ANALOG DEVICES

AN-338
APPLICATION NOTE

ONE TECHNOLOGY WAY • P.O. BOX 9106 • NORWOOD, MASSACHUSETTS 02062-9106 • 617/329-4700

Loading an ADSP-2101 Program via the Serial Port

by Gerald McGuire

INTRODUCTION

For many DSP applications, it is desirable to have a DSP processor under the control of a host computer. In these situations, the host computer would download a program for the DSP to execute. The ADSP-2101 provides two serial ports suitable for program download from a host computer. This application note details the ADSP-2101 monitor program for downloading from a serial port. The monitor program itself would be booted from EPROM or other boot memory. While this example uses serial port zero, the principal could be extended to download via a memory-mapped parallel port.

A MONITOR

The task of the host computer is to download a series of instructions to the ADSP-2101 for execution. The ADSP-2101 receives the incoming instructions, loads them into program memory and when all instructions have been received, executes them. Prior to and during the download from the host, the ADSP-2101 executes a monitor program. This monitor activates the serial port, receives the instructions and places them in program memory for execution.

The ADSP-2101 instruction is twenty-four bits wide but many hosts, including eight-bit processors, more readily handle byte-wide data. Since the serial port can accommodate serial words from three to sixteen bits in length, byte-length data words are easily received.

Whenever a program memory write occurs, the sixteen most significant bits are supplied by the source register, explicitly named in the instruction, and the eight LSBs are supplied by the PX register. The basic tactic of the monitor program is to assemble the two most significant bytes in a data register (using the Shifter) and load PX explicitly with the least significant byte. A program memory write then writes the correct twenty-four bit instruction.

In addition to the transfer of instructions through the serial port into program memory, the monitor program must also know when the download is complete and execution can begin. A straight forward method is to count the number of instructions sent to the serial port. A count value is sent to the ADSP-2101 before the first instruction. This is the count of the instructions to follow. After each instruction is downloaded, the count can be decremented.

The downloaded program must avoid overwriting the monitor program while the monitor executes. The last instruction of the monitor program is identified by a global label which also identifies the beginning of the available space for downloaded code. The monitor program must be linked with the downloaded program so that the downloaded program makes the correct address references including the reference to this global label.

The indirect addressing capabilities of the Data Address Generators on the ADSP-2101 make it easy to cycle through the correct sequential locations starting with the label.

The final concern is the interrupt table. If the downloaded program is interrupt-driven, the interrupt table (program memory H#0000 to H#001C) must contain valid instructions for servicing expected interrupts.

There are several ways to do this. First, the monitor program itself could contain the valid interrupt table for the program to be downloaded. This assumes that the interrupt structure of the downloaded program is known when the monitor program is created. Second, the interrupt table may be downloaded through the serial port just as the rest of the program is. The DAG can loaded with the start address of the interrupt table and the instructions can be loaded, but you may not overwrite the interrupt being used to receive the data on the serial port until all instructions have been received.

The monitor program example does not load an interrupt table. The best approach is dependent on your application.

IMPLEMENTATION

The first task of the monitor program is to setup and enable the

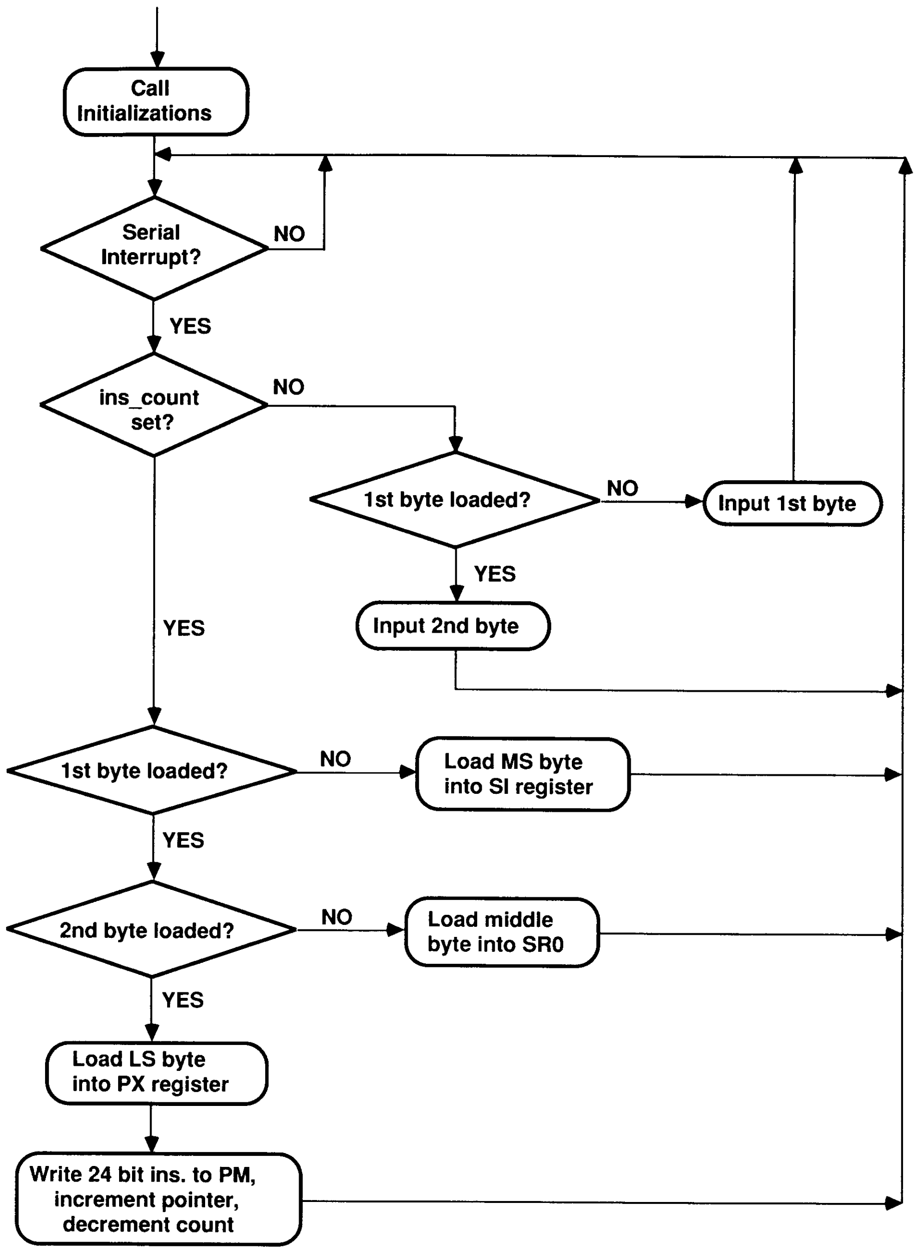

Figure 1. Boot Program Flow Diagram

serial port. Serial ports on the ADSP-2101 are extremely flexible in terms of framing options, word lengths and timing. The ADSP-2101 serial ports may receive the frame synch and serial clock from the host processor or generate them internally.

As the program is downloaded from a host computer, the ADSP-2101 looks to the host for serial port information. That is, the serial port frame synchronization and serial port clock are supplied by the host computer. For purposes of illustration, the code that appears at the end of this application note uses normal framing and external receive frame synchronization. For externally generated serial clocks the ADSP-2101 can support frequencies up to the processor instruction rate.

The flow for the monitor program is shown in Figure 1.

Once the serial ports are enabled, the monitor program waits for a serial port interrupt signifying that a serial word has been received. The first two serial words received are the instruction count. As the serial word is eight bits, two serial port words make up the instruction count. The separate bytes of the instruction count are combined in the shifter and loaded into data memory. This count represents the number of instructions to be downloaded from the host and does not include the interrupt table. The interrupts are handled automatically, as the interrupt table has a fixed length.

With the count downloaded, the ADSP-2101 is ready to accept instructions through the serial port. Instructions are downloaded a byte at a time just as the instruction count was. The most significant byte is first. It is loaded into the SI register and the byte count ("count") is decremented. The middle byte of the instruction is loaded into the SR0 register. These two bytes are combined in the shifter with the results residing in the SR0 register. Once again the byte count is decremented. Finally, the least significant byte is loaded into the PX register. Now that all three bytes are loaded into registers on the ADSP-2101, the downloaded instruction can be written to program memory.

When all is downloaded, a jump to the new downloaded program is all that is necessary to begin execution.

The monitor program is listed at the end of this note.

SUMMARY

A monitor program initializes the serial port and receives instructions, writing them into program memory, then beginning execution. This method of booting is useful when the ADSP-2101 is under the control of a host computer or controller. Any size program may be downloaded (up to the full addressing capability of the ADSP-2101) with this particular method of implementation. Only the program memory used by the monitor program cannot be loaded. That space could, however, be used for program memory data storage.

```
.MODULE/RAM/BOOT=0                          serial_boot_monitor;
.VAR/DM                                     count;              {counts bytes}
.VAR/DM                                     ins_count;          {counts instructions}
.GLOBAL                                     code_start;         {end of monitor space}


                      JUMP restarter; NOP; NOP; NOP;            {restart vector}
                      RTI; NOP; NOP; NOP;                       {IRQ2 not used}
                      RTI; NOP; NOP; NOP;                       {sport0 TX not used}
                      JUMP serial; NOP; NOP; NOP;               {sport0 RX }
                      RTI; NOP; NOP; NOP;                       {sport1 TX not used}
                      RTI; NOP; NOP; NOP;                       {sport1 RX not used}
                      RTI; NOP; NOP; NOP;                       {no timer used}

restarter:            CALL initializations;
wait_loop:            IDLE;
                      JUMP wait_loop;

initializations:      I4 = H#3ff3;                              {pointer to mem map reg}
                      I5 = ^code_start;                         {pointer to start of prog}
                      M4 = 0;
                      M5 = 1;
                      L4 = 0;
                      L5 = 0;
                      SR0 = 0;
                      SR1 = 0;

                      AX1 = 1;
                      DM(count) = AX1;                          {count val for # of bytes}
                      DM(I4,M5) = 0;                            {disable autobuffer}
                      DM(I4,M5) = 0;                            {no frame divide modulus}
                      DM(I4,M5) = 0;                            {no clk divide modulus}
                      DM(I4,M5) = H#2007;                       {extrnl RFS & SCLK, no compand}
                                                                {SLEN 8, no multichannel}
                      AX0 = H#1000;
                      DM(H#3fff) = AX0;                         {enable sport0}
                      AX0 = H#FFFF
                      DM(ins_count) = AX0;

                      IMASK = 8;                                {sport0 rec interrupt only}
                      RTS;

serial:               AY1 = DM(ins_count);
                      AR = PASS AY1;
                      IF GT JUMP next_instruction;              {get next instruction}
                      IF LT JUMP load_word_count;               {get number of instructions}
                      IF EQ JUMP code_start;                    {start downloaded program}
```

```
{load the count, that is, the number of instructions to be downloaded}
{this happens in two bytes  The most significant byte first}


load_word_count:        AY0 = DM(count);                         {is this 1st or 2nd byte}
                        AR = PASS AY0;
                        IF NE JUMP first_byte;
                        IF EQ JUMP second_byte;

first_byte:             SI = RX0;                                {first byte decrem. count}
                        AR = AY0 - 1;
                        DM(count) = AR;
                        RTI;

second_byte:            SR0 = RX0;                               {second byte...}
                        SR = SR OR LSHIFT SI BY 8 (LO);          {put two bytes together}
                        DM(ins_count) = SR0;                     {store in ins_count}
                        AX0 = 3;
                        DM(count) = AX0;                         {load count for ins.}
                        RTI;


{load the next instruction.  Instructions are 24 bits long and appear}
{at the serial port in 8 bit fragments.  The most significant byte 1st}

next_instruction:       AX0 = 2;                                 {decide which byte is due}
                        AY0 = DM(count);
                        AR = AX0 - AY0;

                        IF LT JUMP most_sig_byte;
                        IF EQ JUMP middle_byte;
                        IF GT JUMP least_sig_byte;

most_sig_byte:          SI = RX0;                                {load MS byte into SI}
                        AR = AY0 - 1;                            {decrement count}
                        DM(count) = AR;
                        RTI;

middle_byte:            SR0 = RX0;                               {load Middle into SR}
                        SR = SR OR LSHIFT SI BY 8 (LO);          {put MS and middle together}
                        AR = AY0 - 1;                            {decrement count}
                        DM(count) = AR;
                        RTI;

least_sig_byte:         PX = RX0;                                {put LS byte into PX}
                        PM(I5,M5) = SR0;                         {write SR0 into PM}
                                                                 {PX provides 8 LS bits}
                        AX0 = 3;
                        DM(count) = AX0;                         {reset byte count}
                        AR = AY1 - 1;                            {decrement ins count}
                        DM(ins_count) = AR;
                        RTI;

code_start:             NOP;

.ENDMOD;
```