Hardware Examples 13

13.1 OVERVIEW

This chapter describes some hardware examples of circuits that can be interfaced to the ADSP-21xx serial ports, host interface port (HIP), or the memory port. As with any hardware design, it is important that timing information be carefully analyzed. Therefore, the data sheet for the particular ADSP-2100 family processor used should be used in addition to the information presented in this chapter.

13.2 BOOT LOADING FROM HOST USING BUS REQUEST & GRANT

All ADSP-2100 family processors that have internal program memory RAM support boot loading. With boot loading, the processor reads instructions from a byte-wide external memory device (usually an EPROM) over the memory interface and stores the instructions in the 24-bit wide internal program memory. Once the external memory device is set up to provide bytes in the proper order, the boot operation can run automatically and transparently at reset or when forced in software. See Chapter 10, "Memory Interface."

In some systems where the ADSP-21xx is controlled by a host processor, it is necessary to boot the DSP directly from the host. In this case the host, rather than an EPROM, is the source of bytes to be loaded into on-chip memory. If the ADSP-21xx has a host interface port (such as the ADSP-2111), it can perform automatic boot loading through this port. If the processor does not have a host interface port, however, it can still boot through the memory interface using the bus request signal, as described below.

This example shows a simple way to download programs from a host processor to the internal program memory of an ADSP-21xx. There are several techniques for connecting a DSP processor to a host. The choice of which technique to use depends upon the I/O structure of the host, availability of I/O port lines, and the amount of address decoding logic already available in the system.

Figure 13.1 illustrates a minimal system implementation to allow a microcontroller to boot an ADSP-21xx. The only hardware required is a D-type flip-flop and a 5 kΩ resistor. The resistor is used to pull the ADSP-21xx's $\overline{BMS}$ pin (Boot Memory Select) high.

The ADSP-21xx automatically enters its booting sequence after the processor is reset (when the MMAP pin is tied low) or when software initiates a reboot operation. When the ADSP-21xx begins to fetch a byte from external boot memory (in this case, the host processor), it asserts $\overline{BMS}$. When $\overline{BMS}$ goes low, the flip-flop is preset and the $\overline{Q}$ output brought low. This low signal asserts $\overline{BR}$ (bus request) on the ADSP-21xx. When bus request is recognized by the ADSP-21xx, the current execution cycle is allowed to finish and then processor operation is suspended. The ADSP-21xx then asserts $\overline{BG}$ (bus grant) in the next cycle (after $\overline{BR}$ is recognized).

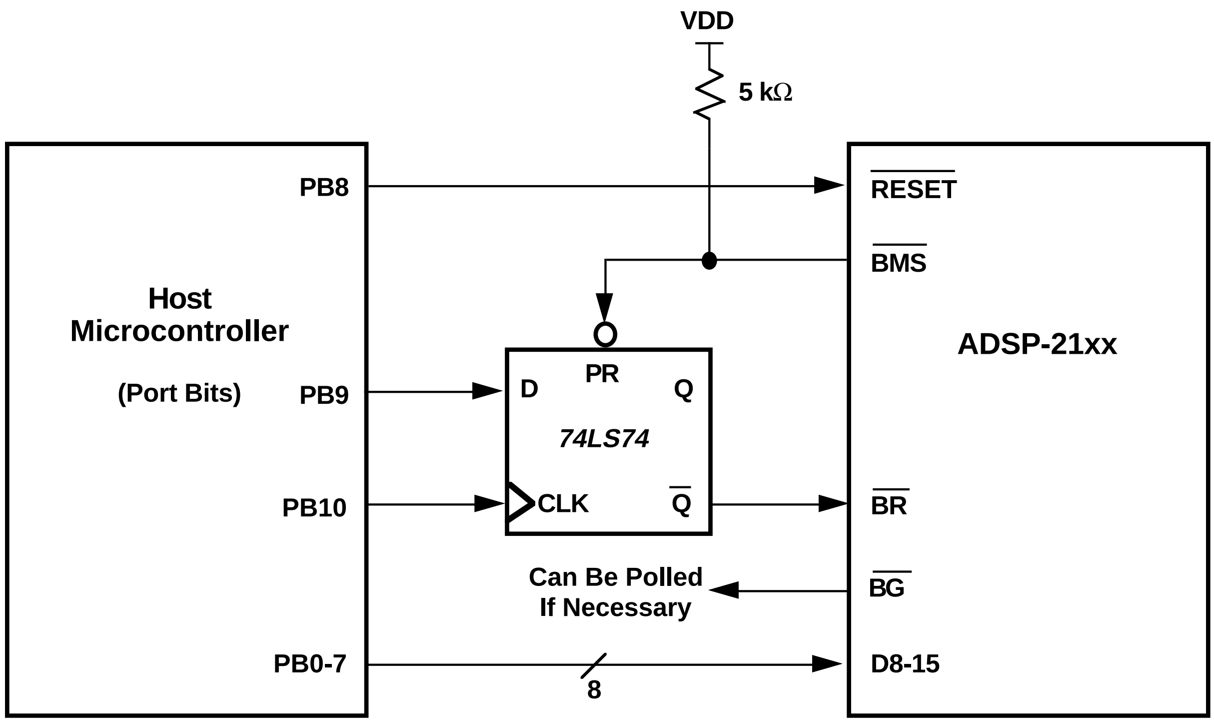

Figure 13.1 ADSP-21xx Booting From Host

When a low-level signal at the D input is clocked into the flip-flop, the $\overline{Q}$ output is brought high, deasserting $\overline{BR}$.

The bus request pin ($\overline{BR}$) of the ADSP-21xx is used to stop and synchronize the booting process. The host releases bus request, causing the ADSP-21xx to read one byte of boot data. During the read operation the $\overline{BMS}$ pin is asserted, which in turn causes the $\overline{BR}$ pin to be asserted and the ADSP-21xx to be put back into a bus request state. The ADSP-21xx remains suspended, waiting for the next byte of boot data.

Three programmable port bits of the microcontroller (PB 8-10) are used to provide the handshake mechanism for the transfer of each byte of boot data. Alternately, PB9 and PB10 could be implemented as a memory-mapped port location. PB8 is used to bring the ADSP-21xx out of reset, starting the boot process. Note that if PB8 is not low at power-up, the ADSP-21xx will start executing undefined instructions until PB8 is brought low.

The boot data is presented by the microcontroller either through 8 port bits (PB0-7) or through a memory-mapped port. The PB0-7 bits should be put into a high-impedance state after the boot is complete, to prevent bus contention if the ADSP-21xx tries to write to external memories or peripherals.

A typical boot sequence for this system is as follows:

1.) Bring PB8 low to reset the ADSP-21xx.

2.) Clock a high state into the flip-flop with PB9 and PB10 to bring $\overline{BR}$ low.

3.) Bring PB8 high to bring the ADSP-21xx out of reset.

4.) Place a byte of boot data on the data bus (PB0-7.).

5.) Clock a low state into the flip-flop with PB9 and PB10 to bring $\overline{BR}$ high.

6.) Wait a minimum of six processor cycles while the ADSP-21xx fetches the data byte and the flip-flop asserts $\overline{BR}$.

7.) Repeat steps 4, 5, and 6 for each byte of boot data. After the last iteration, the ADSP-21xx will automatically start execution.

Note: The proper loading sequence for boot data must be followed (i.e. the order in which the host passes bytes to the ADSP-21xx). This sequence is described in the Chapter 10, “Memory Interface.” To create a file for booting, use the PROM Splitter utility of the ADSP-2100 Family Development Software. The PROM Splitter automatically organizes the bytes in the proper order for booting.

13.3 SERIAL PORT TO CODEC INTERFACE

A codec (COder/DECoder) incorporates analog-to-digital conversion, digital-to-analog conversion, and filtering in one device. The codec shown in this example also performs pulse-code modulation (PCM) encoding and decoding according to the CCITT μ-law standard. PCM compresses digital data so that fewer bits are needed to store the same information. The ADSP-21xx serial ports have both μ-law and A-law companding (compressing/expanding) capability.

In the example described here, a codec converts its analog input to digital data, compresses it and sends it serially to the SPORT on an ADSP-21xx processor. At the same time, the processor sends compressed serial data via the SPORT to the codec, which expands the data and converts the result to an analog signal.

13 Hardware Examples

Figure 13.2 shows an industry standard µ-law companding codec connected to a serial port (in this case, SPORT0) on an ADSP-21xx processor. The codec's analog input at VFXI+ is internally amplified by a gain which is controlled by the resistor combination at GSX and VFXI–. The gain is

$$20 \times \log (R1 + R2)/R2$$

in this case, 20 log 2.

The ADSP-21xx controls codec operation by supplying master and bit clock signals. In the configuration shown, the codec transmit and receive sections operate synchronously. MCLKR and MCLKX are the master clocks for the receive and transmit sections of the codec, respectively. BCLKX is the bit clock and in this configuration is used for clocking both received and transmitted serial data. MCLKR, MCLKX and BCLKX must be synchronous and in this case they are the same signal, namely the SCLK0 output generated by the ADSP-21xx processor. The BCLKR/CLKSEL input, tied low, selects the frequency of MCLKX to be 2.048 MHz. The ADSP-21xx must be programmed for internal SCLK0 generation at 2.048 MHz.

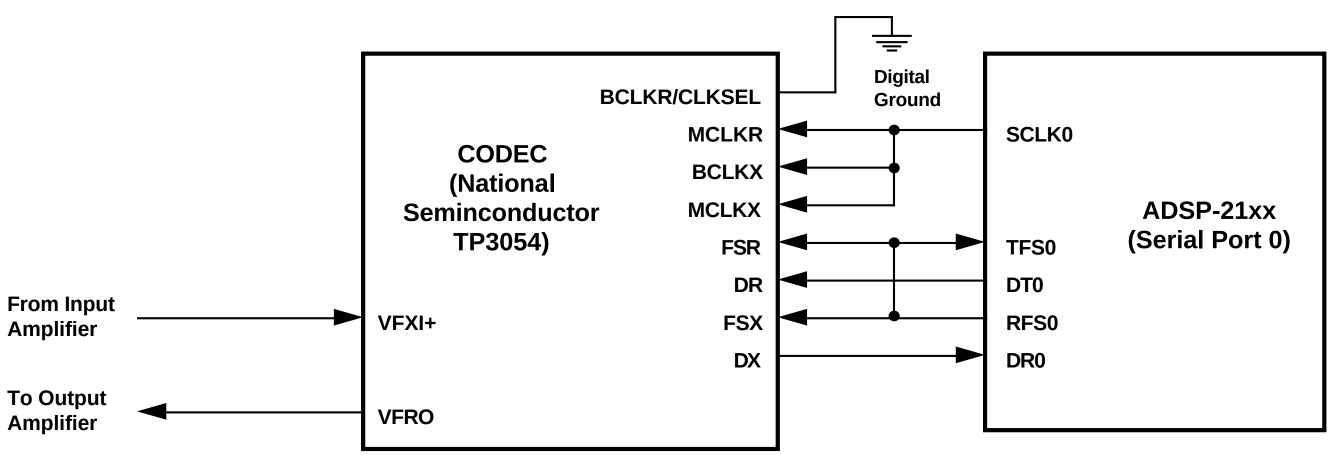

Figure 13.2 ADSP-21xx Serial Port (SPORT0) To CODEC

The processor uses frame synchronization signals to tell the codec to send and receive data. To transmit data to the codec, it sends a TFS0 pulse to the FSR input of the codec and then outputs the eight bits on DT0 on the next eight serial clock periods. The codec receives the data on its DR input. Likewise, the processor initiates a data receive operation by sending an RFS0 pulse to the codec's FSX input, which causes the codec to output eight bits on its DX output on the next eight serial clock periods. The processor receives the data on its DR0 input. The ADSP-21xx must be programmed to use normal framing, 8-bit data words, and internal, active-high frame sync generation.

The ADSP-21xx code shown in Listing 13.1 configures SPORT0 for operation as required in this example:

- Internally generated serial clock
- 2.048 MHz serial clock frequency
- Both transmit and receive frame syncs required
- Use normal framing for both transmit and receive
- Internally generated transmit and receive frame syncs
- Both frame syncs active high
- Word length of eight bits
- μ-law companding

This code assumes the processor operating at 12.288 MHz. The code also sets up the processor to request data from the codec at an 8 kHz rate (this register is not initialized at reset and should always be written before the SPORT is enabled if RFS is generated internally). The processor transmits data as needed by the program it is executing.

```
AX0=0x6927;       {Int SCLK, RFS/TFS req, norm framing,}
DM(0x3FF6)=AX0;   {generate RFS, active HI, Mu-law, word length 8}

AX0=2;            {value of SCLKDIV for 2.048 MHz}
DM(0x3FF5)=AX0;   {with a 12.888 MHz CLKOUT}

AX0=255;          {RFSDIV=256, 256 SCLKs between}
DM(0x3FF4)=AX0;   {frame syncs, 8 kHz framing}

AX0=0x1038;       {enable SPORT0 only, leave defaults}
DM(0x3FFF)=AX0;
```

Listing 13.1 Serial Port Initialization Example

13.4 SERIAL PORT TO DAC INTERFACE

Any DSP process must ultimately output analog information. The serial port of the ADSP-21xx processors can send data directly to a DAC (digital-to-analog converter) for conversion to an analog signal.

Analog Devices' AD766 is a DAC that requires no extra logic to interface to the SPORT. The AD766 receives 16-bit data words serially, MSB first, which it then converts to an analog signal. Its digital interface consists of three inputs: DATA, the serial data input; $\overline{\text{CLK}}$, for clocking data into the DAC (active low because data is clocked on the falling edge) and LE (latch enable), which latches each 16-bit word into the conversion section of the DAC.

The serial port connection to the AD766 is shown in Figure 13.3. In this configuration, the processor generates SCLK internally and provides it to the DAC. Serial data is output from the DT pin to the DATA input of the DAC. The TFS signal provides the DAC's LE input.

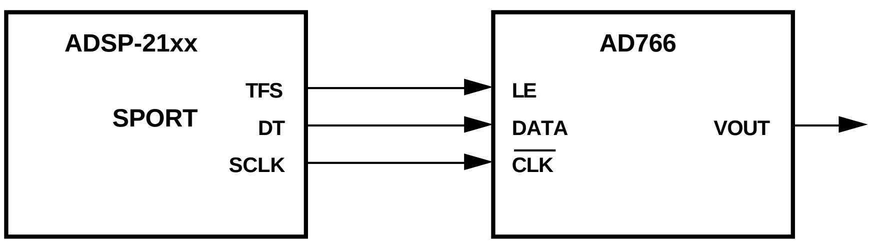

Figure 13.3 Serial Port Interface To AD766 DAC

LE should go low on the clock cycle after the LSB (sixteenth bit) of a word is transmitted, to latch the 16-bit word into the DAC. To provide this timing, TFS is configured for the alternate framing mode, non-inverted; it goes high when the first bit is transmitted and low after the last bit is transmitted. This low-going edge latches the word into the AD766. The only restriction is that the SPORT cannot transmit continuously; there must be a break in between the last bit of one word and the first bit of the next so that TFS can go low. Figure 13.4 shows the timing.

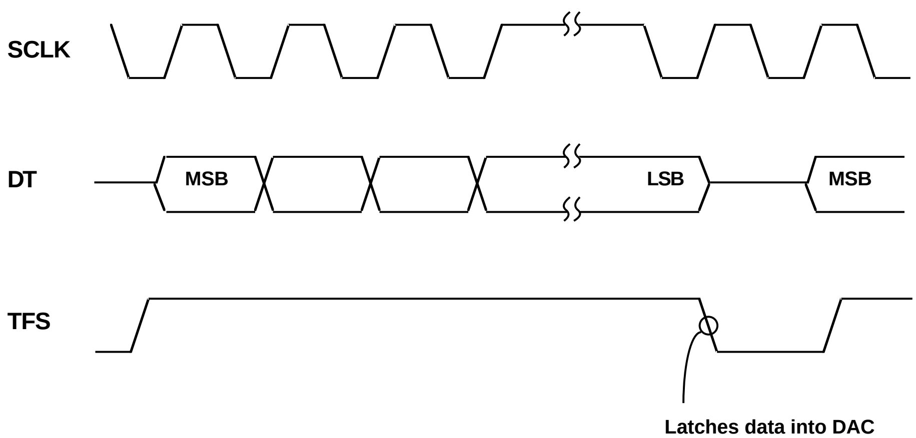

Figure 13.4 SPORT To AD766 DAC Timing

The configuration of the SPORT control registers for this application is shown in Figure 13.5.

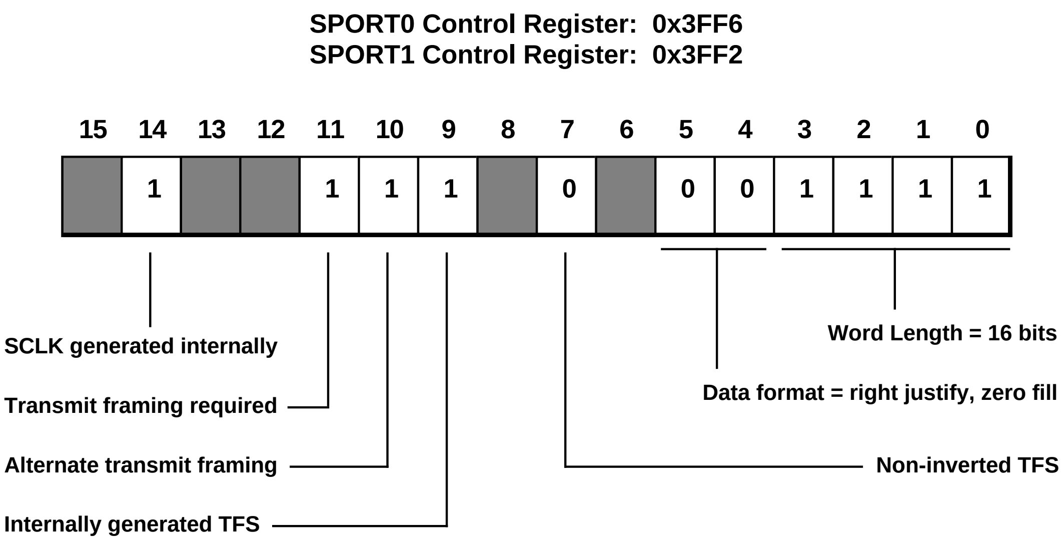

Figure 13.5 SPORT To AD766 DAC Control Register Settings

13.5 SERIAL PORT TO ADC INTERFACE

An ADC (analog-to-digital converter) converts an analog signal to digital samples that a DSP processor can operate on. The ADSP-21xx processors can receive data from an ADC directly through a serial port.

Analog Devices' AD7872 is an ADC that requires no extra logic to interface to the SPORT. The AD7872 converts an analog signal to 14-bit samples. Each sample is padded with two zero MSBs to yield 16-bit samples. The AD7872 outputs each sample serially, MSB first. Its digital interface consists of three pins: SDATA, the serial data output; SCLK, for clocking data out; and $\overline{\text{SSTRB}}$, (serial strobe), which frames each serial word.

The serial port connection to the AD7872 is shown in Figure 13.6. The timer regulates sampling via the $\overline{\text{CONVST}}$ input at a constant frequency. Instead of the timer, an unused serial clock or flag output from the ADSP-21xx processor can be programmed to generate the $\overline{\text{CONVST}}$ signal. The AD7872 generates SCLK internally and provides it to the processor. With the CONTROL input held at –5 V, the SCLK signal is continuous, running even when no data is being output.

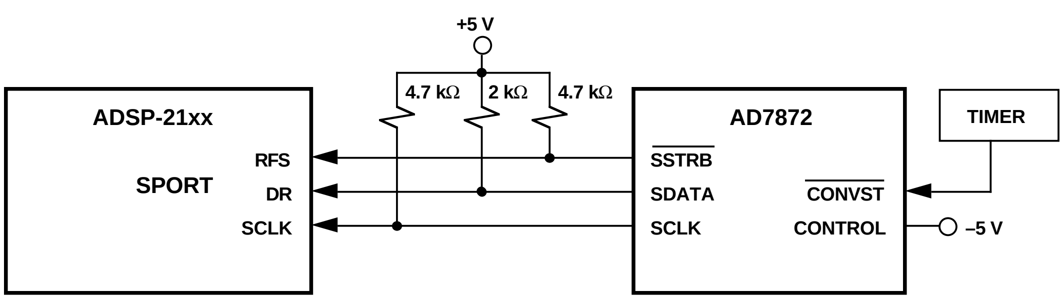

Figure 13.6 Serial Port Interface To AD7872 ADC

Serial data is output from the SDATA output of the ADC to the processor's DR pin. The $\overline{\text{SSTRB}}$ signal provides the RFS input to the processor. $\overline{\text{SSTRB}}$ goes low when the first bit is transmitted to the processor. Figure 13.7 shows the timing of the serial data transfer.

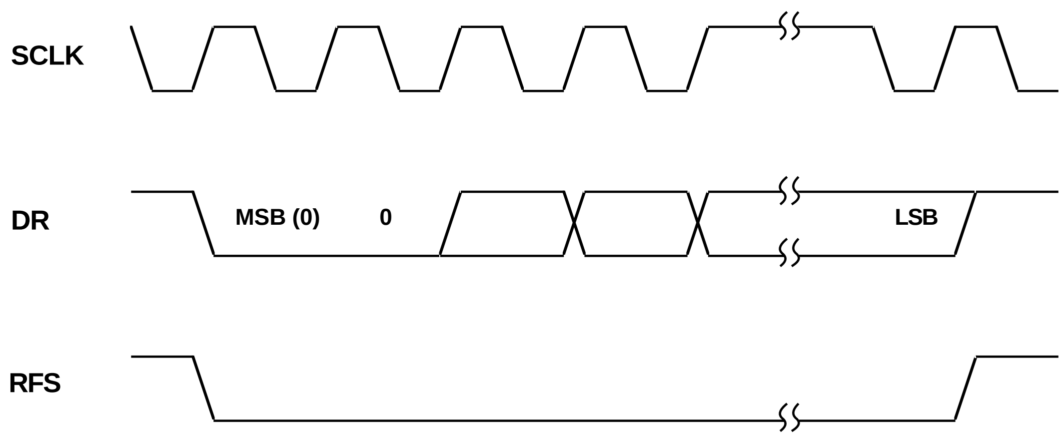

Figure 13.7 SPORT To AD7872 ADC Timing

RFS is configured for the alternate framing mode, externally generated, with inverted (active low) logic. The SPORT must also be programmed for external serial clock and a serial word length of 16 bits. The configuration of the SPORT control register for this application is shown in Figure 13.8.

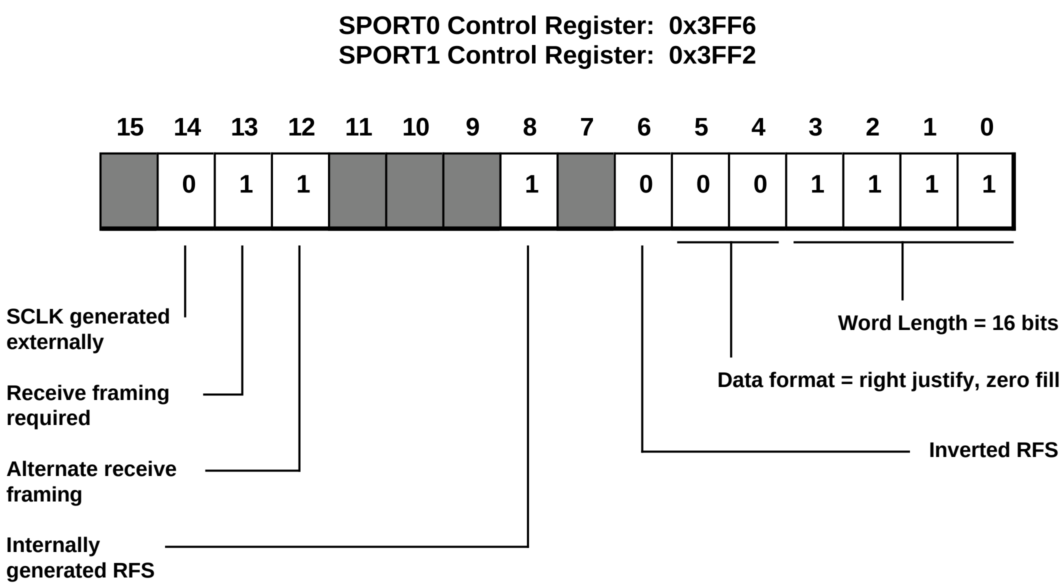

Figure 13.8 SPORT To AD7872 ADC Control Register Settings

13.6 SERIAL PORT TO SERIAL PORT INTERFACE

The serial ports provide a convenient way to transfer data between ADSP-21xx processors without using external memory or the memory bus and without halting either processor. The serial ports are connected as shown in Figure 13.9—in this example, SPORT1 of processor #1 is connected to SPORT0 of processor #2.

The serial clock used by both processors is generated internally by processor #1. Processor #2 is configured to receive its serial clock externally. The serial port control registers should be set up with the following parameters.

Processor 1, SPORT1	*Processor 2, SPORT0*
SCLKDIV = system-dependent	SCLKDIV = system-dependent
SLEN = system-dependent	SLEN = system-dependent
ISCLK = 1	ISCLK = 0
TFSR = 1	TFSR = 1
RFSR = 1	RFSR = 1
IRFS = 0	IRFS = 0
ITFS = 1	ITFS = 1
RFSDIV = don't care	RFSDIV = don't care

TFSW1 = RFSW1 = TFSW2 = RFSW2 = system-dependent
INVRFS1 = INVTFS1 = INVRFS2 = INVTFS2 = system-dependent

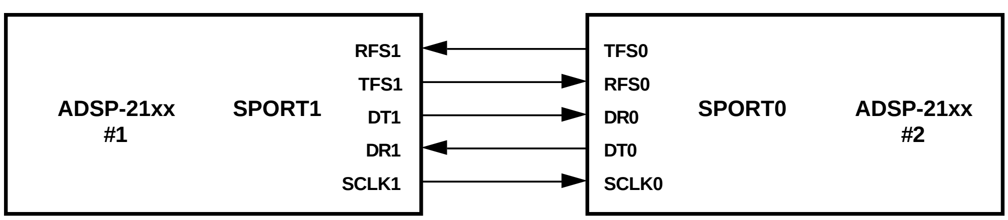

Figure 13.9 Serial Port Interface Between Two ADSP-21xx Processors

Frame synchronization is used to coordinate the transfer of serial data. Each processor generates a transmit frame sync (TFS) signal internally and expects to receive its receive frame sync (RFS) signal externally, from the other processor. The framing mode can be normal or alternate, but must be the same for both SPORTs. Likewise, the SPORTs must be configured for the same serial word length and companding type, if companding is used, or data format if companding is not used.

The autobuffering capability of the serial ports can be used in this configuration to transfer an entire buffer of data from the data memory space of one processor to the other's, *without interrupt overhead*. The serial ports handshake automatically—when one processor writes its' TX0 register, the data is automatically transmitted to the other processor's RX0 register and an autobuffer cycle is generated.

In fact, autobuffer transfers can occur in both directions at the same time, in the background, while each processor is executing some other primary function. Each SPORT will generate an interrupt when the autobuffer transfer is complete. The description of autobuffering in the Serial Port chapter shows an example of the code for setting up autobuffering.

13.7 80C51 INTERFACE TO HOST INTERFACE PORT

The host interface port (HIP) on the ADSP-2111, ADSP-2171, and ADSP-21msp5x processors facilitates communication with a host microcomputer such as the Intel 80C51. An example connection is shown in Figure 13.10. In this example, the HIP data registers (HDRs) and HIP status registers (HSRs) of the ADSP-2111 occupy eight contiguous locations in the memory space of the 80C51.

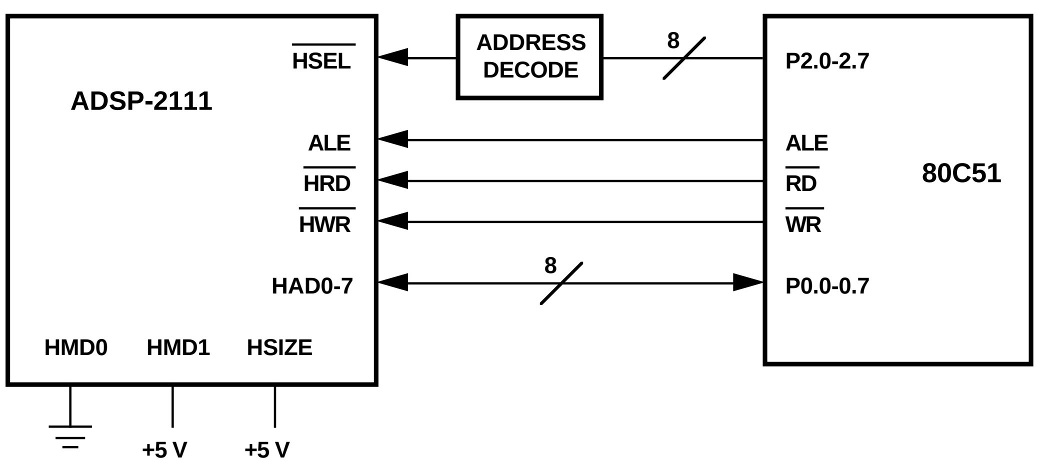

Figure 13.10 Host Port Interface to 80C51 Microcomputer

To access one of the HIP registers, the 80C51 asserts ALE and outputs a 16-bit address, with the upper half on P2.0-2.7 and the lower half on P0.0-0.7. The upper half is decoded to select the HIP via $\overline{\text{HSEL}}$, and the lower half selects the HIP register via HAD0-7. The ALE assertion causes the HIP to latch the address so that the 8-bit data can then be transferred on the HAD0-7 lines. The 80C51 asserts $\overline{\text{WR}}$ for a write or $\overline{\text{RD}}$ for a read.

In this example, the 80C51 reads and writes 8-bit data, so the ADSP-2111's HSIZE input is tied high. Only the lower eight bits of each HIP register are used. HMD0 is tied low because the 80C51 uses separate read and write strobes rather than a single Read/$\overline{\text{Write}}$ line. HMD1 is tied high because the address and data use the same bus (time-multiplexed using ALE) rather than separate buses.