask A17

        STL     A, buf
        PORTW   buf, 3          ; set A17
        LD      *AR2+,A         ; offset dest. address in flash
        STLM    A,AR4           ; initialize flash pointer
        PORTW   setrp,0h        ; only boot block locked

; write run address
        LD      *AR2+,B         ; internal RAM run address
        MVDD    *AR5,*AR4       ; send write command to flash
        STL     B,*AR4+         ; write run address into flash
        MAR     *AR5+0          ; update pointer
        ST      #050h,*AR4      ; clear status register
        ST      #70h, *AR4      ; select status reg read mode
L1      MVDD    *AR4, *AR5      ; get status
        BIT     *AR5, 08h       ; wait while busy
        BC      L1,NTC
        BIT     *AR5, 0Bh       ; test write successful
        MAR     *AR5-0          ; update pointer
        BC      error, TC

; write length
        LD      *AR2+,B         ; length
        MVDD    *AR5,*AR4       ; send write command to flash
        STL     B,*AR4+         ; write length into flash
        MAR     *AR5+0          ; update pointer
        ST      #70h, *AR4      ; select status reg read mode
L2      MVDD    *AR4, *AR5      ; get status
        BIT     *AR5, 08h       ; wait while busy
        BC      L2,NTC
        BIT     *AR5, 0Bh       ; test write successful
        MAR     *AR5-0          ; update pointer
        BC      error, TC
```

```
; main loop

L7      MVDD    *AR5,*AR4       ; send write command to flash
        MVDD    *AR2+,*AR4+     ; send data to flash
        MAR     *AR5+0          ; update pointer
        ST      #70h, *AR4      ; select status reg read mode
L3      MVDD    *AR4, *AR5      ; get register value
        BIT     *AR5, 8         ; test bit SB7
        BC      L3,NTC          ; wait if SB7 not set
        BIT     *AR5, #0Bh      ; test write successful
        MAR     *AR5-0          ; update pointer
        BC      error, TC
        SUB     #1, B           ; decrement counter
        BC      L7,BNEQ

        ST      #050h,*AR4      ; clear status register

; return
        ST      #1111h, *AR1    ; send confirmation to host
        B       L4
error   ST      #0FFFFh,*AR1    ; send error condition

L4      LD      #0,DP           ; initialize DP to access HPIC
        ST      #8h, 02Ch       ; interrupt host
        RETE


;
; erase one block
;

erase
        MAR     *AR2+
        PORTW   *AR2, 2         ; set A16
        LD      *AR2+, -1, A    ; mask A17
        STL     A, buf
        PORTW   buf, 3          ; set A17
        LD      *AR2+,A         ; offset dest. address in flash
        STLM    A,AR4           ; initialize flash pointer
        PORTW   setrp,0h        ; only boot block locked
        ST      #050h,*AR4      ; clear status register
        ST      #020h,*AR4      ; erase command
        ST      #0D0h,*AR4      ; erase confirm
; erase control loop
        MAR     *AR5+0          ; update pointer
        ST      #70h, *AR4      ; select status reg read mode
L5      MVDD    *AR4, *AR5      ; get register value
        BIT     *AR5, 8         ; test bit SB7
        BC      L5,NTC          ; wait if SB7 not set
        BIT     *AR5, #0Ah      ; test erase successful
        BC      error1,TC

        MAR     *AR5-0
```

```
        ST      #050h,*AR4      ; clear status register

; return
        ST      #1111h, *AR1    ; send confirmation to host
        B       L6
error1
        ST      #0FFFFh, *AR1   ; send error condition to host

L6      LD      #0,DP           ; initialize DP to access HPIC
        ST      #8h, 02Ch       ; interrupt host
        RETE


;
; load data from flash to internal RAM
;

ldata
        MAR     *AR2+
        PORTW   *AR2, 2         ; set A16
        LD      *AR2+, -1, A    ; mask A17
        STL     A, buf
        PORTW   buf, 3          ; set A17
        LD      *AR2+,A         ; offset source address in flash
        STLM    A,AR3           ; initialize flash pointer
        PORTW   setrp,0h        ; Needed to access the flash in
                                ; read mode
        LD      *AR3+,B         ; internal RAM run address
        STLM    B,AR4           ; initialize RAM address pointer
        LD      *AR3+,A         ; length
        SUB     #1,A            ; get # of repetitions
        LD      #0,DP           ; initialize DP for loop count
        RPT     8               ; repeat as specified by AL
        MVDD    *AR3+,*AR4+     ; move flash to internal RAM

; return
        ST      #1111h, *AR1    ; send confirmation to host

        ST      #8h, 02Ch       ; interrupt host - DP already at 0
        RETE

        .end
```

2.4 Algebraic assembly language source code

Since the DSK-Plus is delivered with only the TMS320C54x algebraic assembler, the DSK specific source code is given below. Note that all the references to the host port interface have been deleted from the software because the HPI is already used by the Go-DSP Code explorer HLL-Debugger.

```
.title  "flash.asm"

        .mmregs
HPMEM   .set    100Ah
SSTACK  .set    80h

        .setsect".text",        0x0300, 0
        .setsect "vectors",     0x0180, 0
        .setsect "flashcom",    0x0600, 1
        .setsect "test",        0x100A, 1


        .copy   "config.asm"


        .sect   "flashcom"
fwrite  .word   040h            ; write command
fboot   .word   000h            ; reset RP\ and WP\
ferase  .word   020h            ; erase command
fconf   .word   0D0h            ; confirm erase
setrp   .word   001h            ; set RP\
status  .word   0               ; buffer for flash status
buf     .word   0               ; 1 word data buffer




        .sect   "vectors"
reset   dgoto   _c_int00
        SP = #SSTACK

; pointer assignment:
; AR0 = index register
; AR1 = HPI memory pointer write
; AR2 = HPI memory pointer read
; AR3 = source pointer for flash transfers
; AR4 = destination pointer for flash transfers
; AR5 = pointer to internal data

        .text
_c_int00:
        DP = #SWWSR             ; initialize DP
        SWWSR = #76FFh          ; 3 Wait-states inserted to access
                                ; data space
```

```
        DP = #fwrite            ; initialize DP
        AR5 = #fwrite           ; set pointer to internal data

loop:
        AR1 = #HPMEM
        AR2 = #HPMEM
        AR0 = #6                ; offset to status
        TC = bit(*AR2,0Eh)      ; test bit 1 of the address 0x1000
        if (TC) goto progm      ; program flash mem

        TC = bit(*AR2,0Dh)      ; test bit 2 of the address 0x1000
        if (TC) goto erase      ; erase flash mem

        TC = bit(*AR2,0Ch)      ; test bit 3 of the address 0x1000
        if (TC) goto ldata      ; load flash data

        TC = bit(*AR2+,0Fh)     ; test bit 0 of the address 0x1000
        nop                     ; pipeline delay
        nop                     ; pipeline delay
        if (NTC) execute (2)    ; no valid command recognized
        goto loop               ; return to loop

;
; load program from flash to internal RAM and execute
;
        port(2) = *AR2          ; set A16
        A = *AR2+ << (-1)       ; mask A17
        @buf = A
        port(3) = @buf          ; set A17

        A = *AR2+               ; offset source address in flash
        AR3 = A                 ; initialize flash pointer
        port(0h) = @setrp       ; Needed to access the flash in
                                ; read mode
        B = *AR3+               ; internal RAM run address
        AR4 = B                 ; initialize RAM address pointer
        A = *AR3+               ; length
        A = A - #1              ; get # of repetitions
        DP = #0                 ; initialize DP for loop count
        repeat(@8)              ; repeat as specified by AL
        *AR4+ = *AR3+           ; move flash to internal RAM
        *AR1 = #1111h           ; send confirmation to host
        call    B               ; branch to program
        goto    $               ; useless instruction, just written
                                ; to avoid crash on the Code explorer
;
; program flash memory
;

progm
        mar(*AR2+)              ; mandatory for A16 timings
        port(2) = *AR2          ; set A16
        A = *AR2+ << (-1)       ; mask A17
        @buf = A
```

```
        port(3) = @buf          ; set A17
        A = *AR2+               ; offset dest. address in flash
        AR4 = A                 ; initialize flash pointer
        port(0h) = @setrp       ; only boot block locked

; write run address
        B = *AR2+               ; internal RAM run address
        *AR4 = *AR5             ; send write command to flash
        *AR4+ = B               ; write run address into flash
        mar(*AR5+0)             ; update pointer

        *AR4 = #050h            ; clear status register
        *AR4 = #70h             ; select status reg read mode
L1      *AR5 = *AR4             ; get status
        TC = bit(*AR5,08h)      ; wait while busy
        if (NTC) goto L1
        TC = bit(*AR5,0Bh)      ; test write successful
        mar(*AR5-0)             ; update pointer
        if (TC) goto error

; write length
        B = *AR2+               ; length
        *AR4 = *AR5             ; send write command to flash
        *AR4+ = B               ; write length into flash
        mar(*AR5+0)             ; update pointer
        *AR4 = #70h             ; select status reg read mode
L2      *AR5 = *AR4             ; get status
        TC = bit(*AR5,08h)      ; wait while busy
        if (NTC) goto L2
        TC = bit(*AR5,0Bh)      ; test write successful
        mar(*AR5-0)             ; update pointer
        if (TC) goto error

; main loop
L7      *AR4 = *AR5             ; send write command to flash
        *AR4+ = *AR2+           ; send data to flash
        mar(*AR5+0)             ; update pointer
        *AR4 = #70h             ; select status reg read mode
L3      *AR5 = *AR4             ; get register value
        TC = bit(*AR5,8)        ; test bit SB7
        if (NTC) goto L3        ; wait if SB7 not set
        TC = bit(*AR5,#0Bh)     ; test write successful
        mar(*AR5-0)             ; update pointer
        if (TC) goto error
        B = B - #1              ; decrement counter
        if (BNEQ) goto L7

        *AR4 = #050h            ; clear status register

; return
        *AR1 = #1111h           ; send confirmation to host
        goto    L4
error   *AR1 = #0FFFFh          ; send error condition
```

```
L4      goto    $

;
; erase one block
;

erase
        mar(*AR2+)
        port(2) = *AR2          ; set A16
        A = *AR2+ << (-1)       ; mask A17
        @buf = A
        port(3) = @buf          ; set A17
        A = *AR2+               ; offset dest. address in flash
        AR4 = A                 ; initialize flash pointer
        port(0h) = @setrp       ; only boot block locked
        *AR4 = #050h            ; clear status register
        *AR4 = #020h            ; erase command
        *AR4 = #0D0h            ; erase confirm
; erase control loop
        mar(*AR5+0)             ; update pointer
        *AR4 = #70h             ; select status reg read mode
L5      *AR5 = *AR4             ; get register value
        TC = bit(*AR5,8)        ; test bit SB7
        if (NTC) goto L5        ; wait if SB7 not set
        TC = bit(*AR5,#0Ah)     ; test erase successful
        if (TC) goto error1

        mar(*AR5-0)
        *AR4 = #050h            ; clear status register

; return
        *AR1 = #1111h           ; send confirmation to host
        goto L6
error1  *AR1 = #0FFFFh          ; send error condition to host

L6      goto    $

;
; load data from flash to internal RAM
;

ldata
        mar(*AR2+)
        port(2) = *AR2          ; set A16
        A = *AR2+ << (-1)       ; mask A17
        @buf = A
        port(3) = @buf          ; set A17
        A = *AR2+               ; offset source address in flash
        AR3 = A                 ; initialize flash pointer
        port(0h) = @setrp       ; Needed to access the flash in
                                ; read mode
        B = *AR3+               ; internal RAM run address
        AR4 = B                 ; initialize RAM address pointer
        A = *AR3+               ; length
```

```
        A = A - #1              ; get # of repetitions
        DP = #0                 ; initialize DP for loop count
        repeat(@8)              ; repeat as specified by AL
        *AR4+ = *AR3+           ; move flash to internal RAM

; return
        *AR1 = #1111h           ; send confirmation to host

        goto    $
        .end
```

To test the four flash routines described above, another configuration file has been used. This file is called “config.asm”.

The “config.asm” file that has been used to test the programming routine is shown below:

```
.sect   "test"
        .word   0002h         ; test program
        .word   0003h         ; A16=1 A17=1
        .word   0D000h        ; start write address 0x2FFF of the Flash mem
        .word   2000h         ; DSP RAM run address
        .word   0005h         ; section length
        .word   0010h         ; data
        .word   0020h
        .word   0030h
        .word   0040h
        .word   0050h
```

3. Possible future enhancements

Two types of improvements could be added to the design described in this document.

On the hardware side, the main change would be to use a TMS28F400AST instead of the TMS28F400ASB. This would provide a top boot block organization that would allow the designer to remove the SN74ABT16240. It would therefore save cost and space on the daughter board. It would also be possible to map the external Flash memory into program space in order to transform the DSK-Plus into a stand-alone board. This would require some changes at the software level as well.

On the software side, one possibility would be to make the above assembly language code C-callable. This would add the flexibility of the C language software development environment.

4. Conclusion

This paper presents to the designer the main routines needed to control an external Flash memory TMS28F400 from a TMS320C542 DSP controller. Since all the members of the TMS320C54x family are code compatible, such software could be used without any modification by the following devices :

- TMS320C/LC541
- TMS320C/LC542
- TMS320LC543
- TMS320LC545/LC545A
- TMS320LC546/LC546A
- TMS320LC548
- TMS320LC/VC549

Furthermore, because of the simplicity of the TMS320C54x assembly language, it would be extremely straightforward to translate the above code to any other Texas Instruments DSP controller assembly language.

References

1. TMS320C54x, TMS320LC54x Fixed-Point Digital Signal Processors Data-Sheet, July 1997
2. TMS28F004Axy, TMS28F400Axy 4194 304 bit Auto-select Boot-block Flash memories Data-Sheet, August 1996
3. TMS320C54x DSP CPU and Peripherals Reference Set Volume 1, 1997
4. TMS320C54x DSP Mnemonic Instruction Set Volume 2, 1996
5. Advanced BiCMOS ABT Bus Interface Logic data book, 1994
6. AHC/AHCT, HC/HCT and LV CMOS logic data book, 1996