Controlling and Monitoring DSP Conductor Configurations

1. Introduction

DSP Conductor™ is a development environment that allows you to graphically define audio DSP algorithms, generate DSP code, and adjust the parameters of the running code.

Although DSP Conductor is intended for use by product engineers rather than end users, it is often desirable to allow end users to control a subset of DSP parameters. This control may be accomplished by a local microcontroller attached to the host port of the DSP or by a remote application communicating to the DSP via the IP-based simple network management protocol (SNMP). All parameters in a stand-alone DSP configuration may be adjusted by either of these methods. The DSP Conductor for CobraNet™ application itself uses SNMP to control and monitor configurations.

This application note describes how to identify and interface with the parameters you designate as end-user controllable.

2. Generating Deliverables

Once you have built, adjusted, and are satisfied with a DSP configuration you may use the items in the *Tools ⇨ Generate Deliverables...* menu to generate reference files for the configuration. Clicking *Tools ⇨ Generate Deliverables... ⇨ As Header…* will bring up a file browser allowing you to choose the location and name for a C header file, called the *configuration header file*, that describes the interface to your configuration. By default the file will have the same root name as your configuration with an extension of *.h*. *Tools ⇨ Generate Deliverables... ⇨ As XML…* will bring up a file browser allowing you to choose the location and name for an XML file, called the *configuration XML file*, that describes the interface to your configuration.

3. Interpreting the Configuration Header File

For each DSP parameter in the configuration, four header definitions are generated. All four are of the following format:

```
#define dspc_<primitive name>_<primitive ruid>_<parameter name>_<metric> <value>
```

`<primitive name>` - Name given to the primitive in the XML files that define the primitive. A single and unique name is given to each primitive in the device library. Refer to the application note AN277, *"Creating DSP Conductor Primitives"* for detailed descriptions of the XML files. The primitive name is typically a more technical description of the primitive than the name shown on the corresponding processing block in the GUI. Examples include *generator_sine_wave* and *mixer_NxM*.

<primitive ruid> - The primitive's relatively unique identifier (RUID) is a number that uniquely identifies an instance of a primitive within a configuration. The primitive RUID is required to differentiate multiple instances of the same primitive in a configuration.

<parameter name> - Name given to the parameter in the XML files that define the primitive. Primitives typically have numerous parameters, each uniquely named. Refer to the application note AN277, *"Creating DSP Conductor Primitives"* for detailed descriptions of the XML files. Example parameter names include *c1*, *hold_count*, and *send*. For multi-channel devices such as mixers, the parameter name may also be constructed to include a channel number in the form <parameter name>_<channel number>.

A different <metric> code is used for each of the four entries associated with a DSP parameter. The four metric codes, *wo*, *rw*, *fb* and *sg* are described below.

wo - Word Offset

Word offset of the parameter. The value is used to generate HMI addresses and SNMP OIDs. See Section 7. *Forming HMI Addresses* and Section 6. *Forming SNMP OIDs* sections below for a complete description of the use of word offsets.

rw - Read/Write

Access rights for the parameter: 0 - read-only, 1 - read/write. Access rights come into play when generating HMI addresses and SNMP OIDs as there are separate regions assigned to read-only and read/write parameters.

fb - Fraction Bits

Number of fractional bits in the parameter. The DSP targeted by DSP Conductor is a 32-bit, fixed-point processor. All DSP Conductor parameters are 32 bits. The value of this definition indicates the fixed position of the decimal point for the parameter: 0 - the parameter is an integer with full 32-bit range, 31 - the parameter is a fraction with its absolute value less than or equal to one.

sg - Sign

Indicates whether the parameter carries a signed or unsigned value: 0 - unsigned, 1 - signed.

4. Interpreting the Configuration XML File

The configuration XML file shown in Figure 1 contains a superset of the information contained in the configuration header file. The XML file could, as an exercise, be transformed into the header file using XSLT, Python, JavaScript or any programming language with an XML parser available. The XML file has additional information available that may make this format more useful. It is arranged by element, and it contains the title of each element as well as the full hierarchical path if the element is inside a Hierarchical Block. This allows coefficients to be addressed using the text on the block rather than the relatively obscure <primitive ruid> field used in the header file, assuming all elements have unique names.

The following description of the sample configuration file assumes the reader has a basic understanding of XML. In the following sentences *element* (with a lower case 'e') refers to an XML element, and *Element* (with an upper case 'E') refers to a DSP Conductor processing Element. Note that within the configuration XML file, 'device' is used as a synonym for 'Element' and 'coefficient' is used as a synonym for 'parameter'. Please refer to Figure 1.

```
 1. <?xml version="1.0" standalone="yes" ?>
 2. <dsp_conductor_control_map>
 3.   <info>
 4.     <signature value="3239014490" />
 5.     <project_name value="untitled1" />
 6.     <date_time value=" 11/02/2005  04:33 PM" />
 7.   </info>
 8.   <devices>
 9.     <device title="Sine" type="generator_sine_wave" path="Sine" id="1" >
10.       <coefficient name="gain" offset="2" mode="RW" signed="1" fract_bits="31" />
11.       <coefficient name="omega" offset="1" mode="RW" signed="1" fract_bits="31" />
12.       <coefficient name="ramp" offset="0" mode="RW" signed="1" fract_bits="31" />
13.     </device>
14.     <device title="Presence" type="detector_signal_presence" path="Presence" id="2">
15.       <coefficient name="hold_time" offset="5" mode="RW" signed="1" fract_bits="0" />
16.       <coefficient name="infinite_hold" offset="4" mode="RW" signed="1" fract_bits="0" />
17.       <coefficient name="presence" offset="1" mode="RO" signed="1" fract_bits="0" />
18.       <coefficient name="threshold" offset="3" mode="RW" signed="1" fract_bits="31" />
19.     </device>
20.   </devices>
21. </dsp_conductor_control_map>
```

Figure 1. Example Configuration XML File Contents

Line 1 is standard XML boilerplate.

Line 2 is the *root* element of the document, called *dsp_conductor_control_map*. This element ends on line 21 in the example.

Line 3 is the beginning of the *info* element that provides general information about the configuration. This element ends on line 7 in the example. There will be exactly one info element in the file.

Line 4 is the *signature* element. This element contains a single attribute called *value* that contains a 32-bit unsigned long integer that is the signature of the configuration. The signature uniquely identifies the layout and the data types of the coefficients in the configuration. The signature value may be read from the first location in the read-only parameter buffer. When you establish communications with the CS4961xx you should read the signature value and confirm that it matches the signature from the configuration XML before reading or writing any coefficients. If the signatures do not match, any values that you read will be meaningless and any values that you write will have unpredictable results.

Line 5 is the *project_name* element that contains a single attribute called *value* that contains the name of the configuration.

Line 6 is the *date_time* element that contains a single attribute called *value* that contains a string describing the date and time that the configuration was compiled. (*Note:* The *value* attribute was inadvertently omitted from configurations generated by versions 1.1.0 and 1.1.1 of DSP Conductor).

Line 8 is the beginning of the *devices* element. This element contains a series of device elements. There will be exactly one devices element in the file.

Line 9 is the beginning of the first of many *device* elements. There will be one device element for each DSP Conductor processing Element in the configuration. The device element contains four attributes.

The *title* attribute contains the text on the face of the Element.

The *type* attribute contains the internal type name of the device, which is more technical than the title. This is the same as the <primitive name> described in Section 3.

The *path* attribute normally looks exactly like the title but, in the case where the Element resides inside a hierarchical block, *path* will contain the title of the block followed by a forward slash and then the title of the Element. Since Elements can be nested inside Hierarchical blocks inside other Hierarchical blocks to an arbitrary depth, the path is really the concatenation of the titles of all of the blocks, from the outermost block to the innermost block followed by the tile of the Element itself, all separated by forward slashes.

The *id* attribute contains a unique integer identifying the instance of the Element in the configuration, this is the same as the <primitive ruid> described in Section 3. (*Note:* The id attribute is not available in release 1.1.0 or 1.1.1).

Finally, the *device* element contains a series of coefficient elements.

Line 10 is the first of many *coefficient* elements. There is one coefficient element for each parameter that can be read or written to control the DSP configuration. Each coefficient element contains five attributes.

The *name* attribute contains the name of the parameter. This is the same as <parameter name> as described in Section 3.

The *offset* attribute is the word offset of the parameter.

The *mode* attribute is either 'RW' signifying read-write or 'RO' signifying read-only.

The *signed* attribute is either *1*, indicating the parameter is signed or *0*, indicating the parameter is unsigned.

The *fract_bits* attribute indicates the number of fractional bits in the parameter as described in Section 3. Note that the minimum value of offset among all 'RW' parameters is 0, but the minimum value of offset among all "RO" parameters is 1. This is because offset 0 in the 'RO' space contains the signature as described above.

5. Parameter Values

All DSP Conductor parameters are 32-bit values. While the sign (*sg*) and fraction bits (*fb*) attributes give some indication of how the DSP might interpret the values, in the end, it is the specifics of the DSP implementation that determine exactly how values are interpreted. The way values are interpreted can be discerned by examining the *Crunch Functions* section in the element's implementation XML file. For details on the implementation XML file refer to Cirrus application note AN277, *"Creating DSP Conductor Primitives"*.

6. Forming SNMP OIDs

All SNMP variables are referenced by the object identifier (OID). An OID is a dot-separated string of integers enumerating the path from the root of the SNMP management information base (MIB) to the variable. Firmware supporting DSP Conductor features a *dspExtensions* SNMP extension agent rooted at *iso(1).org(3).dod(6).internet(1).private(4).enterprises(1).peakAudio(2680).cobraNet(1).dspExtensions(4)*.

DSP Conductor parameters under this extension are separated into two tables of 32-bit integers. The first table, rooted at *dspExtensions(4).control(2).controlRWTable(2)*, holds read-only parameters. The second table, rooted at *dspExtensions(4).control(2).controlROTable(4)* holds read/write parameters. Note that access to SNMP tables is achieved using 1-based indices. The word offsets specified in the configuration header file are 0 based.

To construct an OID for a read/write variable, append the parameter's word offset (*wo*) plus one to the OID for the *controlRWTable* values:

iso(1).org(3).dod(6).internet(1).private(4).enterprises(1).peakAudio(2680).cobraNet(1).dspExtensions(4).control(2).controlRWTable(2).controlRWEntry(1).controlRWValue(2).

For example, the OID for a read/write parameter with word offset *5* is 1.3.6.1.4.1.2680.1.4.2.2.1.2.6

To construct an OID for a read-only variable, append the parameter's word offset (*wo*) plus one to the OID for the *controlROTable* values:

iso(1).org(3).dod(6).internet(1).private(4).enterprises(1).peakAudio(2680).cobraNet(1).dspExtensions(4).control(2).controlROTable(4).controlROEntry(1).controlROValue(2).

For example, the OID for a read-only parameter with word offset *13* is 1.3.6.1.4.1.2680.1.4.2.4.1.2.14

6.1 1.0.0 Beta OIDs

The *1.0.0* beta version of DSP Conductor uses an early revision of the *dspExtensions* MIB with slightly different OIDs as follows:

Read/write base OID:

iso(1).org(3).dod(6).internet(1).private(4).enterprises(1).peakAudio(2680).cobraNet(1).dspExtensions(4).controlRWTable(3).controlRWEntry(1).controlRWValue(2)

Read-only base OID:

iso(1).org(3).dod(6).internet(1).private(4).enterprises(1).peakAudio(2680).cobraNet(1).dspExtensions(4).controlROTable(5).controlROEntry(1).controlROValue(2)

7. Forming HMI Addresses

Parameters accessed through HMI are located in two separate address blocks. Access to the parameters is controlled by access control rights.

Read/write parameters are based at HMI address 0x76000. To generate an HMI address for a specific read/write parameter, add 0x76000 to the word offset (wo) definition for the parameter.

Read-only parameters are based at HMI address 0x7A000. To generate an HMI address for a specific read-only parameter, add 0x7A000 to the word offset (wo) definition for the parameter.

Contacting Cirrus Logic Support

For all product questions and inquiries contact a Cirrus Logic Sales Representative.
To find the one nearest to you go to **www.cirrus.com**