DSP SOFTWARE

This document describes software issues that are common to the boards containing digital signal processors, or DSPs. The utility board (ARC-50) contains a DSP supplied by Freescale, formerly Motorola, from the DSP56000 series. The timing (ARC-22) and PCI interface (ARC-64 and ARC-65) boards each contain a processor from the DSP56300 series. These DSPs contain digital signal processors with a 24-bit integer data word, a 16-bit address space, a fast ALU, a Reduced Instruction Set Computer (RISC) architecture that executes most instructions in one clock cycle and extensive on-chip peripheral support. These peripherals include separate address spaces for on-chip program and data memory, a synchronous serial interface, boot logic and a simple interface to an external data bus.

BOOT AND APPLICATION SOFTWARE

Assemblers and linkers for the DSP56000 and DSP56300 series of devices can be downloaded from "www.astro-cam.com". These were copied from Motorola several years ago, and are available for the Sun Solaris and Windows operating systems. The Windows version can be run on a Linux operating system by using the "wine" set of tools, available for free download from the "www.winehq.org/site/download" site.

The timing and utility boards require a boot and an application code to operate. The boot code initializes the DSP after reset and provides communication and memory maintenance functions. The boot file is written to the EEPROMs on each board by a ROM burner at the factory, and is supplied to users in the Motorola S-record format in the filenames "tim.s" and "util.p*". A boot loader program in the DSP writes the boot code from the EEPROM to the DSP internal memory right after power-on reset. It is intended that a typical user will not normally need to modify this code nor write over the EEPROMs.

Application files contain software that is specific to the particular function and set of desired tasks for each board. Application programs are transferred from the host computer to the internal memory of the DSP, referred to as downloading the application. The timing board application contains all the code specific to the CCD or IR array being used, such as the serial and parallel register clocking patterns, the clocking and DC bias voltages, the global reset code and so on. The application programs are intended to be heavily modified by users to be optimized to their particular needs and detectors. The utility board application code is generally pretty stable and uniform for all systems, although the user is encouraged to tweak it at will. And, finally, the application and boot code for the PCI interface board are joined together in one file, "pciboot.asm", and written to the DSP all at once when the computer is powered on. Alternate PCI software may be downloaded, but this is only required to use a different version than is already resident in the EEPROM.

GENERATING APPLICATION PROGRAMS for DOWNLOADING

Scripts executing on the host computer are used to convert source code to application files suitable for downloading from the host computer to the controller. These scripts assemble files with names like "timboot.asm", "tim.asm", and "timhdr.asm", then link them together into a file with a name like "tim.lod" that is downloaded to the DSP. Relative addresses are converted to absolute addresses by a linker, and the "tim.lod" file contains machine code

that is directly executed by the DSP, as well as timing information that is interpreted by the controller hardware. Downloading consists of having the host computer read each word in the "tim.lod" and write it to the DSP internal memory using the WRM command described below.

The file "tim.lod" contains the boot program as well as the application program, and the user must ensure that the boot program assembled with this script is identical to the one used to generate the EEPROM on the board of interest.

BOOT COMMANDS

Several commands are executed by all the DSP boards by the boot program, as follows. Every command must be preceded by a header ID, as described in the ARC-22 User's Manual.

TDL number "Test Data Link". The DSP will read "number" and transmit it back to the source in order to test functionality of the communications path.

RDM address - "Read DSP Memory". Read from internal DSP or ROM memory. The most significant nibble of the address designates the memory space, as follows:

Bit #20 = 1 selects P: memory
Bit #21 = 1 selects X: memory
Bit #22 = 1 selects Y: memory
Bit #23 = 1 selects ROM memory

WRM address value - "Write Memory". Write "value" to internal DSP or EEPROM memory, following the same encoding of the memory space as RDM.

A handshaking system exists to inform the sender the commands have been received and processed. Most commands reply with a "DON" when they finish executing, though there are exceptions to this. A timeout routine has been implemented in the command processor that requires that all words of a command be received by the command processor within about two milliseconds of receiving the header ID. The TIMEOUT parameter in the boot files can be changed by the user if desired. If the number of words specified in the header is not received in time then the words are simply discarded.

PROGRAM NOTES

Following are some notes on the programming of the controller boards. The intention is not to be exhaustive or complete, but to be helpful with some of the subtleties of the code.

The boot program starts at the location P:START that is specified in an equate table at the beginning of each source code file. It is chosen to be as low as possible and uses interrupt service routine vector locations that are assigned by the DSP but unused by the particular board. As a result a warning message is generated during assembly to the effect that illegal instructions for interrupt service routines are being generated, and these warnings should be ignored.

Command processing is done by having routines linked to input data devices write incoming commands to circular buffers indexed by address registers that are permanently allocated to that task. The address register assignment is listed after the ROM_ID table in the source code. The circular buffers are set up to be 32 words long, which is enough to contain typically ten commands, and resides in the Y: memory space. This allows several commands to be sent to a given board in sequence without having to wait for each one to be executed; they will be executed in the order in which they are received. Each input device is assigned a separate circular buffer to prevent intermingling of commands. If a command is not being executed then the program will scan the address register values for an increment caused by an incoming command. As soon as one is found a consistency check is performed to verify that its first word is a valid header. If it is correct, then the last byte is read to determine when the complete command has been received. If there is more than one input data source then the command is moved to a separate buffer that contains co-mingled commands. A loop is entered to check that the entire command has been entered. Once the command is complete the header destination number is examined to see if the command needs to be executed by the board or passed on to another one in the chain. A table of valid commands is examined for a match with the second word of the command. If no match is found an "ERR" reply is returned to the source. Otherwise execution continues at the address indicated by the command.

A reply routine, named FINISH, constructs the reply header ID by moving the source byte to the destination byte location, tacking on the correct source byte number, and adding two for the number of words in the reply. All replies are two words in length. The reply is added to the circular command buffer and then processed as any other command would be.

The INIT routine is executed once by the DSP on boot up. It is overwritten by application code downloaded from the host. As a detail, note that when application code is downloaded the boot code is overwritten while it is being used to execute the download process with repeated calls to the WRM command. This is why the EEPROM boot code needs to be the same as the boot code used to assemble the download file. The INIT routine initializes DSP control registers, sets up the circular buffers, reads in X: memory from ROM, and initializes for the processing of interrupts. The timing board INIT routine goes on to set the DC bias DACs with safe intermediate values in preparation for the power-up sequence