Technical notes on using Analog Devices DSPs, processors and development tools
Contact our technical support at dsp.support@analog.com and at dsptools.support@analog.com
Or visit our on-line resources http://www.analog.com/ee-notes and http://www.analog.com/processors

Configuring the Signal Routing Unit of ADSP-2126x SHARC® DSPs

Contributed by K. Malsky

Rev 1 – February 12, 2004

Introduction

The ADSP-2126x family of SHARC® DSPs is capable of interfacing with a wide variety of peripherals. Much of this versatility comes from the processor's "soft" connections between the I/O ports and the physical package pins. When most processors are designed into real-world systems, many device pins are tied high or low, pulled up, pulled down, or left unconnected. A complex system often has many input pins with fixed or default values that must be hard-wired and unused outputs pins. The Signal Routing Unit (SRU) on an ADSP-2126x DSP is a software-controlled matrix that can eliminate the need for pins that do not serve any true I/O purpose.

The SRU provides maximum flexibility by allowing you to define the function of the 20 pins of the digital audio interface (DAI). However, this flexibility brings complexity that can be overwhelming when beginning a new design. This document provides guidance for engineers who are starting their first project using the SRU and the DAI pins. It offers helpful hints and tricks that may assist experienced users.

Getting Started

The SRU, which is documented in the *ADSP-2126x SHARC DSP Peripherals Manual* [1], can be somewhat difficult to approach. By its nature, any connection matrix requires a clear understanding of what is being connected. The naming convention for these endpoints is very consistent, but frequently counterintuitive. In an attempt to make the nomenclature more intuitive, we'll begin by using familiar terms and focusing on the outside of the processor.

Step 1: Take Inventory of the Unique Signals

As mentioned above, only signals that actually provide information to and from peripherals need to be connected to the SHARC DSP. Since there is a means of routing within the DSP, signals need only to be connected to a single pin, regardless of the number of internal places the signal is used. You do not have to connect the same signal to two or more DAI external pins.

Identify the peripherals that you are trying to connect to the SHARC DSP, and count the unique signals. If the same clock or frame sync is connected to multiple devices, it counts as one signal. When a serial data stream drives multiple output devices, it also counts as a single signal. List the unique I/O signals and look carefully to see what else you may be able to eliminate.

For example, if you find two signals are identical, but of opposite polarity (inverted), count them as one signal, as the SRU can generate either from the other. If a clock signal is a phase-aligned, integer sub-multiple of another clock signal, group them together. The Precision Clock Generator (PCG) is a peripheral within the DAI that may allow you to connect only the fastest clock. For example, if there is a clock at frequency *f* and another at *f/8*, it is likely that only the faster clock needs a DAI pin. Read

about the PCG in the *ADSP-2126x SHARC DSP Peripherals Manual* [1] for details.

Step 2: Note the Direction of Signal Flow

Next to each signal in your list, indicate whether it should be an input to the pin buffer, an output from the pin buffer, or bidirectional.

The external connection to the SHARC DSP DAI pins (the wire lead or ball) is part of a peripheral known as a pin buffer. Pin buffers will be explained in detail in the next section. For now, just think of them as I/O pins on the SHARC with programmable behavior.

Most pin buffers are used only in one direction in a given design. Note that many peripherals have pins that are capable of being bidirectional, but are only used in one direction in the system. When a pin buffer is unidirectional, programming the SRU is dramatically simplified.

In cases where the pin is bidirectional, determine what causes the direction to change. Is it the state of another pin? Is it the state of a processor-level control register? Is it the software configuration of a port? Think about what may be controlling when the SHARC is driving a logic value onto the bidirectional pin and when the pin is just reading a logic input.

Step 3: Allocate the DAI Pins

At this point, it is likely that you will have reduced your list to 20 or fewer signals. If you have a few extra, don't panic. There are additional pins that may be designated as various types of GPIO, including FLAGS, IRQ, and device selects.

Build a "cheat sheet" that lists the DAI pin numbers (1-20), the signal to which you are connecting each pin, and whether it is an output, input, or bi-directional (from the perspective of the SHARC DSP). Even if the schematics are easy to read, a clean version of this table will be invaluable until your system is up and running.

Once the SRU is configured correctly (one or more routing patterns depending on your application), most of this will become transparent.

Programming the SRU

Think of each physical DAI as a 3-terminal peripheral with logical connections for an input, an output, and an enable that activates the pin buffer amplifier.

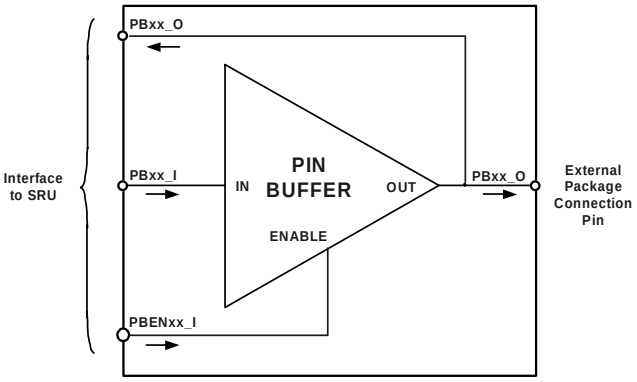

Figure 1. A Pin Buffer

A pin buffer is like a small buffer amplifier that can source enough current to drive the pin and a trace on the circuit board. When switched on (i.e., when its enable input is logic high), the logic value at the pin buffer input is driven onto the pin buffer output. When switched off (i.e., when its enable input is logic low), the buffer amplifier is high impedance, and the logic level of the pin buffer output is easily controlled by an external source. Pin buffers are the logical gateway for the physical IC package leads associated with the DAI.

Step 4: Program the Inputs to the SHARC DSP

Understanding the nomenclature is, arguably, the most difficult part of using the SRU. Programming is very simple. Ensure that you understand the next paragraph before continuing.

Since a pin buffer is an on-chip peripheral, the signal you connect to the physical package is

referred to as the pin buffer output. Although it is an input to the SHARC, it is an output from the pin buffer. Note that Figure 1 shows two connections to the trace, which are labeled PBxx_O (pin buffer output). One of them is part of the SRU interface, and the other is the external package connection pin. When the pin buffer is used as an input, the signal follows this path. Stated another way, a pin buffer output within the SRU is always equal to the logic value on the external pin.

All DAI pins that do not change signal flow direction can be routed in the SRU relatively simply. The signal follows the hard-wired path mentioned above. For each DAI pin that is an input to the SHARC DSP, tie the pin buffer enable low to ensure that the pin does not drive the line.

Next, connect the pin buffer output to the place in the SRU where you want to connect the signal. For example, the following macro instructions connect DAI pin 7 to the frame sync input of SPORT4:

```
SRU(LOW,PBEN07_I);
SRU(DAI_P07_O,SPORT4_FS_I);
```

Listing 1. Configuring a DAI Pin as an Input

Bit fields in SRU registers are always inputs and therefore can have one value only. The name of the node ends in "_I" to remind you that it is an input. Only one output may be connected to each input. The destination shown above may be any input shown in the bitfields of the SRU registers for Group A through Group D.

For example, an external frame sync is connected to SPORT3 as follows:

```
SRU(LOW,PBEN07_I);
SRU(DAI_P07_O,SPORT3_FS_I);
```

Listing 2. DAI Pin Input to SPORT3 Frame Sync

An output signal is just an encoding – an enumerated value entered into a bit field. Thus, an output may connect to any number of inputs within the same group.

The third SRU connection is not always necessary, but it is recommended that you tie the pin buffer input to low as follows:

```
SRU(LOW,DAI_P07_I);
```

Listing 3. Setting an Unused Pin Buffer Input Low

There are a couple of reasons for this. First, it can make your code clearer. If the enable and input of a pin buffer are tied low, that DAI pin is obviously being used as an input (or not at all). Also, preventing logic from unnecessary switching saves power and reduces RF radiation. For example, there is no reason to pass a high frequency clock through the SRU if it will be ignored. Last, all of the DAI pins are connected to the serial ports when the processor comes out of reset. The default value is not logic low. If there is a mistake in your code that enables the pin buffer, the output may be driven with a signal that is difficult to identify. Again, this is not strictly necessary, but it is recommended practice.

Step 5: Program the Outputs from the SHARC

For each pin that is an output from the SHARC DSP, tie the pin buffer enable high. This ensures that the pin buffer does drive a signal onto the external pin. More specifically, it ensures that the logic value at the pin buffer's input is driven onto the pin buffer's output.

Connect the (internal) source signal to the pin buffer input. For example, use the following macro instructions to connect the output of Timer 0 to DAI pin 14:

```
SRU(HIGH,PBEN14_I);
SRU(TIMER0_O,DAI_P14_I);
```

Listing 4. Configuring a DAI Pin as an Output

Note that the source signal (TIMER0_O in this example) must be an output listed in the Group D Sources – Pin Signal Assignments table in the

ADSP-2126x SHARC DSP Peripherals Manual [1], and will end in "_O" as a reminder that it is an output.

An input may be connected to only one output, but an output may be connected to multiple inputs.

An output signal may be connected to numerous inputs. For example, connect the clock output of SPORT2 to both DAI pin 7 and as a clock input for SPORT1 as follows:

```
/* Make DAI pin 14 an output */
SRU(HIGH,PBEN14_I);

/* Send the clock out DAI pin 14 */
SRU(SPORT2_CLK_O,DAI_P14_I);

/* Use the SPORT1 clock for SPORT2 */
SRU(SPORT2_CLK_O,SPORT1_CLK_I);
```

Listing 5. Routing an Output to More than One Input

In Listing 5, instead of connecting clock output to pin buffer input and then pin buffer output to the other SPORT clock input, notice that both of the inputs are connected directly to the clock output. This is because the input and the output of the pin buffer are not guaranteed to have the same value. As long as the pin buffer enable, PINEN14, is set to high (asserted), this daisy-chaining is equivalent to the parallel connection made above. However, if the pin enable is deasserted, the pin acts as an input and the clock input to SPORT1 is driven from off-chip.

Unless you specifically want the connection to change based on the value of the pin buffer enable, the best practice is to connect directly to the source as shown in Listing 5.

It is not necessary to route a pin buffer output when it is not needed.

Step 6: Program the Bi-directional DAI Pins

As mentioned in Step 2, a pin that serves as both an input and an output must have a signal present in the SRU that dictates the flow direction. Several peripherals, which are explicitly designed to be bidirectional, have input, output, and enable nodes associated with a single signal.

For example, each of the serial port signals (clock, frame sync, data channel A, and data channel B) can be an input or an output. The direction of these signals is controlled in the core by writing to the SPCTLx registers. You can connect the clock signal for SPORT0 to DAI pin 6 as follows:

```
SRU(DAI_P06_O,SPORT0_CLK_I);
SRU(SPORT0_CLK_O,DAI_P07_I);
SRU(SPORT0_CLK_PBEN_O,PBEN07_I);
```

Listing 6. A Bidirectional DAI Pin

Refer to the discussion of bidirectional pins in the *ADSP-2126x SHARC DSP Peripherals Manual* [1] for more information. Also, the table of valid sources for group F, located in the manual's appendix, describes the I/O register and includes all pin enable signals from all explicitly bidirectional peripherals.

Step 7: Be Creative and Use of the Options

Any Group F source can control pin direction, and you can perform tricks by taking advantage of the MISC signals. You are not limited to using these internally generated enable signals.

The code in Listing 7 may be a little hard to follow at first, but it demonstrates very powerful functionality. SPORT1 is always a clock master and SPORT0 is always a clock slave. Furthermore, the clock output from SPORT1 is always being driven as an output on DAI pin 4. The clock input to SPORT0 is always the same as the external signal on DAI pin 7. DAI pin 1 is always an input for control signal.

```
/* SPORT1 clock is an output */
SRU(SPORT1_CLK_O,DAI_P04_I);
SRU(HIGH,PBEN04_I);

/* Pin 7 direction is set by pin 1 */
SRU(MISCA0_O,PBEN07_I);
SRU(DAI_PIN01_O,MISCA0_I);
```

```
/* When pin 7 is an input, the off-
   chip signal drives SPORT0 clock */
SRU(DAI_P07_O,SPORT0_CLK_I);

/* When pin 7 is an output, SPORT1
   clock drives SPORT0 clock */
SRU(SPORT1_CLK_O,DAI_P07_I);

/* Pin 1 (dir control) is an input */
SRU(LOW,PBEN01_I);
SRU(LOW,DAI_P01_I);
```

Listing 7. DAI Pin Output

The tricky part is that DAI pin 7 (SPORT0 clock in) is a bidirectional pin, and its direction is controlled by DAI pin 1. When DAI pin 1 is low, DAI pin 7 is an input and SPORT0 receives its clock signal from off-chip. When DAI pin1 is high, DAI pin 7 is an output (equal to the SPORT1 clock output) and SPORT0 receives the internally generated clock from SPORT1.

Step 8: Optimize the Initialization Code

Each macro command used in this document expands to six SHARC DSP assembly language instructions. However, each macro call modifies a small bit field of a one SRU control register only. In SHARC assembly, a register can be loaded with a full 32-bit value in a single instruction.

In a typical system, the SRU is primarily configured at initialization, and modified infrequently. Write and debug your application using the macro as shown above. Create a subroutine that encapsulates a large number of these macro calls.

When you feel that the routing is correct and your I/O is working smoothly, start your application in the debugger and place a breakpoint just after you have completely configured the SRU. Open the debug windows that show the values of the various SRU registers and note their values. You can now comment out all of the macro calls (leave them in place as documentation), and replace them with a small number of instructions that write the value of the SRU control register completely.

For example, the following macro calls connect the frame syncs inputs for all six SPORTs to DAI pin 3:

```
SRU(DAI_P03_O,SPORT0_FS_I);
SRU(DAI_P03_O,SPORT1_FS_I);
SRU(DAI_P03_O,SPORT2_FS_I);
SRU(DAI_P03_O,SPORT3_FS_I);
SRU(DAI_P03_O,SPORT4_FS_I);
SRU(DAI_P03_O,SPORT5_FS_I);
```

Listing 8. Configuring Frame Syncs Using Macros

This will expand to 36 instructions, which will set the value of the control register SRU_FS0. The following in-line assembly instruction performs the same function:

```
asm("r0=0x00210842; dm(SRU_FS0)=r0;");
```

Listing 9. Configuring Frame Syncs Using Inline asm

This reduces the initialization code size from 36 instructions to 2 and removes the overhead penalty associated with the macro. You do not have to calculate this value (or waste time debugging it) because it displays directly in the debugger window. Obviously, if you are writing in assembly, you can ignore the wrapper. For a typical system, you can save the code space and execution time of a few hundred 48-bit instructions.

Summary

The aggregate bandwidth of the DAI pins is much higher than it may appear at first glance because the SRU eliminates unnecessary pins. An even less obvious benefit, however, is the myriad ways in which one signal can be used to manipulate another signal, such as gating, triggering, masking, and re-clocking.

This application note scratches the surface in describing the possibilities. DAI resources, such as the Precision Clock Generators, versatile timers, flags, interrupt sources, and pin buffer

inverters increase the number of "trick" permutations dramatically. Be creative, and you may be surprised at what the SRU has to offer.

References

[1] *ADSP-2126x SHARC DSP Peripherals Manual*. Revision 1.0, December 2003. Analog Devices, Inc.

Document History

Version	Description
Rev 1 – February 12, 2004 by K. Malsky	Initial Release