MFC DSP Software Outline

This document provides a description of the DSP software provided on the MFC board and some tips and guidelines to add more functionality or to make changes to the existing software. An overview of the main components on the MFC board is shown in Fig. 1. The primary functions carried out by the DSP software is the initialization of the board at powerup, processing tasks from the slot0 controller such as data acquisition, parameter scaling, user interface support through its UART and various data transfer operations to and from the FPGA through its parallel port and serial ports. The description is based on a template software project "mfctestdsp.pjt" that is used for testing the various hardware components on the MFC board.

The source files in the Analog Devices VisualDSP project "mfctestdsp.pjt" are listed in Table 1.

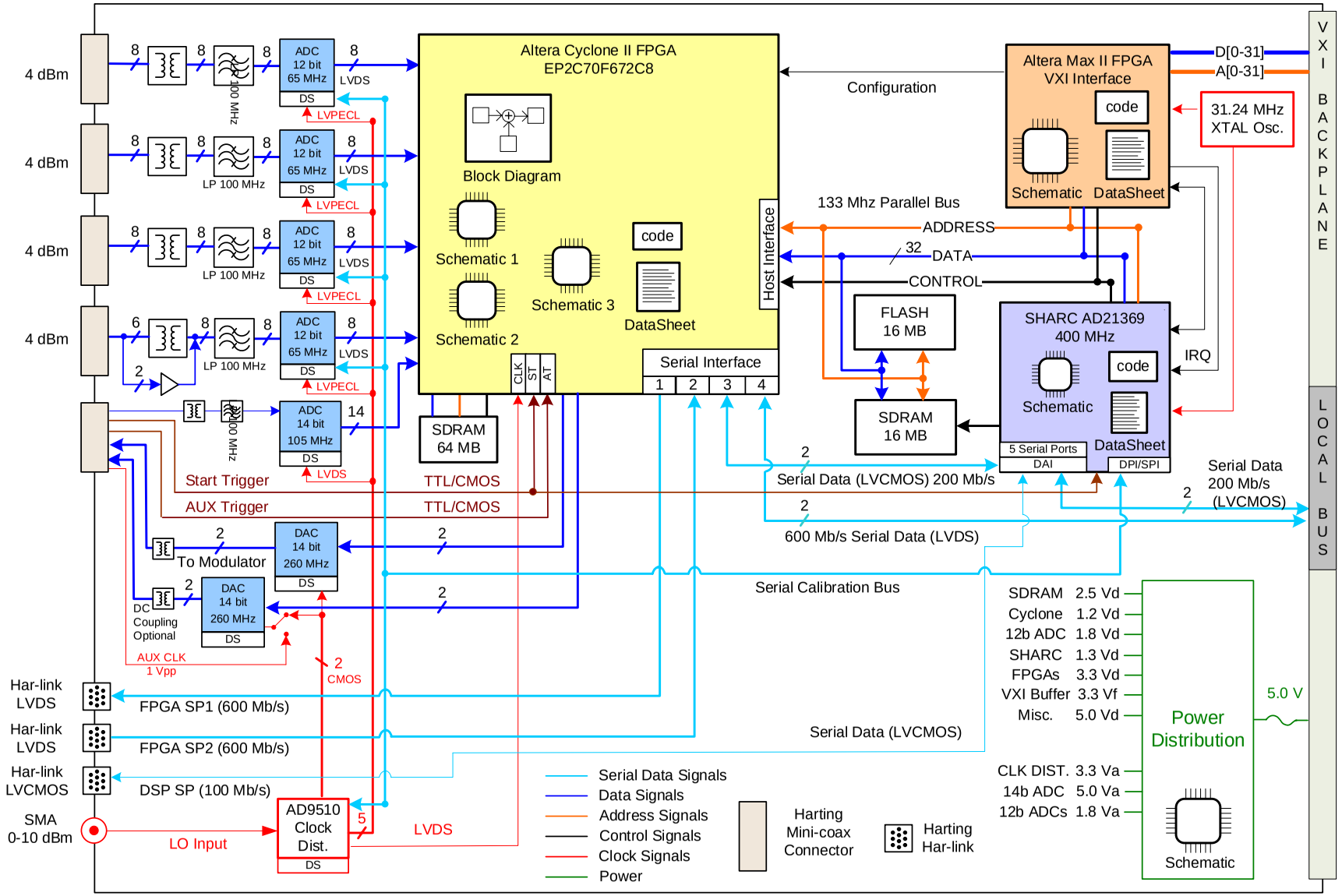
Assembly files	C files	Header files	Other Files
mfctest.asm	mfctest_main.c	mfctest.h	mfctest.ldf
mfctest_isr.asm	mfctest_init.c	mfctest_c.h	
mfctest_rth.asm	mfctest_initDAI.c		
	mfctest_initDPI.c		
	mfctest_initPLL.c		
	mfctest_SPI.c		
	mfctest_SPORT.c		

Table 1 Source files in DSP project "mfctestdsp.pjt"

Linker Description File

The linker description file "mfctest.ldf" specifies the locations for the various code and data segments within the four blocks of the DSP internal memory. This is shown graphically in Fig. 3. The DSP has a total of 2 Mb of SRAM memory addressable as 16, 32 or 48-bit words. DSP instructions are 48-bits long. All code segments are placed in Block0. The segments for program memory data and data memory data are placed in separate blocks (2 and 3) to make computations faster in the dual processing elements. There are separate address and data buses for program and data memory which allows for simultaneous access of dual data elements. The filter coefficients of an FIR filter could be placed in data memory data and the

Fig. 1 MFC Block Diagram



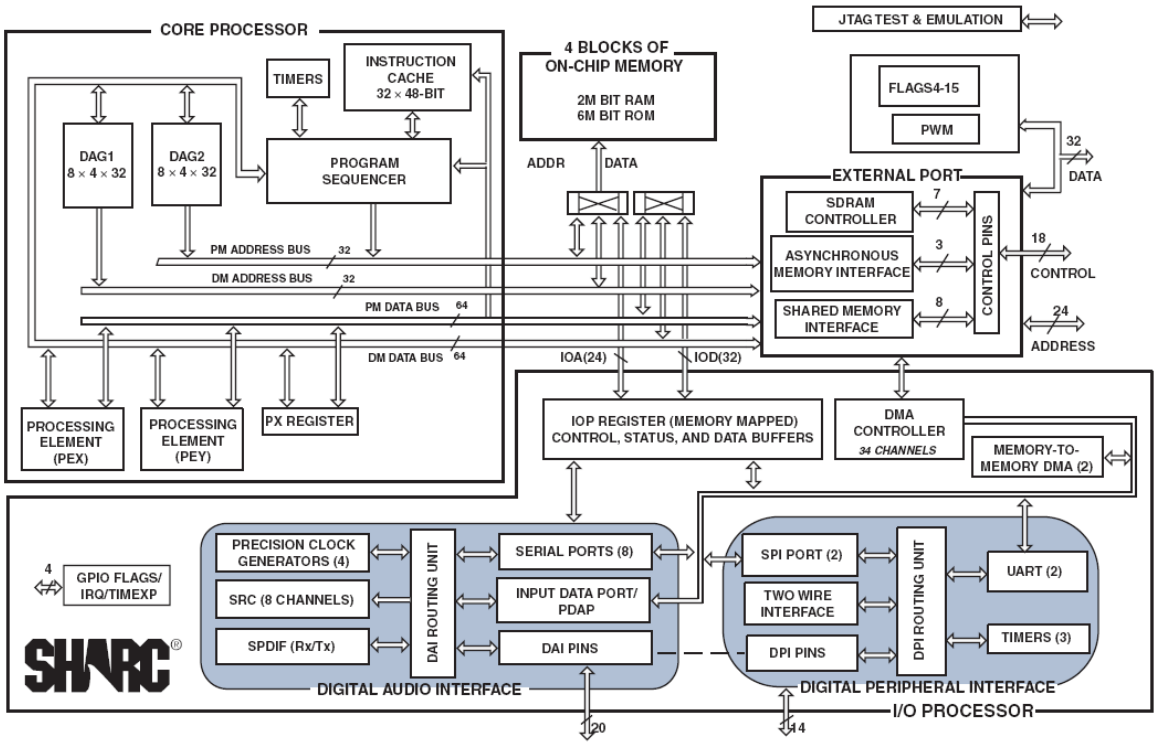


Fig. 2 ADSP-21369 Functional Block Diagram

Block 0 (0.75 Mb)	Block 1 (0.75 Mb)	Block 2 (0.25 Mb)	Block 3 (0.25 Mb)
seg_rth (256)	ftp_data (32)	seg_pm_jmp_tbl(128)	seg_heap (8,192)
seg_init (256)	wfm_data (4,096)	seg_pmda (8,064)	
seg_init_code(208)			
seg_init_code (15,568)	seg_dmda (12,256)		
	stack (256)		
	seg_stak (7,936)		
48-bit (16,384 Words)	32-bit (24.576 Words)	32-bit (8,192 Words)	32-bit (8,192 Words)

Fig. 3 DSP SRAM memory allocation

array of past input data values could be placed in program memory data. The 128 word segment “seg-pm_jmp_tbl” is used to store pointers to vector interrupt (VIRPT) functions which are used for DSP tasks initiated at interrupt priority from the slot0 and could be used for a variety of time critical and housekeeping tasks.

mfctest.asm

This file is used to locate any assembly functions in the project such as initialization routines. The DSP has four memory select lines which allow it to access the FPGA, its dedicated SDRAM memory, the Flash memory used for code storage and the MaxII VXI interface chip. Any variables placed in these memory segments may be used in programs like any other variable in the DSP internal memory – however it is necessary to use the NO_INIT option which allows the bootloader program to load the DSP memory from Flash without initialization of the external memory elements which could cause the loading process to hang. For example, in the powerup sequence for the MFC board, the FPGA is configured only after the DSP is running and therefore cannot be initialized when the code first starts executing. Initialzation can proceed after it is known that the devices are configured and therefore will respond to writes.

mfctest_isr.asm

Interrupt service routines in assembly for vector interrupts and other interrupts are placed here. IRQ0 is a hardware interrupt connected to the MaxII chip along with three flag lines IRQMux0,1 and 2 whose level indicates one of 8 possible interrupts, 5 of which are used presently. Two are used for bus control between the DSP and the MaxII. When slot0 is accessing the FPGA or the Flash chip, the DSP needs to relinquish bus control. This causes the DSP to disable its external port. When the data access is completed a second interrupt signals the DSP to resume bus mastership. Two more interrupts are used for read/writes to addresses in the DSP internal memory or the SDRAM dedicated to the DSP. In this case, both data and address are provided by the MaxII and the DSP completes the access by writing the data to the requested location or fetching the data and writing it to the MaxII for transfer to the slot0 controller. The last interrupt is dedicated to the VIRPT interrupts where any function from the VIRPT jump table may be called by the slot0 controller. Eight message registers are provided in the MaxII for passing arguments or reading results. Four VIRPT functions are provided with this project. The first is a test adder routine that returns the sum of the two integer inputs – this function is used as a basic check that the DSP is running and responding to requests. Two routines are provided to read and write the SPI addressable registers of the four ADC’s (9222) and the AD9510 clock chip. The FPGADataAcquire() function is used to initiate a data

acquisition process of the requested channel. In the present implementation, the DSP sends a trigger to start acquisition of 65k points, reads them in from the FPGA, formats it as 32 bit integers and stores it in its SDRAM for retrieval by the slot0 controller. This program, along with a Labview program running with backdoor access to the backplane, is used for testing all the 33 input channels of the MFC board.

mfctest_rth.asm

The run time header initializes the C – environment, sets up the interrupt vector table and calls the main() routine. Four instructions are provided for each interrupt in the table. These are used to push the status stack, switch context and jump to the ISR routine. ISR routines may be written in assembly or C. Depending upon which interrupt function is used in the C run time environment there could be considerable overhead in the ISR's written in C. The context switching function in the DSP core registers can be exploited to speed up the transfer of control during the interrupt handling and minimize the overhead. The approach used in this project is to have all assembly ISR's switch context to the alternate context. *It is important to note that the C runtime environment knows nothing about the alternate context.* Any assembly routines called from C could still use the primary context that the C environment is using. This makes it simple to pass arguments and read results in the core registers designated for this purpose (r4 – first parameter, r8 – second parameter, r12 – third parameter, stack for rest of parameters, r0 – 1 word return value, r1 – address of block with remaining results). When assembly routines are called from C, the stack operation must be handled by the called routine. This is done by using the "entry" and "exit" macros or the "leaf_entry" and "leaf_exit" macros at the beginning and end of the assembly routine.

For assembly ISR's use the "push sts" (status stack) at the beginning and the "pop sts" macros at the end to restore the context at the end of the interrupt. This is done automatically for the IRQ0,1 ,2 and the high priority timer interrupt TMZHI. For all other interrupts, it must be done manually. Context switching is done by setting the bits in the "mode1" register. IRQ1 interrupt is tied to the start trigger input of the MFC which can be used to run DSP tasks from an external hardware trigger.

mfctest_main.c

The function main() calls all the initialization routines by invoking the function initMFCBoard(). The function init_ASM_Stack() defines the stack area for assembly code. The final function called is the init_SPI_Devices() routine which loads the ADC and clock chip

registers with the user selected parameters. When all DSP initialization code has been completed, a flag is pulled low to indicate to the MaxII that the DSP is loaded and the configuration of the FPGA may proceed. Both the DSP program and the FPGA configuration is stored in the Flash and has to be retrieved over the shared parallel bus which makes it necessary to stagger the bootup of these devices.

mfctest_init.c

The DSP clkin input is connected to a 31.24 MHz crystal. The function initPLL_SDRAM() sets up the clk multiply and divide ratios to arrive at the core clock frequency. In the current project the core clock is set to 328 MHz. *It is important to note that when choosing multiply and divide factors for the internal PLL, the PLL frequency should not exceed 800 MHz.* The DSP may be run at 400 MHz core clock if the power supply to the DSP core is adjusted to 1.4 volts. The frequency for the SDRAM clock is set by a ratio of the core clock. A ratio of 2.5 sets this frequency to around 130 MHz. All external port transfers including the SDRAM and the asynchronous memory interface (AMI) are conducted with this clock which can go to a maximium of a 166 MHz. The parameters such as number of wait states, data bus width etc. are set here for the three AMI ports connected to the FPGA, MaxII and the Flash chip. The functions initDAI() and initDPI() are used to connect the desired DSP peripherals, such as serial ports, SPI ports, UARTS etc. to specific pins using the SRU macro. If a serial port needs to be changed from a transmitter to a receiver for example, the changes would be made here. These functions are expected to be run only during initialization. initSPI() and initSPORT() initialize the SPI ports and the serial ports.

Programming the Flash with DSP code

The MFC board can be used for development in a non-vxi environment by storing the FPGA code in the onboard serial configuration device and the DSP code in the Flash chip. Both can be loaded independent of each other in this configuration. The “MFCProgFlash” project takes a specified DSP loader file and writes it to the Flash. The loader file can be created by compiling the desired project in VisualDSP using the “MFCBootKernel.dxe “ as the user provided boot kernel which is customized for the MFC board. In the Project Options under the load tab the options to be specified are “boot type – Parallel”, “format – Include” and “width – 16 bit”. Slot 0 functions are available for writing DSP loaders and FPGA configuration files to the Flash which is useful for remote downloads.