TEXAS INSTRUMENTS

PCMCIA TMS320 DSP MediaCard

Application Report

1996 Digital Signal Processing Products

Texas Instruments

Printed in U.S.A., March 1996

SPRA052

PCMCIA TMS320 DSP MediaCard Card Application Note

Mathew George, Jr. (Joe)
Semiconductor Group

SPRA052
March 1996

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software 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, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

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

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI 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.

Introduction

A complete manufacturing package is available from Texas Instruments that describes the information in this application note in detail, including electronic schematics on request. The manufacturing document consists of three items: the specification document [7], a board-level description document [8], and an FPGA-level document [9].

Historically, our card was defined as a board level prototype of the "Basara" concept. A description of this concept is Texas Instruments classified information [3]. In order to verify this concept, a prototype was specified. This prototype was code named SABRE1. SABRE1 is a generic coprocessor with a standardized, memory-like interface and can serve as both a computing device and storage device. This evolved to the next version called "Texblast", a large development board with a separate daughter card for a codec. This later evolved to the current DSP MediaCard version 1.0, with a DSP and on-board stereo codec for sound and fax/modem applications.

DSP MediaCards can also be accessed by the PC as an I/O mapped device. These cards with their on board DSP can be used either as standard memory or as multi-function peripheral cards. DSP algorithms can be loaded by the host PC in the same manner that as it loads to any PCMCIA memory. Once the program is loaded, the host may command the DSP to execute the algorithm as a co-processor. Two functional models were concentrated upon for the DSP interface: the lower-rate single-data model as seen in a modem Host/DSP/Communication System and the higher-rate block-data transfers as seen in a sound card Bus Arbitration system.

To date TI had developed a form-factor showcase card called SABRE1 and a larger development card called Texblast. Both of these cards use the Texas Instruments TMS320C51 DSP in 100-pin TQFP which is a fixed-point 16-bit device running in this design at a maximum rate of 40 MIPs. A brief summary of the PCMCIA bus can be found in a vast variety of articles including [4]. Meanwhile, specific information regarding the Texblast board can be found in Appendix A of [7] for your reference. The final stage will be called the Media Card.

The following features were designed into these cards:

- Card is mapped addressable as host PC memory or I/O device through PCMCIA card services with medium speed (around 200−500 ns) access time of PCMCIA common memory (read/write) when in standard mode.
- Smart mode operation for the PC to control DSP directly and share common memory with DSP
- The same memory is available to the DSP as a maximum speed (25 ns instruction cycle time), zero wait state, DSP external memory configuration with expandibility designed in.
- Separate attribute memory to the PC per PCMCIA spec (slower 400−500-ns speed, read-only, even-byte access), also available to the DSP in its global data memory
- On-board logic to arbitrate-memory bus between DSP and PC with software programmable features
- Page control of DSP memory from DSP to increase its address space
- Direct PC control of active DSP tasks and memory pages
- PC control of DSP operating clock modes and DSP control of DSP speeds for power management
- Bootload of code to the DSP from its global-data memory under PC control
- Dual-serial I/F and emulation pins to analog front end (AFE) boards.

- Four general-purpose DSP inputs and eight outputs (bit I/O) to AFE boards
- Direct interrupt control and handshake between PC and DSP for PC/DSP communication
- DSP interrupts from AFE board
- PC interrupt from AFE cards to monitor external events
- Dedicated memory for real-time DSP operating systems with high-level language-based interface
- Software interface based on PCMCIA socket and card services using direct mapping of PCMCIA card into PC memory (under development)

System Architecture

The DSP MediaCard interfaces to the PC either as PCMCIA memory or I/O-mapped peripheral. This translates to the DSP as 32M (16 bit) words of external program/data space and 32M (8 bit) bytes of global-data space, both of which must obviously be paged in a 16 bit DSP. The specific memory configuration of the Texblast card is described in [7]. The minimum common memory to the DSP is set at 64K bytes to ensure that at least two 32K word pages of external program/data memory is available to the 'C5x DSP. There is maximum flexibility in a combined program data space/separate program and data space (CPD/$\overline{\text{SPD}}$) paging feature that allows application adaptation in DSP memory configuration. The entire common memory is accessible by PC in byte mode (8 bits) or in word (16 bits) mode. The entire attribute memory is also addressable by PC in byte mode with even bytes being valid as per PCMCIA specification [1]. The minimum amount of attribute memory available to the DSP as global-data memory is 32K bytes. The host PC can also access the card as a 16-bit I/O device. The I/O address for the card is selectable by PC in the card-configuration registers mentioned in 4.2, specified in [1], and detailed in [7].

Figure 1 shows a simple block diagram of an example system to visualize a card structure. The PCMCIA connector appears on left side of the board. The FPGA/ASIC integrates all parts of the system; it is the system traffic cop and more detail regarding its implementation can be seen in Section 7. There is an address/data/control signal bus on the left side of the FPGA/ASIC that connects to the PCMCIA bus. On the right side of the FPGA/ASIC is a second bus that connects to the memories, DSP, and any other peripherals that can be chosen to be placed there (such as a parallel stereo codec [12]). There is an analog front-end (AFE) board connector that has been defined and placed on the other side of the board for external serial and slow I/O interface such as modem or scanner port. An oscillator supplies clocks to DSP and FPGA/ASIC.

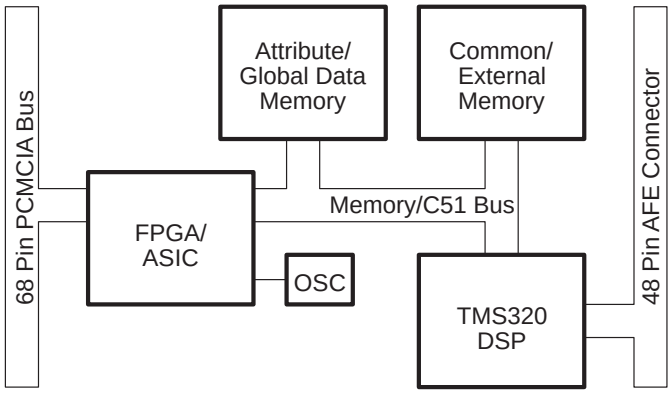

Figure 1. System Architecture Block Diagram

Part or all of the common memory can be shared by the on-board DSP for algorithm execution. This memory has been optimized first for speed (25-ns instruction cycle when using a single bank) and next for expandability for the DSP. Upon reset, the entire common memory is available to the PC. The host may download any signal-processing algorithm into the common memory for execution by the on-board DSP as explained in the software interface section. DSP accesses the memory as 16-bit words only and can use a paging scheme to expand its address reach. The paging is under software control of the DSP. There are also control and status registers used by both PC and DSP that are implemented on-board in a separate FPGA/ASIC device. The read-only attribute memory similarly is shared by the PC and DSP, though the read/write card configuration registers (CCRs) as defined by the PCMCIA spec are also located in the FPGA/ASIC. The DSPs internal memory is not available for the PC (especially on the 100 pin TQFP packages that do not have the $\overline{\text{IAQ}}$ signal required for accessing 'C5x SARAM. See [6] for a possible workaround, even though this is not supported in this application design).

The arbitration of DSP and PC buses are performed on-board with PC access of the memory having higher priority. There are a variety of configurations to maximize shared-memory efficiency, depending on the application. There is also a communication channel between PC and DSP that is accomplished by way of dedicated data, status, and control registers in the FPGA/ASIC and can be either interrupt or poll driven. The optimum configuration for a system is heavily dependent on the DSP and somewhat on the PC software running on the system. Alternate configurations are provided in this interface to give the programmer alternatives based on their code. The PCMCIA spec there is only one interrupt line $\overline{\text{IREQ}}$. Therefore, a status register (DSPSR) must be polled on the host side to determine which interrupt is being registered.

The system clock to the DSP is provided by the FPGA/ASIC to allow control of DSPs operating speeds in two modes. These modes must be set while the DSP is in reset. In the default mode, the DSP can run at a fixed 25-ns instruction cycle time. In the second mode, the DSP clock input frequency is at 50 ns and can be reduced under DSP software control to reduce power for slow speed operation (such as external event monitoring). This spec defines a divide by 1, 2, 4, and 8 that corresponds to 50 ns, 100 ns, 200 ns, and 400 ns. All these numbers assume the use of a 40-MIP device. In the standard mode of operation, the on-board DSP is reset with clocks shutoff to minimize power consumption.

As previously mentioned, the card provides a separate 48-pin connector for the DSP to interface to analog front-end cards (See Appendix B.2 of [7]). The DSP serial-port signals are available at this connector along with the two 'C5x bit I/O signals XF and $\overline{\text{BIO}}$. Also, eight outputs bit and four input bits are available on this connector to monitor, configure, and control external A/Ds, D/As, and other peripherals. These are controlled by the DSP using a register in its I/O space. Two external interrupts, and the DSP timer output are provided. $\overline{\text{INT4}}$ is combined with the AFE interrupt request signal. In other words, this signal goes to the PC via the FPGA so that it can be recognized even if the DSP is in reset. The connector also provides DSP-emulation control pins to help DSP algorithm development on the card. Thus a bare-boned AFE card would require and XDS the emulation header configuration for connection to a TI XDS-510 emulation system. Any pull-up resistors necessary for operation or to reduce power on inputs must be mounted on the DSP MediaCard. Figure 2 provides more detail about the board-level wiring. See the schematic in [8] for actual configuration.

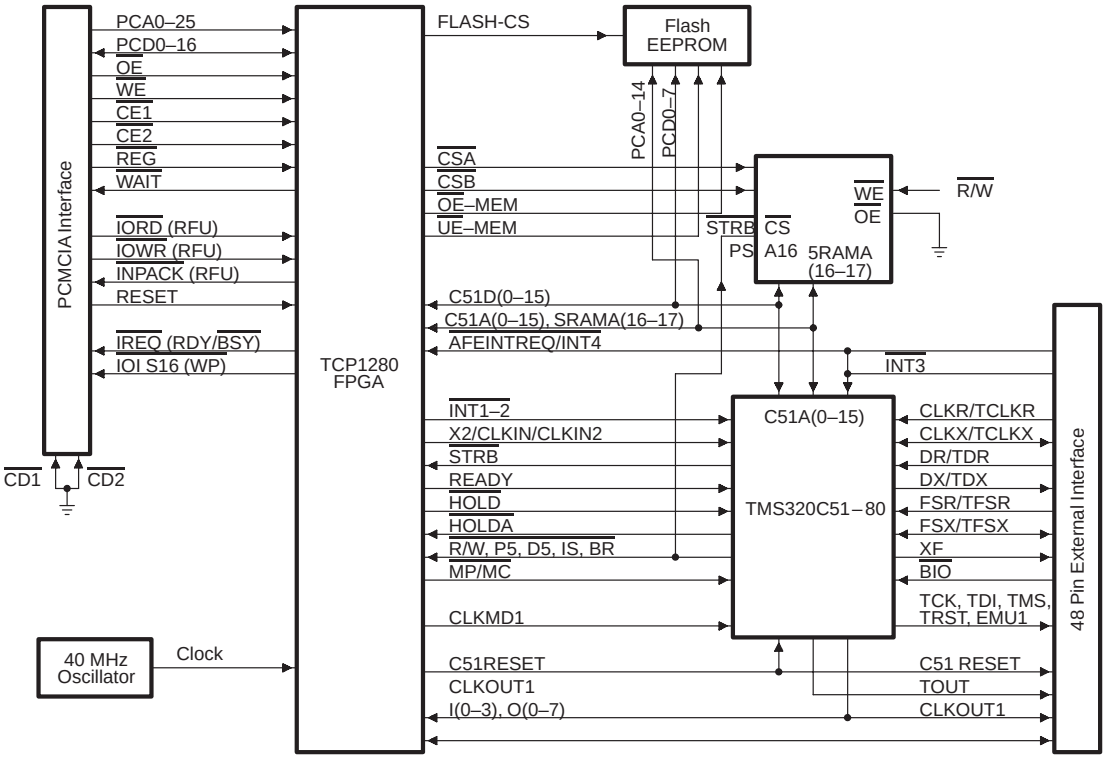

Figure 2. System Board Level Block Diagram

Operation

Various aspects of the operation of the PCMCIA card should be understood that roughly coincide with the Type I and Type II modes described in [1].

Operational Modes

According to PCMCIA specifications when the card is plugged into a slot, the card information structure (CIS) is read out of attribute memory and informs the PCMCIA slot controller how to configure this card.

The DSP MediaCard operates in two modes.:

- Standard mode
- Smart mode

Standard Mode

In the standard mode the DSP is reset, clocks are off, and the DSP is off the bus completely (i.e. $\overline{\text{HOLDA}}$ signal is asserted low) giving the PC low-power optimized access to the memories. This is the default mode that occurs when the card is plugged in and is intended for the memory only mode of PCMCIA [1]. In this mode, the card appears to the PC and is used by the PC as a standard-memory card only. When activated by the host PC, the on-board DSP is not active. The size of the memory available on the card is specified by the card information structure (CIS) in the attribute memory. The card configuration registers in attribute memory are available in this mode. See PCMCIA standard rev 2.1 for details of this mode of operation.

Smart Mode

In the smart mode, the DSP can be activated and share memory with the PC. In this mode, smart mode registers become active and available to PC and DSP. The first 16 bytes of common memory are re-mapped into physical locations in the ASIC for the PC (PC smart mode registers or PCSMMRs). PCSMMRs are DSP-control and -status registers that become available to the PC. During activation or after activation DSP smart mode memory registers (or DSPSMMRs or 'C51SMMRs) which are control and status registers to the DSP become available to the DSP as eight words in DSP I/O space. Communication between DSP and PC can be accomplished ? a set of other SMMRs called host-communication registers (DSPTXD, DSPRXD) as either interrupt or polling driven. When PC accesses PCSMMRs, the DSP operation is not halted since no bus arbitration is needed during these accesses. Depending on the application there are also other bus-arbitration optimizing features that can be used. This mode complies with the PCMCIA memory and I/O card mode of the PCMCIA spec [1].

Switching Between Standard Mode and Smart Mode

When the PC writes a DSP signature pattern A320 (Attention 320), four times consecutively to the SIGR, the DSP is activated and card is switched from the standard mode to the smart mode.

A320 is valid X86 data and can occur as part of the normal PC operation. For the signature to be valid, PC must write the same pattern to the same location in succession without any other reads or writes to any other location in the PCMCIA card. Address 400h functions as normal common memory location, although it is monitored.

Once the first signature code is detected, any of the following actions by the PC (before the signature stream is completed), disqualifies the stream from being valid:

- Any read from any valid address on the card
- Any write to any address (other than the signature address) on the card
- A write to signature address with another pattern.

Once a valid signature is detected, the MODCTL bit is set in the DSP control register. The card is switched from standard mode to smart mode and the first 16 bytes of the PC's common memory is re-mapped into the ASIC. Any subsequent PC access to this block of common memory is re-directed into the ASIC. To switch out of smart mode, a 0 should be written to the MODCTL bit.

An alternate method of switching in and out of smart mode uses the mode bit in the CCSR which is located in attribute memory. Writing a 1 causes the card to switch into smart mode while a 0 switches back to standard. These two methods are ORed together, so either one active in smart mode supersedes the other in standard mode. Note that a good method of testing whether the card is in smart mode is reading a PCSMMR and checking for default values as shown in Figure 3.

Smart Mode Registers

Control of the card's operating mode is accomplished via the control, status, and communication registers in the FPGA/ ASIC. Some of these are accessible only by the PC, some only by the DSP, and some by both PC and DSP. These registers are mapped in the common-memory space of the PC and I/O space of the DSP. For both PC and DSP, some of these registers are reserved as marked. In the standard mode, only the signature register (SIGR) is accessible to the PC. Other registers exist only when the card is in the smart mode of operation. The communications registers are dual mapped into the PC's I/O port also when I/O is enabled.

On the PC, these registers are mapped into the first 16 bytes of common memory (000000h – 00000Fh) address of the PC. This means that the PC will not change accidently the first few DSP interrupt vectors

when the card is in smart mode and separate program and data space mode. However, avoiding overwrites of these vectors must be maintained carefully for any other mode. In the smart mode, this block of common memory is not available to the PC for shared-memory access. Even though the PCSMMs are evenly byte addressed, they are 2-bytes long and should be accessed in PCMCIA word mode. When the PC accesses the communication, control, and status registers, it is accessing the registers physically located in the ASIC and not in the common memory.

The registers implemented in the FPGA/ASIC are as follows:

Register	Memory Address Access		Default		
Name	PCMCIA Common Memory	'C5x I/O Space	Value	PC	DSP
SIGR	000400h	—	xxxx	R/W	—
Reserved	000000h	—	—	—	—
DSPCR	000002h	—	0108h	R/W	—
DSPSR	000004h	—	0xxxh	R	—
DSPTXD	000006h	0050h	0000h	R	W
DSPRXD	000008h	0051h	0000h	W	R
Reserved	00000Ah–00000Fh	—	—	—	—
PCSR	—	0052h	000xh	—	R
BIOR	—	0053h	000xh	—	R/W||R
SYSCFG	—	0054h	0080h	—	R/W
Reserved	—	0055h–005Fh	—		

Figure 3. Smart Registers

In the standard mode, only the SIGR register is defined and the card monitors all PC writes to this location for valid signature pattern (while it writes to common memory). When the card operates in smart mode, SIGR is not monitored and any write to SIGR writes to that location in common memory.

The following describes each register briefly more detail can be found in [7].

DSP Control Register (DSPCR):

The DSPCR is a 16-bit R/W register accessible only to the PC. It is located at common-memory address 00000002h when the card is in smart mode. This register is used to control DSP operation (such as reset, clock modes, clock gating, and the MP/$\overline{\text{MC}}$ bit). It also enable/disables card interrupt (i.e., $\overline{\text{IREQ}}$ signal to the PC, memory-bus arbitration features, standard-smart mode switching, and $\overline{\text{IREQ}}$ interrupt clear and prime. Even though some protections have been included with the DSP-specific bits, the operation is not fully automated. Thus the user must program DSPCR while taking into consideration DSP operation. For example when individual writes should be done for turning on the DSP clock, taking the DSP out of reset, etc.

DSP Status Register (DSPSR):

The DSPSR is a 16-bit register read-only register accessible only to the PC. It is located at common-memory address 00000004h when the card is in smart mode. This register is used to monitor card-interrupt event status for the PC-interrupt $\overline{\text{IREQ}}$ [1] and various memory/DSP bus arbitration signals such as $\overline{\text{HOLDA}}$ [5].

DSP Data Transmit Register (DSPTXD):

DSPTXD is a 16-bit register used by the DSP to communicate to the host PC. The PC has only read access to this register and any PC write to this register is ignored. The DSP has only write access to this register and any DSP read from this register causes invalid data to be read. DSPTXD is dual mapped into (a) PC I/O address as specified by CCSR in attribute memory and (b) PCMCIA memory address 000006h. Contents of this register can be read by the PC either in the memory space or in the appropriate I/O space (PC COM port 1 – 4 as specified by CCSR). A DSP write to this register generates a TXFULL interrupt to the PC, if enabled. Similarly, a PC read from this register generates a TXEMPTY interrupt to the DSP if enabled. See Section 6 for more detail.

DSP Data Receive Register (DSPRXD):

DSPRXD is a 16-bit register used by the PC to communicate to the DSP. The PC has only write access to this register and any PC read from this register causes invalid data to be read. The DSP has only read access to this register and any DSP write to this register is ignored. DSPRXD is dual mapped into (a) PC I/O address as specified by CCSR in attribute memory and (b) PCMCIA memory address 000008h. PC can write to this register either in the memory space or in the appropriate I/O space (PC COM Port 1 – 4 as specified by CCR). A DSP read from this register generates a RXEMPTY interrupt to the PC enabled. Similarly, a PC write to this register generates an RXFULL interrupt to the DSP, if enabled.

PC Status Register (PCSR):

The PCSR is a 16-bit read-only register to the DSP located at I/O address 0052h when the card is in smart mode. It is used by the DSP space to determine status of the host communication registers and peripheral status.

Bit I/O Register (BIOR):

The BIOR is a 16-bit read/write and/or read-only register accessible only to the DSP. It is located at I/O address 0053h when the card is in smart mode. This register is used for bit I/O to control the AFE card. Bits 0–3 (I0–I3) are defined as read only input bits and bits 15–8 (O7–O0) are defined as read/write output bits. The DSP writes and reads directly from this register to control and monitor external events. At power-up, all output bits are set to 0.

System Configuration Register (SYSCFG):

The SYSCFG register is a 16-bit register containing four read-only input bits and eight read/write output bits. It is accessible only the DSP and is located at I/O address 0053h when the card is in smart mode. This register is used by the DSP to control the frequency of the clock input to the DSP when CLKMD1 = 0, bus arbitration, global data memory paging, and external-memory configuration and paging.

Operational Examples

The following few sections give some examples of card setup and use.

Setting Up DSP

After the card has been placed in a valid PCMCIA slot with a valid enabler for the PCMCIA 'C51 DSP MediaCard and communication is being done through PCMCIA card/socket services:

1. Switch to smart mode by writing a 20h to location 10h in attribute memory (COR).
2. Read DSPCR (location 02h in common memory) for the default value (108h – configured as DSP in reset, DSPCLK is off, CLKMD1 = 0) (i.e., divide-by-2-mode, microprocessor mode, all interrupt enables off, manual hold is inactive, as is maximum overdrive).
3. Write 10Ah to DSPCR to turn on the clock (CLKOUT1 is 50 ns).
4. Write 10Bh to take DSP out of reset.

If switching to PLL (25 ns i.e., CLKMD1 = 1) mode from step 4 above, do the following:

1. Read DSPCR (should read 10Bh).
2. Write 10Ah to DSPCR (reset DSP).
3. Write 108h to DSPCR (turn off clock).
4. Write 10Ch to DSPCR (switch CLKMD1. Other changes such as MP/$\overline{MC}$, MAXOD, etcetera should be done during this step).
5. Write 10Eh to DSPCR to turn on clock. (CLKOUT1 is 25 ns).
6. Write 10Fh to DSPCR (to take DSP out of reset).

Loading and Executing Single Algorithm

Initially PC loads the desired algorithm to the DSP memory and initializes the DSP. This probably is done using the PCMCIA card services protocol which is specified in [1] and described in [5].

Then PC enables the $\overline{AFEINT}$ by setting AINTEN bit allowing PC to be interrupted by the AFE card (voice activated switch, ring detect, etc.). Following the enable, the PC can reduce power consumption and turn the DSP clock off by setting CLKON bit to 0. This puts the DSP in hold mode, buses in 3-state, allowing PC quicker access to remaining unused memory on the card. When the desired external event occurs (indicated by $\overline{AFEINT}$), PC turns the DSP clock on and DSP starts executing the algorithm. Since the algorithm is already loaded into DSP memory, there is no delay in loading the algorithm. The code also can be written into global-data memory, and the DSP can be bootloaded by the PC.

Loading and Executing Multiple Algorithms

PC initializes DSP and loads DSP operating system.

Operating system loads various DSP algorithms (first algorithm in page 1, second algorithm in page 2, etc.) PC and DSP must follow a predetermined handshake protocol. Commands and data can be passed easily by using the communication registers without halting DSP operation. DSP operating system controls enabling of DSP program/data pages and transmission on processed data to the PC. The characteristics of this operating system can greatly determine which data exchange protocol is more efficient: (1) the lower-rate single-data model as in a modem or (2) the higher-rate block-data transfers as in a sound card.

Reset

When the card is powered on and hard reset occurs (i.e., a PCMCIA RESET signal is asserted and deasserted), it is in standard mode (TMS320C51 inactive i.e., $\overline{RS}$ = 0 and no clocks are going to the DSP). DSP control logic on the FPGA holds the TMS320C51 in reset by way of the $\overline{RS}$ signal and keeps the clock

off until the host turns it on after a mode change. When a mode change is requested from standard to smart mode, the PC can take the TMS320C51 out of reset. This can be done since the smart registers are available to the PC when switched to smart mode. Initialization or operating system code could be loaded into the SRAM starting at address 0h prior to the change to smart mode. Or the DSP code could be loaded into FLASH EEPROM, allowing the DSP to boot load out of FLASH memory. When the host requests a mode change from smart to standard, the TMS320C51 is once again reset and held in reset until the next mode change.

The card can have a soft reset by writing to the SRESET bit in the COR of the CCSR in the attribute memory as defined by the PCMCIA spec. This soft reset has the same effect as a hard reset, except that the CCSR values are not cleared. The card does not have to be in smart mode to access the CCSRs.

Memory

The PCMCIA DSP MediaCards provide two separate memory spaces for the common memory and attribute memory as defined by [1]. Both memory spaces are accessible by DSP and PC. DSP accesses the common memory in its program and data space, and attribute memory in its global-data memory space. The following sections contains the specs for a general 'C51 DSP MediaCard implementation. As mentioned in Section 2.0, this DSP MediaCard spec supports up to 64M-bytes RAM as PCMCIA common memory and 32M bytes (even bytes) of separate attribute memory (since all 26 PCMCIA address lines are decoded). But this spec limits the paged DSP external program/data space to 9M words and global-data space to 128K bytes (because of paging scheme given in SYSCFG in [7]).

Common Memory

The size of common memory that can be shared by the PC and the DSP depends on the number of address lines pinned out to the memory bus, and, obviously, the amount of memory on the card. The CPD/$\overline{\text{SPD}}$ feature allows for maximum memory flexibility depending on the application. In both modes, page 0 is fixed and subsequent pages are enabled using SYSCFG (If either of these modes is not used, then the appropriate bits can be RESERVED). As mentioned in Section 3.2 and specified in [7], the 32K pages are enabled on an N–1 basis. The 'C5x DSP can access up to a maximum of 9M bytes of common memory on the card starting at address 0 with the configuration shown in SYSCFG. This is calculated by using 32K-word blocks of memory with four bits of paging and two spaces with CPD/$\overline{\text{SPD}}$ = 0, and 32K-word blocks of memory with eight bits of paging and one space with CPD/$\overline{\text{SPD}}$ = 0.

DSP common (external) mem = (32K X 2^4 X 2) + (32K X 2^8 X 1)

= 9M words= 18M bytes

Any additional memory on the card is only accessible by the PC. The 'C5x DSP accesses the memory in pages of 32K x 16 each. The 288 pages are divided as 16 pages of program memory and 16 pages of data memory and 256 pages of combined program and data. Only four or two pages can be active at any time (four if CPD/$\overline{\text{SPD}}$ = 0, two if CPD/$\overline{\text{SPD}}$ = 1). Page 0 is always active. This allows DSP operating systems to use page 0 as system memory and additional pages as application-specific memory. Page 0 and 1 are selected with CPD/$\overline{\text{SPD}}$ = 0 as default upon power up.

Figure 4 is a typical memory map of the PCMCIA common memory for separate program and data spaces.

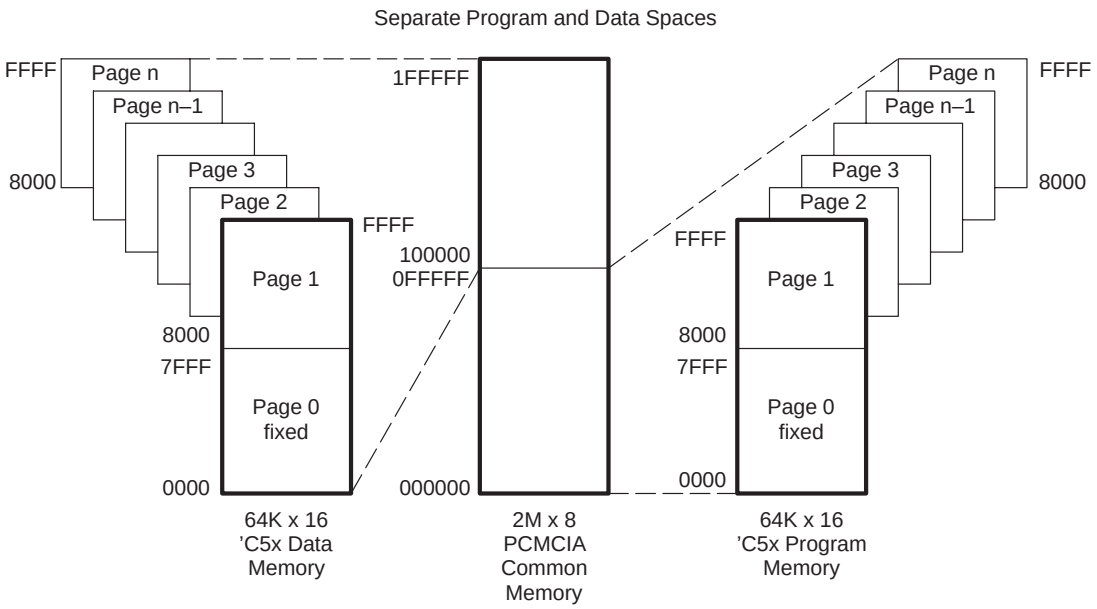

Figure 4. Typical PC/DSP Common Memory Map (Separate Program and Data)

In separate program-data space mode (CPD/$\overline{\text{SPD}}$ = 0), there are separate program and data spaces. This is similar to how the TI software development systems (SWDSs) were designed. The bits 11–8 in SYSCFG are defined as EXPP bits to page through program memory and the bits 15–8 in SYSCFG are defined as EXDP bits to page through data memory. Figure 4 details the specific configuration of the memory map of this particular DSP MediaCard. According to PCMCIA specs common memory is the main shared memory from which code is run, data is stored, etc. The PCMCIA spec also contains an execute-in-place (XIP) standard that allows code to be run from the PCMCIA bus.

Figure 5 is a typical memory map of the PCMCIA common memory for combined program/data space.

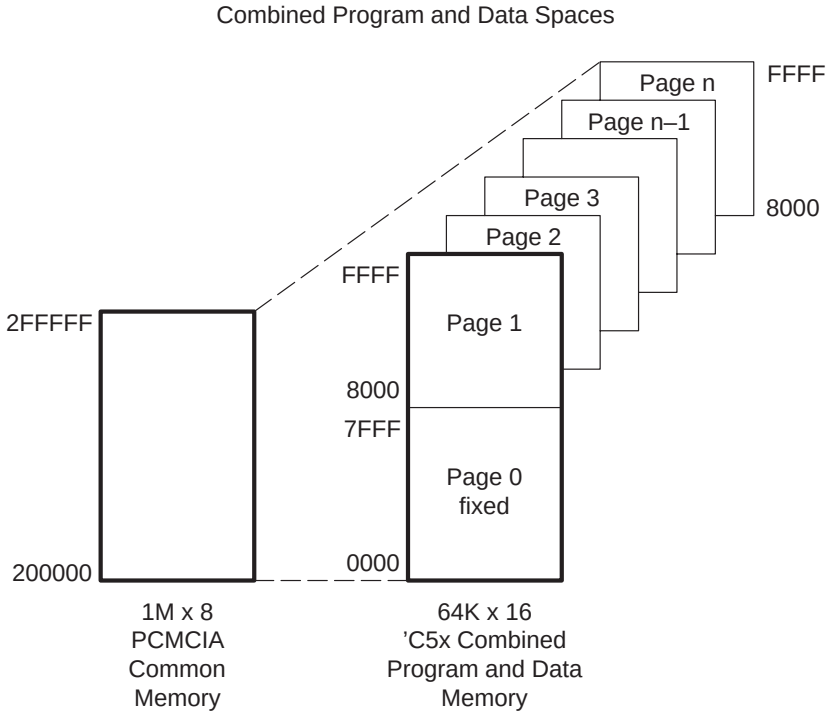

Figure 5. Typical PC/DSP Common Memory Map (Combined Program Data)

In combined program-data space mode (CPD/$\overline{\text{SPD}}$ = 1), the program and data spaces are ORed together to give the flexibility in assigning program and data memory. However, the user must be careful not to overlap program and data assignments since one of the spaces is overwritten. This is similar to how the TI evaluation modules (EVMs) are designed. The bits 15−8 of SYSCFG are defined as EXC bits to page through combined program and data memory. ORing the program and data space together reduces in half the memory addressability range for the part, and the additional bits for paging increases it. Reference [7] shows the configuration for a specific card.

Attribute Memory

Again, the size of attribute memory that can be shared by the PC and the DSP depends on the amount of address lines pinned out to the memory bus, and the amount of memory on the card. The ’C5x DSP can access up to a maximum of 128K bytes of attribute memory on the card starting at address 0000 with the configuration shown in SYSCFG (assuming that the ’C5x global-memory register (GREG) is configured correctly. This is calculated by using 32K word blocks of memory, and 2 bits of paging:

DSP attribute (global data) mem = 32K × 2^2 = 128K bytes

Any additional memory on the card is only accessible by the PC. Most of attribute memory (except CCSRs) is located in non-volatile read-only memory (such as ? and EPROM). Cards can decide to use a FLASH EEPROM which would allow user reprogramability under special conditions.

Figure 6 shows a typical memory map of the PCMCIA attribute/global data memory. The actual memory map for a particular implementation can be found in Figure 6 of [7].

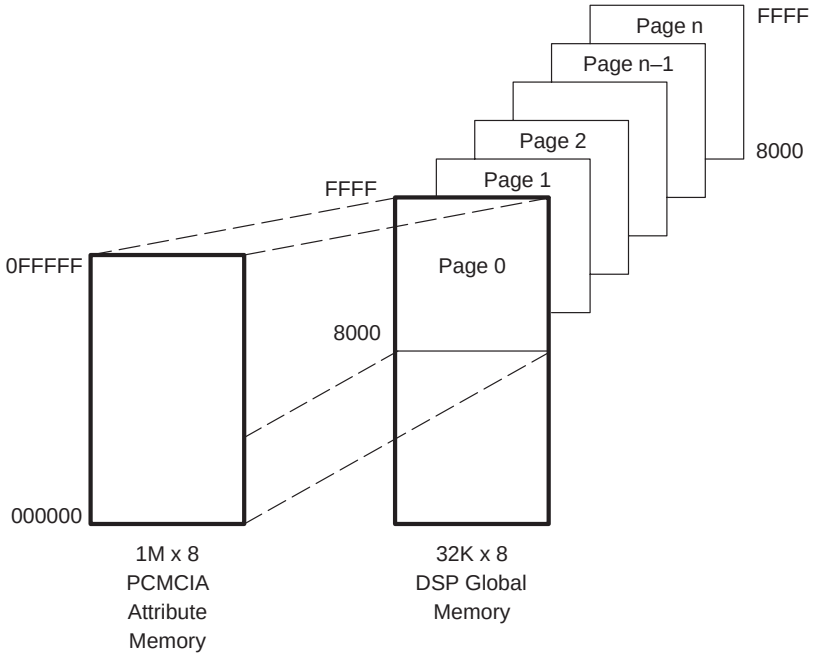

Figure 6. Typical PC/DSP Attribute/Global-Data Memory Map

According to PCMCIA specs the lower part of the attribute memory (starting at location 00000000h) contains the CIS. These are configuration bytes, called tuples, are read by the PCMCIA controller in a computer when a card is inserted to determine the cards configuration. As specified by PCMCIA, four 8-bit read/write registers are located in the FPGA/ASIC and called the card configuration registers (CCR). These bytes are used for storing card configuration information (PC I/O interrupt, I/O address, etc.) and are physically located in the FPGA/ASIC. Figure 7 describes the beginning of a typical CIS (The actual one for this board can be seen in Appendix B.2 of [7]):

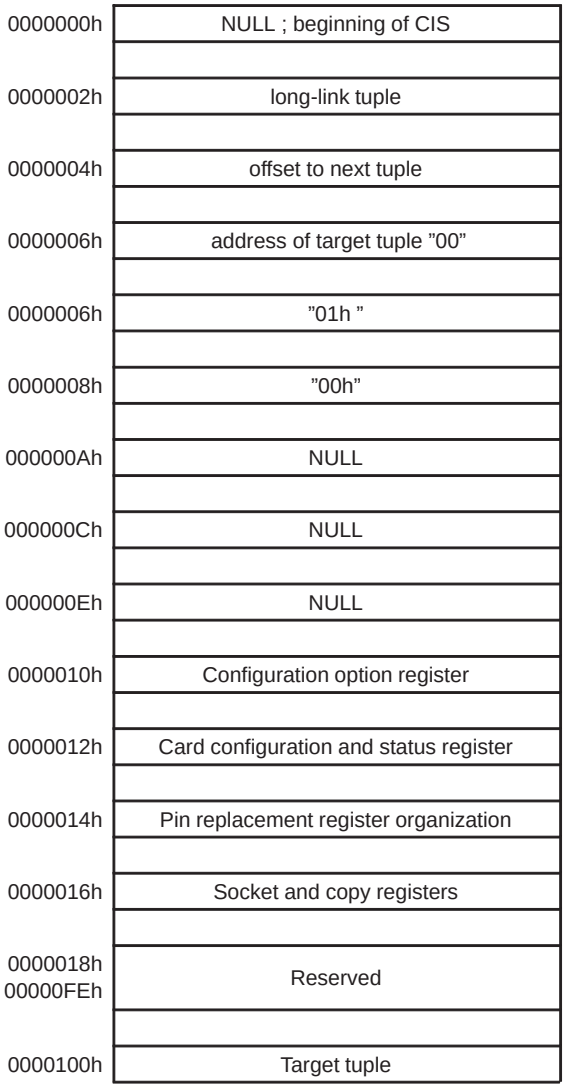

Figure 7. Example of a Typical Attribute Memory Map

More detail on the CCRs as defined can be seen in [1] while implemented in [7]. They are only reset by a hard reset from the PCMCIA bus. They are not affected by a soft reset.

Memory Interface Signals

The primary goal was to get fast DSP instruction cycle (25 ns) with relatively affordable SRAMs (15 ns). This required the use of a no-decode interface since the data access time of a TMS320C5x-80 is 15 ns. Other motivational factors included getting highly dense memories (causing the need for paged memory) and thin packages for the PCMCIA size-form factors. Also low power was a major consideration. The following few sections describe how various memory interfaces should work and where they were used in the PCMCIA card. The memory map section shows a figure of the final memory map and describes how it is configured.

No Decode Memory Interface (SRAMs)

The memory interface has been optimized for maximum DSP performance. Since address access time (t_{aa}). (see [10]) in a 25 ns 'C5x is 15 ns and 15 ns SRAMs are being used, the memories must be accessed by the DSP with no address decode. Thus, RAMs that have $\overline{WE}$ controllable access are needed. This allows the RAM $\overline{OE}$ to be tied low, the DSP memory strobe ($\overline{STRB}$) to be connected to a memory chip select ($\overline{CS}$ or $\overline{CE}$), and DSP read-write signal (R/$\overline{W}$) to be connected to RAM write enable ($\overline{WE}$). The 'C51 timing shows that this configuration allows for no decode zero wait-state 25-ns instruction-cycle external-memory accesses (this can be seen in p. 6–? of [11]). Figure 8 shows an example of the 'C5x no-code SRAM interface.

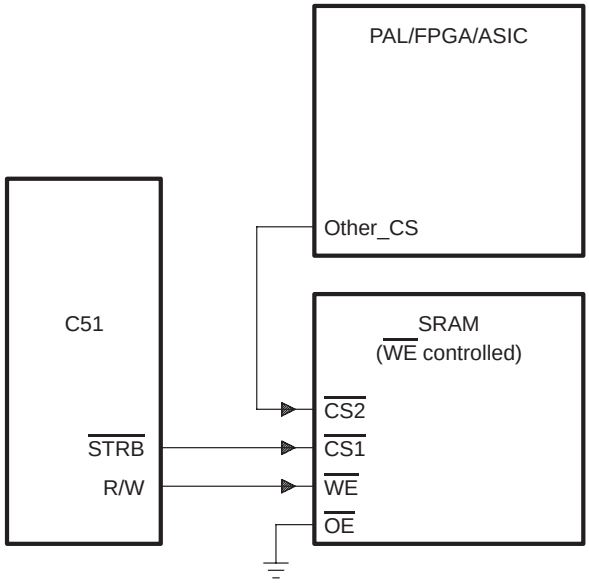

Figure 8. 'C5x No Decode SRAM Interface

One aspect of zero decode logic that should be introduced is the use of address and memory strobes in the configuration shown in Figure 8. In Figure 8, address lines, $\overline{PS}$ and $\overline{DS}$ are not shown. One possible configuration is if the RAM is 64K (i.e., address lines A0 to A15) and that corresponding DSP and RAM address lines are connected. $\overline{PS}$ and $\overline{DS}$ are left unconnected. This gives the combined program and data scheme that is seen in a 'C5x EVM. This means that program and data spaces overlaps in the external RAM. This is how the Toshiba RAM is configured on this card. This gives flexible program/data allocation and also makes memory paging quite easy (assuming the user gets higher density SRAMs. If the user needs to cascade SRAMs, then chip-selects need to be used. In either case, it is assumed that paging is not zero wait state). The disadvantage is that the address range of the DSP has been cut in half.

Another method of no decode logic address mapping is to use 128K memories (like the IDTs), and to connect A16 to one of the memory space strobes $\overline{PS}$ or $\overline{DS}$. In the case of this board, $\overline{PS}$ is connected to A16. Thus the single bank of 2 IDT RAMs (8 bytes each) are divided into separate 64K blocks of program and data which allows full speed operation.

A disadvantage of both these schemes is seen in the name, no decode. This means that sub-64K (or whatever maximum size) blocks cannot be paged, enabled, etc. Also the disadvantage of having both configurations on one memory bus is that $\overline{PS}$ is connected to SRAMA16. This makes paging impossible beyond page 0 for either SRAM or Flash EEPROM. But this card is also a showcase card, and was meant to show all sorts of configurations. In reality it allows a bank switch between combined program/data and separate program/data, but does not let the user page either one.

Decoded Memory Interface (Flash and Other Slow Devices)

The memory interface for the FLASH EEPROM works differently. Since the Flash EEPROM is slow and accesses are not so time critical, accesses use wait states and use decode in the FPGA. This particular interface appears more like a standard-memory interface (similar to Figure 6–13 in [10] except that there is no decode logic there since there is only one device on the external bus). It uses read, write, and select signals ($\overline{OE\_MEM}$, $\overline{WE\_MEM}$, and $\overline{FLASH\_CS}$) from the FPGA to the FLASH EEPROM (these memory strobes have also been used in other cards for slow peripherals such as a parallel stereo codec). These signals are decoded through the FPGA from either the PCMCIA bus and/or DSP read, write, and select signals. In the case of PCMCIA, $\overline{OE}$, $\overline{WE}$, $\overline{REG}$, $\overline{CE1}$, and $\overline{CE2}$ are used. The later three signals select attribute memory and byte/word access. In the case of the DSP, the redundant DSP signals $\overline{RD}$ and $\overline{WE}$ are used along with $\overline{BR}$ to indicate global-data memory. Other devices, such as stereo codecs or UARTs also can be put on this bus and the memory strobes used. A separate $\overline{CS}$ is needed for each device.

Memory Paging

Since a ’C5x is a 16-bit machine with a 16-bit address, memory paging is needed if more than 64 K of memory needs to be addressed in a particular space. Some external device needs to supply the upper addresses for the denser than 16-bit memory. Often this paging is controlled by the processor. This can be done by having the DSP write to a register located in its I/O space whose data lines are the higher address bits (Figure 9).

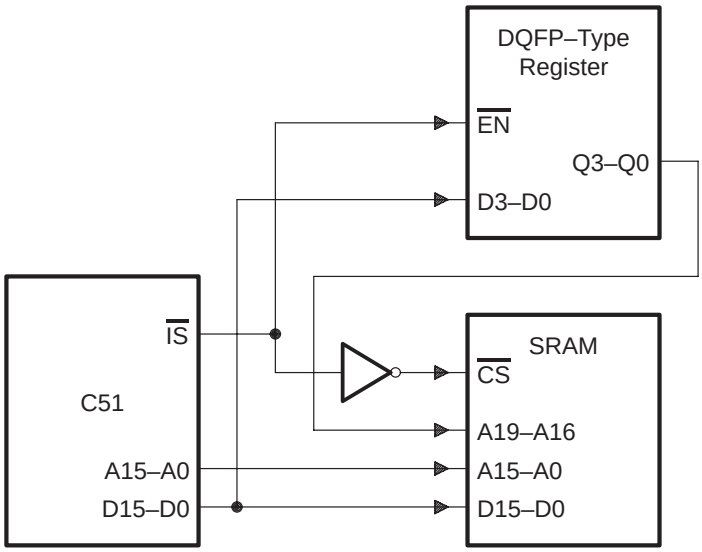

Figure 9. ’C5x Paging Hardware

In the PCMCIA card design (shown in Figure 9) this register was located in the FPGA and allowed a pretty simple paging scheme. This scheme works well with the FPGA since it is still no decode and the SRAMs share the same bus with the DSP. On the PCMCIA card, this is a non-zero wait-states page-swap paging scheme. This means that when banks of memory are paged through (by the DSP writing to an FPGA register located in its I/O space), a few wait states are required for the swap to the other bank. After reaching this bank, subsequent accesses will again be 0 wait state. This paging involves no decode and some manipulation was done to gain the desired memory configurations (see memory map section) and the paging is static. In dynamic paging, some decode would be done on each memory access, probably on address lines and space strobes ($\overline{PS}$, $\overline{DS}$, $\overline{IS}$) thus allowing a wide range of memory configurations. The FPGA register selects the upper address bits coming from the FPGA for the paging of the SRAM. On the PC side, the upper bits are determined by the address accessed selected by the PCMCIA.

Shared Memory Interface (Chip Select)

The schematics in Appendix A of [7] show that the SRAMs used have two chip selects, one from the PC through the FPGA and one from the DSP ($\overline{STRB}$). This means that the accesses can be enabled/disabled from either DSP or PCMCIA. It allows a quasi-dual port memory scheme, sharing one address bus. Thus when the DSP is accessing an SRAM (its $\overline{STRB}$ is asserted), the FPGA/PC chip select must also be asserted. The FPGA logic can tell if DSP is active, if the DSP $\overline{HOLD}$ line is high. Similarly when the PC accesses the SRAMs, the DSPs chip select must be asserted. The indication for the FPGA logic that the DSP is not on the bus is that $\overline{HOLDA}$ is low. But if $\overline{HOLDA}$ is low, then all DSP memory interface signals including $\overline{STRB}$ are in the high-impedance state. One solution might be to put a pull-down resistor on the $\overline{STRB}$ line, but this is not a good idea since the ground plane could get noisier. Rather the FPGA logic configures $\overline{STRB}$ line as a bidirectional signal and asserts $\overline{STRB}$ low when necessary. A pullup is placed on $\overline{STRB}$ as a precaution to compensate for delays through the logic when 3-state/driving switching is occurring.

Shared Memory Interface (Bus Arbitration)

The bus arbitration scheme in the PCMCIA card was probably the most difficult to achieve. It was desired to make the PC have absolute priority over the shared memory.

The $\overline{\text{HOLD}}$/$\overline{\text{HOLDA}}$ operation of the DSP can cause a locked state in a worst-case situation. Thus, the FPGA was designed around this and various features added to optimize arbitrating operation. $\overline{\text{HOLD}}$ is the signal that is used by the bus arbiter on the FPGA to request the memory bus from the TMS320C51. $\overline{\text{HOLDA}}$ is used by the TMS320C51 to grant the use of the memory bus to another device. The bus arbiter asserts $\overline{\text{HOLD}}$ whenever the card is in standard mode, when the host wants access to memory on the card, and if the PC asserts a manual hold. The TMS320C51 asserts $\overline{\text{HOLDA}}$ in response to the asserted $\overline{\text{HOLD}}$ signal to acknowledge that it is not using the bus. If the TMS320C51 is on the bus, it completes its current access before it releases the bus and asserts $\overline{\text{HOLDA}}$. In smart mode, host accesses are not allowed until $\overline{\text{HOLDA}}$ is asserted by the TMS320C51. In standard mode, $\overline{\text{HOLDA}}$ has no effect on host accesses.

On important issue regarding the $\overline{\text{HOLDA}}$ is the possibility of it taking longer than the PC timeout (1.2 μs maximum). Exceeding this wait could cause main memory DRAM refresh to not take place and crash the machine in certain PCs (most should have a timeout built into their PCMCIA bus controller). The FPGA will timeout the access if it takes to long to indicate with a bit interrupt in DSPSR. $\overline{\text{HOLDA}}$ delay is possible if long wait states are used, especially if the DSP clock has been slowed considerably ($\overline{\text{HOLDA}}$ when clocks are shut off).

Another issue involves control signal behavior in the $\overline{\text{HOLD}}$ mode. $\overline{\text{STRB}}$ goes into the high-impedance state when $\overline{\text{HOLDA}}$ is asserted; therefore, the FPGA must assert $\overline{\text{STRB}}$ which is connected to one of the two chip selects on each of the SRAMs when $\overline{\text{HOLDA}}$ is asserted. READY is a TMS320C51 input signal that is used to extend a memory bus cycle (add wait states). The TMS320C51 wait state generator on the FPGA must supply the READY signal if the TMS320C51 external memory transaction cannot be completed in the current cycle. There are some subtleties in the operation of the bus arbitration. But since the READY timing on the ’C5x is difficult to meet, the ’C5x software wait-state generators must be set up for multiple wait states for non-SRAM (i.e., I/O or global data). The default upon reset is 7.

Another important issue that affects bus arbitration is the turn-off time of non-SRAM devices (such as the slow FPGA). P 3–5 of the [13] discusses this issue in a simple design. A turn-off problem occurs with 0 workstation SRAM while accessing something besides SRAM in program or data space (I/O, global-data memory, etc.) due to the slowness of the FPGA. If the last non-SRAM access turn off is slow, then the next 0 workstation program/data access can be corrupted. If this access is a program fetch, then the DSP may jump out of normal code. If this access is data, then obviously bad data is exchanged. One dead cycle seems to be enough to prevent any conflicts even with a 25-ns DSP. This can be done with I/O by purposely using a IN/OUT instructions instead of memory-mapped I/O. Otherwise, the software workstation generator should be used to flip SRAM accesses to at least 1 wait state before the access (and then switch back to 0 workstation after the access).

Bus Arbitration

As mentioned in the Operation section, the FPGA/ASIC arbitrates accesses to the memory bus and its on-board registers (i.e., the PC/DSP shared resources). This arbitration is done with the PC having priority, though any bus cycle started by one device must be completed before giving bus/FPGA access to the other party. There is also a high level of software programmability from the PC that allows the optimizing of bus arbitration to a specific application (using MANUAL HOLD, MAXOD, DEVBUSYINT, DSP $\overline{\text{HOLDA}}$, and PDH bits). This architecture allows zero wait-state access to external memory for the DSP since it lies on the same bus. This was the prime directive of this design. The disadvantage of this architecture is that any other access requires FPGA/ASIC involvement and requires a few wait states. An alternative method of data exchange is using the host-communication registers DSPTXD and DSPRXD. But interrupt or polling latencies do apply in this operation. Thus, PC access of common memory, attribute memory, and

SMMs require wait-states as do DSP access of global-data memory and SMMRs. The PC cannot access DSP internal memory.

Both DSP and PC can access the shared memory on the card. PC always has higher priority for accessing the memory bus on the card (after a concurrent DSP cycle is completed), except for a situation explained in the next paragraph where the PC may override. During PC accesses to the memory bus, the DSP operation is held using the DSP $\overline{\text{HOLD}}$ line. See [10] for latencies of this process and also the HM bit = 0 feature that allows the DSP to continue executing out of internal memory when held. The arbitration logic asserts the $\overline{\text{HOLD}}$ signal to the DSP and extends the PC memory-bus access cycle by asserting the $\overline{\text{WAIT}}$ signal. A maximum timeout is set on the $\overline{\text{WAIT}}$ from the FPGA. Once the DSP acknowledges the hold by asserting $\overline{\text{HOLDA}}$, the PC $\overline{\text{WAIT}}$ is released and access to shared memory is completed. As soon as the PC completes its access, control of the shared memory is returned to the DSP.

In the default mode, any PC access of the FPGA/memory bus applies $\overline{\text{HOLD}}$ to the DSP (i.e., MAXOD = 0 in DSPCR). But since communication, control, and status registers are not resident in the shared memory PC access to these registers can be configured to not $\overline{\text{HOLD}}$ the DSP operation. When MAXOD = 1 in DSPCR, the DSP is not halted on PC accesses of its SMM registers on the FPGA/ASIC. If the DSP tries to access its SMMRs while the PC is accessing its SMMRs, the DSP READY line is held low (hardware wait-state) until the PC SMM access is completed. But if the PC tries to access the memory bus and the DSP tries to access its SMMRs (in DSP I/O) or global-data memory (anything that must use the FPGA) at that same time, the ASIC/FPGA is forced to extend the $\overline{\text{WAIT}}$ until the DSP is off. This mode of operation is done to prevent a lock-up of the system due to DSP operation. This is because the DSP cannot return $\overline{\text{HOLDA}}$ until the bus cycle is over. If while the PC requests the memory bus by asserting $\overline{\text{HOLD}}$ and the DSP tries to access the FPGA, the system could be hung. To prevent this situation, a PC memory access timeout has been added that causes a DEVBUSY interrupt in DSPSR. This interrupt is not cleared until a valid memory access occurs. Another situation that might cause this interrupt is if the DSP is run very slow and does not give back $\overline{\text{HOLDA}}$ fast enough. The solution to assure PC access to the memory bus is to assert the manual hold bit in DSPCR. MANUAL HOLD is also useful for copying blocks of data. The clock to the DSP (DSP CLKON in DSPCR) must be running for the DSP to be able to return $\overline{\text{HOLDA}}$. The value of $\overline{\text{HOLDA}}$ can be polled in DSPSR. If MAXOD = 0, DSP $\overline{\text{HOLDA}}$ is always 0.

A feature added to optimize DSP external-memory usage is a non-binding please-don't-hold-me (PDH) bit located in the PCs DSPSR that the DSP can assert in its SYSCFG to request the PC to stay off the memory bus. When the DSP is doing critical external memory operations, it can write a 1 to this location and the PC can poll this bit (with MAXOD = 1) and choose to stay off the memory bus. But again the PC has priority and can ignore this request.

Memory Access by PC

When PC accesses the shared memory, the DSP is put on hold ($\overline{\text{HOLD}}$ signal asserted) to grant control of the bus to the PC. PCs memory access is extended using wait signal until DSP 3-states its bus as indicated by $\overline{\text{HOLDA}}$ signal. There is a timeout if $\overline{\text{HOLDA}}$ is not granted in time indicated by the DEVBUSYINT bit. This is a dumb clock that just begins counting at the beginning of a PC bus access cycle. If software wait states are used and/or $\overline{\text{WAIT}}$ signal is ignored (as can be done per the PCMCIA spec), this bit becomes meaningless. Running certain configurations of the DSP causes $\overline{\text{HOLDA}}$ to not be returned low in time (such as shutting the clock off (DSP CLKON = 0) when $\overline{\text{HOLDA}}$ = 1 or running the DSP very slow).

When the card is in smart mode, PC cannot access the first 16 bytes of the shared memory (also note that the PC cannot access DSP internal memory). This could be used as protected memory for the DSP. PC accesses to this block do not cause DSP to be put on hold. PC must load the DSP reset and interrupt vectors, and application algorithm prior to taking the DSP out of reset if in DSP microprocessor mode or the code

must be programmed into the DSP global memory for bootload if in the DSP microcomputer mode. Since PC can access the entire memory on the card without consideration of DSP page sizes, memory pages not used by DSP can be dedicated exclusively for PC.

Memory Access by DSP

The ’C5x versions of the DSP MediaCards can address a maximum of 9M words of common memory. The DSP address range is expanded using page selects. Page sizes for ’C5x DSP is 32K (x16). Page 0 (both separate program and data memory in CPD/$\overline{\text{SPD}}$ = 0 or combined program and data in CPD/$\overline{\text{SPD}}$ = 1) is always enabled and can not be deselected using page select bits in SYSCFG register . This allows DSP operating systems to use this memory without affecting any memory dedicated for DSP applications.

The ’C5x version of the DSP MediaCard can address a maximum of 128K byte of global-data memory. At least part of the PCMCIA attribute memory should be available to the DSP in its global-data memory. This allows a non-volatile memory for bootload of the DSP by the PC, thus allowing code to remain on-board. This bootload should always be from page 0 of global-data memory. After bootloading in a kernel, DSP functionality can be engaged. This space is also available for data storage, though accesses are typically slow due to the nature of non-volatile memory.

Host/DSP Communication

DSP communicates to the host PC via its dedicated communication registers DSPTXD and DSPRXD. These registers reside in the I/O space of the DSP. For the PC they are located in common memory, but also dual-mapped into PC I/O space. The I/O location is selectable in the COR register. Both PC and DSP can use hardware interrupts or software polling for communicating to the other device. Figure 10 shows a block diagram of the structure of this communication.

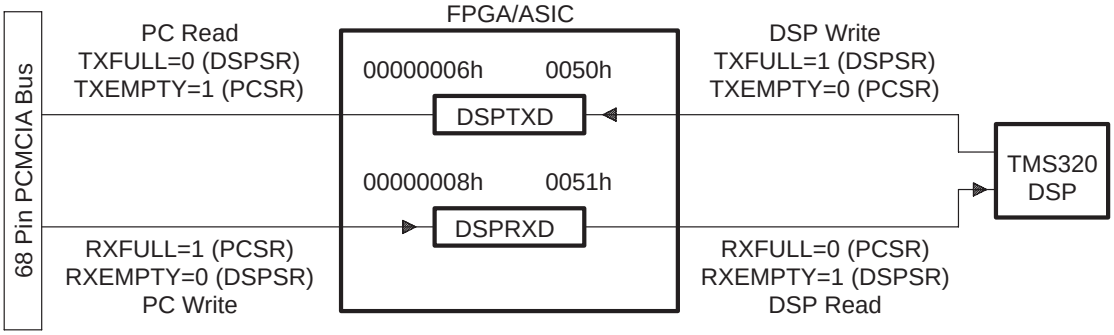

Figure 10. Host Communication Structure

PC must enable the communication interrupts by setting RXINTEN or TXINTEN to 1 to be interrupted. A 1 in either TXFULL or RXEMPTY (in DSPSR) generates an interrupt to the DSP. The PCMCIA standard only defines 1 interrupt line; therefore, after an interrupt is detected, DSPSR must be read to determine the source. The interrupt is cleared by PC reading DSPTXD or writing to DSPRXD respectively. Writing 0 to the interrupt enables disables, but does not clear the interrupt.

A 1 in either RXFULL or TXEMPTY bits (PCSR) generates an interrupt signal to the DSP (such as $\overline{\text{INT1}}$). This interrupt can be masked off in the DSP algorithm if not used. The interrupt signals stay valid until the DSP reads from the DSPRXD or writes to the DSPTXD. Reading or writing to the data register clears the appropriate interrupts to the DSP. Writing 0 to the interrupt enables (’C5x IMR register) only disables the

interrupt, not clear it The clearing/priming protocol required to enable the next $\overline{\text{IREQ}}$ is described in detail under Bit 12 in Section 4.41 of [7]. Through this protocol, the shaping of the $\overline{\text{IREQ}}$ signal is controlled completely by the host programmer. This was done due to the variability of PC IRQ specifications over different PCs. Communication between DSP and PC can also be done via software polling as shown in Figure 10. Two other sources of interrupts to the host are discussed in the next section.

Interrupts

An interrupt to the PC (called $\overline{\text{IREQ}}$ on the PCMCIA bus) can be generated by three different sources on the card. PC must read the DSP status register DSPSR, to determine the source(s) of the interrupt.

The three sources of interrupt to the PC are:

- Communication interrupts, when enabled by the PC. Communication interrupts are generated when:
  - Either one is enabled by PC (TXINTEN or RXINTEN is set to 1)
  - The DSPTXD is full (DSP has sent new data to PC) or DSPRXD is empty (DSP has read old data from PC)
- Analog front end card, when enabled by the PC. AFE interrupt is generated when:
  - It is enabled by PC (AINTEN is set to 1)
  - The AFE card generates an interrupt ($\overline{\text{INT4}}$ to DSP)
- Device busy (memory-bus access timeout), when enabled by the PC. DEVBUSY interrupt is generated when:
  - It is enabled by PC (DEVBUSYINTEN is set to 1)
  - The timeout counter times out on a memory bus access

```
IREQ=(TXFULL and TXINTEN)+(RXEMPTY and RXINTEN)+(AFEINT and
AINTEN)+(DEVBUSYINT and DEVBUSYINTEN)
```

Figure 11. PC Interrupt Control Logic

The sources of interrupt to the DSP are:

- Communication interrupts.
  - DSP receive-interrupt is generated when:
    1) It is enabled by DSP masking its INT1
    2) The RXFULL bit in PCSR is set indicating DSPRXD is full (PC has written new data)
  - DSP transmit-interrupt is generated when:
    1) It is enabled by DSP masking its $\overline{\text{INT}}$[1 or 2]
    2) The TXEMPTY bit in PCSR is set indicating DSPTXD is empty (PC has read previous data)

(These interrupts are connected to DSP $\overline{\text{INT1}}$ and optionally the second one can be connected to $\overline{\text{INT2}}$ if there are no parallel device interrupts needed.)

- Parallel device interrupt (optionally $\overline{\text{INT2}}$)
- Analog front end card, when enabled by DSP ($\overline{\text{INT3}}$ and $\overline{\text{INT4}}$). AFE card must generate proper interrupt signal to the DSP as specified in the C5x User's Guide (literature number SPRU056).

The $\overline{\text{AFEINT}}$ which comes from the AFE connector is connected to both DSP $\overline{\text{INT4}}$ and the FPGA logic to the PC $\overline{\text{IREQ}}$. This interrupt can be recognized by either PC or DSP. Also this configuration allows the PC to handle the interrupt if the DSP is asleep.

Host Communication via Software Polling or Interrupts

While servicing a PC interrupt ($\overline{\text{IREQ}}$), the interrupt must be cleared and primed for the next interrupt to occur. This means that when PC is servicing an interrupt, a 0 must be written to this bit to clear it. Then a 1 must be written to this bit to prime it for the next interrupt. Otherwise $\overline{\text{IREQ}}$ will not toggle again. This puts the time between IREQs (pulsed interrupts pulse only once) fully in control of the PC programmer.

PC Read:

- PC reads DSPSR to verify DSP has written a new data word into DSPTXD. PC waits until TXFULL is set to 1.
- PC reads data from DSPTXD register. This reading:
  - Clears the TXFULL bit in DSPSR (read by PC
  - Sets the TXEMPTY bit in the PCSR (read by DSP)

PC Write:

- PC reads DSPSR to verify DSP has read previous data (indicated by RXEMPTY bit). Waits until RXEMPTY is 1.
- PC writes data into DSPRXD register. This writing by PC:
  - Clears the RXEMPTY bit in DSPSR (read by PC)
  - Sets RXFULL bit in PCSR (read by DSP)

DSP Read:

- DSP reads PCSR to verify PC has written a new data word into DSPRXD. Waits until RXFULL is set to 1.
- DSP reads data from DSPRXD register. This reading:
  - Clears the RXFULL bit in PCSR (read by DSP)
  - Sets the RXEMPTY bit in DSPSR (read by PC)

DSP Write:

- DSP reads PCSR to verify PC has read previous data (indicated by TXEMPTY bit). Wait until TXEMPTY is 1.
- DSP writes data into DSPTXD register. This writing by DSP:
  - Clears the TXEMPTY bit in PCSR (read by DSP),
  - Sets TXFULL bit in DSPSR (read by PC).

If interrupts are used, PC must read the appropriate bits in DSPSR to determine whether the interrupts were generated by transmit or receive operation. After reading or writing, it must then clear and prime IREQSERV as specified in section ? for subsequent interrupts to occur. DSP knows which interrupt by the particular interrupt taken.

FPGA

The following FPGA description gives an outline of the resources and functions provided by the single TPC1280 FPGA on the TMS320C51-based PCMCIA DSP MediaCard. More detailed information can be found in [9].

Overview

The block diagram shown on Figure 12 illustrates functionally how the various functional blocks in the FPGA are distributed. The PCMCIA address and data buses are assumed on the left side while the ’C51 data and address buses are located on the right side of the figure.

The alpha characters [(A), (B), . . .] in the block diagram correspond to the Roman numerals (see Figure 12) which correspond to their sheet name in the hierarchical OrCad schematic.

BLOCK DIAGRAM NO.	OrCad SHEET NAME	BLOCK DIAGRAM NO.	OrCad SHEET NAME	BLOCK DIAGRAM NO.	OrCad SHEET NAME
A	∅	H	VII	S	XIV
B	I	J	VIII	T	XV
C	II	L	IX	U	XVI
D	III	M	X	V	XVII
E	IV	N	XI	W	XVIII
F	V	P	XII	X	XIX
G	VI	R	XIII		

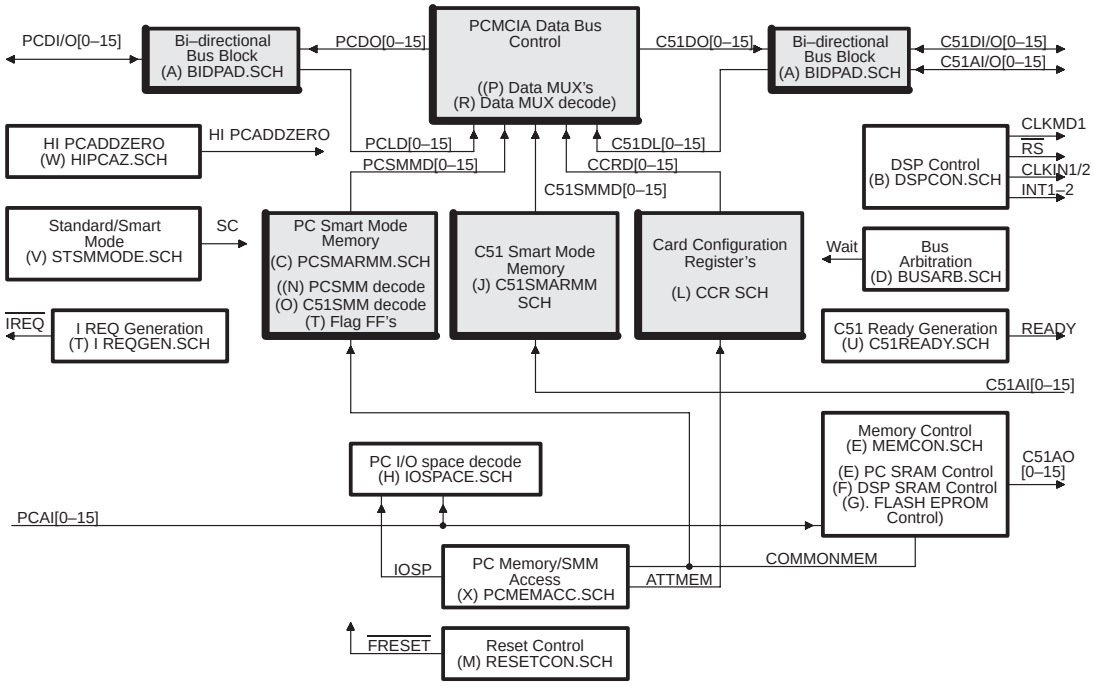

Figure 12. FPGA Block Diagram

The blocks highlighted show how data lines from PC and DSP flow through the FPGA. The rest of the blocks are control blocks. In each block is a descriptive title with an alpha designator and a filename. Next to the alpha characters in capitals is the file name for the particular sheet with the .SCH extension. Some of the sheets have more than one functional block in them and they are correspondingly labeled inside the block within parentheses. These names along with their descriptive titles are mere text in the actual OrCAD schematic and have no OrCAD function. Some of the important signals from each block are shown, but this is not a complete list.

Beginning with the highlighted data blocks, block (A), contains bidirectional buffers that allow the FPGA to read or write both data buses. The ’C51 address is also bidirectional, but PC address is not. It is only an FPGA input. This is because the FPGA can never drive the PC address since PCMCIA bus does not have

the address as an input. On the memory bus, the FPGA needs to drive the address on PC SRAM accesses and flash accesses. On DSP SRAM and Flash EEPROM accesses, the DSP drives address.

Block (V) controls the driving of the data and address buses in (A), based on various control signals from blocks (B), (L), and (X) The data comes from blocks (B), (J), and (L) that contain the FPGA registers that are read and written using the previously mentioned blocks. All of these blocks contain multiplexor arrays which direct FPGA data buses through the FPGA and to the appropriate bus for output. This multiplexing of buses is necessary since 3-state buffers do not exist in internal FPGA/ASIC logic. Block (B) contains PC smart memory registers, (J) contains 'C51 smart mode memory registers, and (L) contains the CCRs.

The control blocks, blocks (B) and (L) mentioned above, have some control functionality in them. Block (B) contains all the address and bus control signal decode/selects for both PC and 'C51 smart memory registers. Block (L) contains the decode/selects for the CCRs. Starting from the top left and moving counter-clockwise, block (W) checks for the upper PC bits (PCA16–25) to be 0 for PC smart memory decode. This eliminates the need for repeated decode of these lines. Block (V) contains the logic that switches the card between standard and smart modes. Block (S) is used to generate the PCMCIA interrupt ($\overline{\text{IREQ}}$) from various sources. Block (H) decodes I/O accesses to COM1–COM4. Block (X) decodes PCMCIA bus control signals for use in enabling FPGA registers for writes. Block (M) handles the reset of the FPGA. Block (E) manages all control signals going to the memory bus to access SRAM or Flash EEPROM. This includes all multilpexing needed between PCMCIA and DSP memory interface signals. Block (V) controls the hardware wait states to the DSP via the READY input on the DSP, while block (D) controls the bus arbitration of the memory bus. The PC always has priority to the memories, but $\overline{\text{HOLD}}$/$\overline{\text{HOLDA}}$ signals of the DSP must be used to ensure that the DSP is off the bus. Block (D) also generates the $\overline{\text{WAIT}}$ hardware wait state to the PCMCIA bus. Block (B) controls a variety of DSP functions (such as clock, DSP reset, and interrupt generations).

The FPGA system clock is run at 50 ns as it goes through one of the TPC1280's two global-clock networks. The design is minimally synchronous meaning that flip-flops are placed on critical control signals only. None of the inputs or outputs are registered (this asynchronicity is compensated by extending the FPGA cycle count). The guideline of three logic-gate levels before each flip-flop is followed (except for post HIPCADDZERO logic). Each logic level becomes an FPGA cycle count.

Concerning pure data flow for FPGA access latency, any write to an FPGA internal register requires a minimum of 2 FPGA cycles (not counting HIPCADDZERO latency which would add 1 cycle). Reads would also take 2 cycles (going through local and global – in (O) – mux arrays, again not counting HIPACADDZERO). But in (E), a great deal of signal shaping is needed to have the memory control signals work correctly. This block's maximum requirement is three logic levels meaning three cycles and adding one cycle for asynchronous uncertainty means four cycles or 200 ns (also, repeated observation with a logic analyzer on FPGA latency showed that 200 ns was a good number, and that 150 ns could be unreliable).

Example FPGA Functions

Some examples of the operation of FPGA logic which provide an idea of how these modules interact for data control follow.

- PC SRAM Access:

  The address and control signals from the PCMCIA bus are input to the FPGA, decoded in block (E) for memory control signal output, decoded in (P) to send the appropriate read/write and address signals to the buffers in (0) then to the memory bus, and the SRAMs respectively. These are bi-directional buffers that must be enabled. Based on the PCMCIA control signals, Block (X) sends a common memory strobe to (P) to enable these buffers in (0) and drive the appropriate line. The data meanwhile flows directly through the MUXes in (P).

- Smart Memory Access:

  If the card is in smart mode and an access is done to a memory location that corresponds to a smart mode register, the strobe still goes from (X) to (P). But the decode in (C) recognizes this condition and send a select signal to (P) to enable the appropriate address/data buffers. Meanwhile, (C)s signal selects the correct register and executes a write or read. In the case of a write, the data is directly written into (C) and nothing's driven on (A). But in a read, the data is selected through two layers of multiplexers – one in (C) [or (J) if 'C51SMM or (L) if CCR], and one in (P), and the enable logic drives the PC data buffers in 0.

Conclusion

The PCMCIA DSP MediaCard system has been proven successful in various systems such as sound generation and modems. Future expansion and applications for this general-purpose card is seriously being examined. The programmability of the FPGA makes prototyping these systems easy. For production systems, cost reduction into an ASIC or CDSP should be possible. The synchronous nature of the design scales down in faster technology, removing the flip-flops and taking advantage of the short propagation delays. A 100-ns access PCMCIA card with 40- or even 50-MIP DSP in a CDSP ASIC should be no problem.

Some thought should be given to some of the trends of the PCMCIA standard. Power maximums can be specified at a number as low as 300 mA operating current. This would make less power consuming 3-V parts more attractive. The whole-host software interface of card services and support from PC operating systems such as Windows 95™, OS™/2, etc. is at issue. A subtle issue that has not been explored greatly is the impact of these high overhead multi-tasking systems (along with Windows 95) on an inherently-efficient low-overhead DSP code. A final possible problem for the system is multi-tasking operating systems being run on DSPs.

Finally, a great deal of this overhead could be eliminated by the adoption of the BASAVA concept of smart memory for the execution of a variety of multimedia applications such as text-to-speech, sound, etc.

Future Expansion

Various features have been identified for future expansion.

- Buffered Host Comm (Top-of-the-Line FPGA)

  Buffers can be added to the host communication registers DSPTXD and DSPRXD so that packets of data can be transferred. Presently these would look like straight FIFOs and require very little modification to this spec. The most important information would be the size of the FIFO. Full and empty flags would be set and cleared when the entire FIFO was empty or full. Any further operation would require modification of this spec.

Windows 95 is a trademark of Microsoft, Inc., OS/2 is a trademark of IBM.

- PC/DSP Write of Attribute Memory

  User programmable writes to the attribute memory/DSP global-data memory would allow hard coding of PC/DSP code in non-volatile memory. Presently this is not envisioned in a standard slot, but would require other hardware (Databook TMI-140 Board). The biggest danger of this feature is loss of the CIS.

- Expanding Memory Size

  A sizable amount of possible memory on both PC and DSP sides has been specified. Any further expansion would involve the addition of pins and bits in registers. Since fast SRAM needed for a 40-MIP DSP is expensive, slower SRAM could be placed on the PCMCIA bus (with appropriate buffers) to make a real DSP/memory card.

- Decoded memories (Dynamic Page/Bank Switching)

  This spec does not specify the method of page/bank switching of DSP external memory (This is discussed some in [7], but in detail in the manufacturing package). The reality of 25-ns zero wait-state operation is that it is static. This means that there is zero decode memory interface between the DSP and the RAMs. This allows zero wait-state operation within a page/bank, but switching pages/banks of memory requires wait states. Zero decode allows reasonable speed/cost memories to be used in a design. With faster/cheaper memories, DSPs, or logic, dynamic page/bank switching is possible. This requires no wait states when switching pages/banks of memory since the interface is decoded.

- Modem Emulation/UART (Pure I/O addressability)

  The present PCMCIA spec does not define the type of mixed memory I/O card that this card is in specific enough terms. It mentions straight memory and recently a tuple 21h that is a modem card identifier. In this mode, the card is used only with PCMCIA I/O strobes and assuming the Hayes Comapatible Modem registers exist. Until the PCMCIA specification catches up, enabler/drivers such as the one described in [?] needs to be used. Another option is configuring this card to look like a pure modem card. This would involve some work in the dual mapping of SMM registers into PC I/O, emulation of Hayes Modem Standard Registers and automation of DSP configuration.

- Daughter Cards

  With the full ’C51 address and data buses along with other signals offered on a Texblast board, it would be easy to build parallel bus daughter cards for added I/O. These could include stereo codec, LAN/WAN, high performance A/Ds, etc. As a matter of fact a nice a TI show case daughter card could be built with a TLC320AD65 stereo codec and SN74ACT7808 FIFO to speed up DSP data access. This coupled with TLC320AC01 on the AFE card for modem/speaker and the general PCMCIA card would combine nine TI parts in a super audio/modem board.

- Software is under development for fully blown DOS™ and Windows™ VxDs to interface the host/DSP in multitasking, multi-processing environment.

DOS and Windows 95 are trademarks of Microsoft, Inc.

- Miscellaneous

  These include parallel I/O bus located on the AFE addressable by either PC, DSP, or both. Also the ability for the AFE to write the Flash. Bidirectional DSPTXD/DSPRXD would make five easier and bidirectional bit I/O would make it more flexible.

- Reference Materials

[1] PCMCIA PC Card Standard, Release 2.01, Personal Memory Card International Association, Sunnyvale, CA., September 1992.

[2] Memory Based Digital Signal Processing, B.I. Pawate, George R. Doddington, Shivaling S. Mahantshetti, Mark G. Howard, and Derek Smith, Proceedings of ICASSP 1990, Albuquerque, New Mexico, April 1990.

[3] SABRE1: A Business Plan for Providing Value-Added Integrated System Solutions, RAj Pawate and Betty Prince, Texas Instruments, Dallas, Texas. November 1991.

[4] PCMCIA: The Expansion System of the Future, Winn Rosch, PC Magazine, January 26, 1993, pp. 321–‘326.

[5] Using Card Services for Configuring PC Cards, John I. Garney and David Lawrence, IC Card Magazine

[6] Supporting External DMA Activity to Internal RAM for TMS320C5x Devices With the PZ Package, Jim Larimer, Designer’s Notebook Number 41, Texas Instruments, Dallas, Texas, 1994.

[7] PCMCIA TMS320DSP/Memory Card Specification, Texas Instruments, Houston, Texas, June 1994.

[8] PCMCIA TMS320DSP/Memory Card Hardware System Description, Texas Instruments, Houston, Texas, June 1994.

[9] PCMCIA TMS320DSP/Memory Card FPGA Description, Texas Instruments, Houston, Texas, June 1994.

[10] TMS320C5x User’s Guide (literature number SPRU056), Texas Instruments, Dallas, Texas 1993.

[11] TMS320C3x User’s Guide (literature number SPRU031), Texas Instruments, Dallas, Texas 1992.

[12] TMS320C5x Evaluation Module Technical Reference (literature number SPRU069), Texas Instruments, Dallas, Texas 1992.

[13] SABRE1: Prototype Specification: A PCMCIA-Based SABRE1 Card, Raj Pawate, Texas Instruments Internal TAR Report, February 27, 1992.

[14] DSP MediaCard Version 1.0 User’s Guide, Raj Pawate, Y. Iwata, K. Yahata, Texas Instruments TAR, June 30, 1995.