DICE Firmware
Command Line Interface
(CLI) Reference

Firmware Version 3.0.x

Build 82 or Later

CONTENTS

DICE Command Reference

Introduction

The DICE Firmware Command Line Interface (CLI) allows developers to manually configure functions in the DICE. The following table gives a brief description of each CLI command.

Syntax

CLI commands, variables and parameters are case-sensitive. For example, "**echo On**" will result in a parameter error, and "**echo on**" should be used.

Commands that have subfunctions are separated with either a period '**.**' or a colon '**:**'

For example, the following commands are equivalent:
> lhl.br
> lhl:br

Built-in commands

Note that "built-in" does not need to be typed in the command-line when using commands in this category. However, all other categories require that the category name be typed as part of the command line for example
> **dal.start 0**

Optional commands

Not all commands listed here are available in all DICE Applications. The available commands depend on which build options are set in the firmware when the application is made.

The table below indicates which target and additional flags are necessary to enable the command if it is not enabled by default.

In general each module has its own CLI tool #define. For example, to enable to CLI tools for the **pb** module, define **_CLI_TOOL_PB** in **cliToolDefs.h**.

Also see **Make.params**, and #defines in the source code, to determine which modules are used. For example, if the application is built with the **_DICE_DRIVER** driver model, then the "**avc**" and related CLI commands such as "**panel**" are not available.

The CLI can be removed altogether to save memory requirements in production applications by removing the **_CLI** flag in **Make.params**.

Chip-specific commands

Commands may differ depending on the chip used in development. Specifically, the modules in the **\firmware\chip\dice**X directory will have their own CLI commands or variations. Your available commands will therefore differ depending on your build configuration. Chip-specific commands or parameters are noted in the tables.

Command-line editing

The CLI saves your last 9 commands in its history buffer. You can recall previous commands using the up/down arrow keys, and you can edit the command line as well. Note that you must use a serial terminal program that supports sending terminal keys to recall commands. HyperTerminal does not support this, although HyperTerminal PE, TeraTerm Pro and many other programs do.

Command Reference

built-in commands

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage built-in
built-in [All targets]	env	Displays all environmental variables and their values Use: **env [<variablename>]**
	get	Gets the value of the specified register Use: Simple Method: **get <variablename|address[:<sizeinbytes>]> [hex|bin|dec|bool]** get FPGARegister1 hex get FPGARegister1 binary Direct Bit Access: **get <variablename|address[:<sizeinbytes>]> bit <integer>** get FPGARegister1 bit 2
	set	Set the specified register to the specified value Use: Simple Method: **set <variablename|address[:<sizeinbytes>]> <integer>** set FPGARegister1 0x81 (assumed as 32 bits) set 0xFF00FF00 0x81 (assumed as 32 bits) set 0xFF00FF00:3 0x81 (24 bits) OR'd Method: **set <variablename|address[:<sizeinbytes>]> <or> <integer>** This method does a common OR operation. Using this command is the same as doing in C: 'var1 |= var2' set FPGARegister1 or 0x80 set 0xFF00FF00:4 or 0x0000FF8F Direct Bit Access: **set <variablename|address[:<sizeinbytes>]> bit <integer> <on|off>** set FPGARegister1 bit 2 on
	setm	Set multiple address-value pairs in one CLI operation Use: **setm <addr1> <data1> <addr2> <data2> ... <addrN> <dataN>**

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage
		built-in
	gms	Get-Modify-Set tool to modify a value. Modifies value specified by variablename|address Use: **gms <variablename|address[:<sizeinbytes>]> <clearmask> <setmask>** <clearmask>: each bit set in clearmask clears bit in value <setmask>: each bit set in setmask sets bit in value identical to: set <variablename|address[:<sizeinbytes>]> and <and-mask> or <or-mask> where <and-mask>: bitwise-inverse of clearmask and <or-mask>: setmask
	dump	Displays raw memory in formatted fashion Use: **dump <address> <unitsize 1,2,4> <unitcount> <items per line>**
	splash	Display startup screen Use: **splash**
	exit	Exits the application Use: **exit**
	reset	Resets the device Use: **reset**
	echo	Turns on/off the echoing of characters. Use: **echo <on|off>**
	prompt	Turns on/off the printing of the prompt character Use: **prompt <on|off>**
	memtest [_CLI_TOOL_MEMTEST] Only for internal_ram test builds.	Test memory area using specific bit width access. Use: **memtest <type> <baseaddr> <rangesize> <bitwidth> <step> <verbose>** <type>: data,addr,rdwr,all <bitwidth>: 8b,16b,32b,all <step>: all: default, testing all addresses (step=4) o.w.: value in steps of 4 (step=4,8,16,...) <verbose>: on|off - output test messages

aml – abstract MIDI layer

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage aml
aml [all targets]	list [_AML]	List all current aml devices Use: **aml.list**
	dest	Set the destination of a RX device Use: **aml.dest <src_dev_id> <dest_dev_id>** <src_dev_id>: as created by app (see aml.list) <dest_dev_id>: as created by app (see aml.list)
	get	Get a character from a monitor enabled device Use: **aml.get <dev_id>** <dev_id>: device id (see aml.list)
	put	Put a character to a monitor enabled device Use: **aml.put <dev_id> <char>** <dev_id>: device id (see aml.list) <char>: character to send, in hex format

avc – (AV/C) audio video control

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage avc
avc [_AVC target]	settarget	set target mode (node and unit) Use: **avc.settarget <targetNode> <targetUnit>** <targetNode>: 'local', 'other' or use showtargets for supported devices <targetUnit>: unit:u, audio subunit:a, panel subunit:p
	gettarget	get target mode Use: **avc.gettarget**
	showtargets	show target devices Use: **avc.showtargets**
	power	sends avc power command Use: **avc.power <ctype> <power_state>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <power_state>: off:0x60 on:0x70 status:0x7f
	unitinfo	sends avc unit info status command Use: **avc.unitinfo <ctype>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage
		avc
	subunitinfo	sends avc subunit info status command Use: **avc.subunitinfo <ctype> <page>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g\n\r"\ <page>: 0,..,7
	reserve	sends avc reserve command Use: **avc.reserve <ctype> <priority> <text>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <priority>: 0x00,..,0x0f <text>: 12 bytes of ASCII or ""
	version	sends avc version status command Use: **avc.version <ctype> <version_information>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <version_information>: 0xff (latest version) 0x00,..,0xfe (specific version)
	vendor	sends avc vendor status command Use: **avc.vendor <ctype>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g
	pluginfo	sends avc plug info status command Use: **avc.pluginfo <ctype> <subfunction>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <subfunction>: 0x00: Isoch+Ext (subunit), 0x01: Async, 0x40-0x7F: General Bus
	channelusage	sends avc channel usage status/notify command Use: **avc.channelusage <ctype> <channel>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <channel>: 0,..,63
	connect	sends avc connect command Use: **avc.connect <ctype> <index>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <index>: in tables of connection combinations
	connectAV	sends avc connectAV command Use: **avc.connectAV <ctype> <index>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <index>: in tables of connection combinations
	connections	sends avc connections status command Use: **avc.connections <ctype>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g
	digitalinput	sends avc digital input command Use: **avc.digitalinput <ctype> <connection_state>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <connection_state>: break:0x60 establish:0x70 status:0xff

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage
		avc
	digitaloutput	sends avc digital output command Use: **avc.digitaloutput <ctype> <connection_state>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <connection_state>: break:0x60 establish:0x70 status:0xff
	disconnect	sends avc disconnect command Use: **avc.disconnect <ctype> <index>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <index>: in tables of connection combinations
	disconnectAV	sends avc disconnectAV command Use: **avc.disconnectAV <ctype> <index>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <index>: in tables of connection combinations
	ipsf	sends avc input plug signal format command Use: **avc.ipsf <ctype> <plugID> <format>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <plugID>: 0,.,,30 <format>: plug signal format [0|0|fmt:6|fdf:24]
	opsf	sends avc output plug signal format command Use: **avc.opsf <ctype> <plugID> <format>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <plugID>: 0,.,,30 <format>: plug signal format [0|0|fmt:6|fdf:24]
	generalbus	sends avc general bus setup command Use: **avc.generalbus <ctype> <index>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <index>: in tables of connection combinations
	signalsource	sends avc signalsource command Use: **avc.signalsource <ctype> <index>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <index>: in tables of connection combinations
	inputselect	sends avc inputselect command Use: **avc.inputselect <ctype> <nodeAddr> <subfuntion>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <subfuntion>: 0:connect, 1:path chg, 2:select, 3:disconnect
	outputpreset	sends avc outputpreset command Use: **avc.outputpreset <ctype> <index>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <index>: in tables of connection combinations
	CCMprofile	sends avc CCMprofile command Use: **avc.CCMprofile <ctype> <index>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <index>: in tables of connection combinations

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage avc
	security	sends avc security command Use: **avc.security <ctype>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g
	checkall	performs avc check all Use: **avc.checkall <ctype>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g
	list	display list Use: **avc:list <list>** <list>: r: reserve reservation n: reserve notify p: power notify u: channel usage notify c: connect notify a: connectAV notify i: input plug signal format notify o: output plug signal format notify s: signal source notify
	getprintmode	get avc unit print mode Use: **avc:getprintmode**
	setprintmode	set avc unit print mode Use: **avc:setprintmode <mode>**
	showprintmodes	show avc unit print modes Use: **avc:showprintmodes**

avs – Audio Video System

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage avs
avs [All targets]	event	Displays the AVS interrupt events or clears event counters Use: **avs.event <operation>** <operation>: dump, dumpifset, clear Dumpifset: dump non-zero counter events
	dump	Dump AVS Tx or AVS Rx registers Use: **avs.dump <PlugID> <Tx|Rx> <all>** <plugID>: rx: 1-4, tx: 1-2 <all>: 0,1 dump all additional general avs registers

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage
		avs
	dumpFmt	Dump AVS format registers Use: **avs.dumpFmt <plugID> <rx|tx> <all>** <plugID>: rx: 1-4, tx: 1-2 <all>: 0,1 dump all fmt registers (use:1)
	reset	Causes a reset of all AVS registers Use: **avs.reset**
	it	Starts or stops the specified AVS Tx|Rx. Causes a reset of all AVS registers. Use: **avs.it <plugID> <rx|tx> <start|stop>** <plugID>: rx: 1-4, tx: 1-2
	itc	Set isoch tx|rx channels Use: **avs.itc <plugID> <rx|tx> <ch>** <plugID>: rx: 1-4, tx: 1-2 <ch>: isoch channel 0-255
	its	Set iso tx|rx channel sample rates Use: **avs.its <plugID> <rx|tx> <samplerate>** <plugID>: rx: 1-4, tx: 1-2 <samplerate>: 32k 44k 48k 88k 96k 176k 192k
	dbs	Set number of audio channels to be streamed for the specified tx|rx Use: **avs.dbs <plugID> <tx|rx> <channels>** <plugID>: rx: 1-4, tx: 1-2 <channels>: max 16 for tx
	dbsx	Set number of audio channels and MIDI to be streamed for the specified tx|rx Use: **avs.dbsx <plugID> <tx|rx> <channels> <bMIDI>** <plugID>: rx: 1-4, tx: 1-2 <channels>: max 16 for tx <bMIDI>: on|off
	qsel	Set QSel slot index values for rx plug Use: **qsel <plugID> <"qselstr">** <plugID>: rx: 1-4 <"qselstr">: "xx xx xx xx xx .. xx" <xx>: byte hexvalue of index value - 17 items
	mute	Set mute on or off for tx|rx Use: **avs.mute <plugID> <rx|tx> <on|off>** <plugID>: rx: 1-4, tx: 1-2
	speed	set isoch tx speed Use: **avs.speed <plugID> <speed>** <plugID>: tx: 1-2 <speed>: s100, s200, s400
	getspeed	get iso tx speed Use: **getspeed <plugID>** <plugID>: tx: 1-2

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage
		avs
	format	Set AVS format configuration for transmitter or receiver Use: **avs.format <plugID> <tx|rx> <format> <channel>** <plugID>: rx: 1-4, tx: 1-2 <format>: trans, aesMst, aesBSClk, aesSlv, label <channel>: 0-7 (block sync input(tx) or output(rx) channel)
	itp	Set AVS ITP on or off Use: **avs.itp <on|off>**
	midi	Set isoch tx|rx MIDI Use: **avs.midi <tx:on|off> <tx:plugID> <rx:on|off> <rx:plugID> <count>** <plugID>: rx: 1-4, tx: 1-2 <count>: in 10ms intervals midi tx use: avs.midi on <plugid> off 0 <count> midi rx use: avs.midi off 0 on <plugid> <count> midi rx+tx use: avs.midi on <plugid> on <plugid> <count>
	txp	Set iso tx presentation offset Use: **txp <plugID> <samples> <phase>** <plugID>: 1-2 <samples>: 0-100 <phase>: 0-100
	rxa	Set iso rx presentation adjust Use: **rxa <plugID> <samples> <phase>** <plugID>: 1-2 <samples>: -100-100 <phase>: 0-100
	fsm [DICEII only]	Set various FSM parameters, counts are in times of 10ms Use: **fsm <toggle:on|off> <errClr:cnt> <noErr:cnt> <disable:cnt>** <cnt>: 1-255
	specify	Set receiver specification parameters Use: **specify <rxID> <DBS> <FMT>** <rxID>: 1-4 <DBS>: yes/no <FMT>: yes/no

avstst – Audio Video System event capture

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage avstst
avstst [All targets] [DICEII only]	capture	Capture <seconds> of svs events, max 8000 Use: **avstst.capture <seconds>**
	dump	Dump captured events Use: **avstst.dump**

cmp – Connection Management Protocol

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage cmp
cmp [_AVC target]	info	Display iMPR,oMPR and all iPCR,oPCR plug register info for node Use: **cmp.info <nodeAddr>**
	bcinestablish	Establish a broadcast in connection Use: **cmp.bcinestablish <iPCRNumber>**
	bcinbreak	Break a broadcast in connection Use: **cmp.bcinbreak <iPCRNumber>**
	bcoutestablish	Establish a broadcast out connection Use: **cmp.bcoutestablish <iPCRNumber>**
	bcoutbreak	Break a broadcast out connection Use: **cmp.bcoutbreak <iPCRNumber>**
	p2pinestablish	Establish a p-2-p in connection Use: **cmp.p2pinestablish <iPCRNumber> <oPCRNumber> <oNodeAddr>**
	p2pinbreak	Break a p-2-p in connection Use: **cmp.p2pinbreak <iPCRNumber> <oPCRNumber> <oNodeAddr>**

cr – Registered Descriptor Callbacks

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage cr
cr [_AVC target]	list	This function shows all registered callbacks Use: **cr.list**
	show	shows key value pairs for a callback descriptor Use: **cr:show <address>**

csr – Control and Status Registers

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage csr
csr [All targets]	info	display configuration ROM CSR Use: **csr.info**
	update	display configuration ROM CSR Use: **csr.update <index> <data>**

dal – DICE Abstraction Layer

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage dal
dal [All targets] See below for detailed usage.	create	Create a clock domain Use: **dal.create <0|1> <rateMode> <inputs> <outputs>** <rateMode>: low, low_mid, high, slave_l, slave_m, slave_h low: accept nominal rates from 32k to 48k low_mid: accept nominal rates from 32k to 96k high: accept nominal rates from 176k4 to 192k slave_l: slave to interface 0 (only) in low mode (DICE II only) slave_m: slave to interface 0 (only) in mid mode (DICE II only) slave_h: slave to interface 0 (only) in high mode (DICE II only) <inputs>: aes, adat … <outputs>: aes, adapt …
	destroy	Destroy a clock domain Use: **dal.destroy <0|1>**
	start	Start a clock domain Use: **dal.start <0|1>**
	clock	Start a clock domain Use: **dal.clock <0|1> <source> <rate>** <source>: aesAny, aes0, aes1, aes2, aes3, adat, tdif, wc, avs1 avs2, avs3, avs4, ext, dsai0, dsai1, dsai2, dsai3, int <rate>: 32k, 44k1, 48k, 88k2, 96k, 176k4, 192k, any
	route	Add a route in a domain Use: dal:route <0|1> <dstDev> <dstCh> <srcDev> <srcCh> <type> **DICE Jr / Mini** <dstDev>: aes, adat, mixer0, mixer1, ins0, ins1, apb, avs1, avs2 **DICE II** < dstDev>: aes, adat, tdif, dsai0, dsai1, dsai2, dsai3,i2s0, i2s1, i2s2, apb, avs1, avs2 <dstCh> : 0..15 **DICE Jr / Mini** <srcDev>: aes, adat, mixer, ins0, ins1, apb, avs1, avs2, mute **DICE II** < srcDev>: aes, adat, tdif, dsai0, dsai1, dsai2, dsai3,i2s0, i2s1, i2s2, apb, avs1, avs2, avs3, avs4, mute <dstCh> : 0..15 <type> : t1, t2, t4, t8
	installCB	Install verbose callback for DAL Use: **dal.installCB**
	removeCB	Remove verbose callback for DAL Use: **dal.removeC**B
	dump	Dump configuration and status for a clock domain Use: **dal.dump <0|1>**

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage
		dal
	dumpsync	Dump sync source status Use: **dal.dumpsync <0|1>**
	config	Set JET PLL Configuration Use: **dal.config <msWaitStable> <msWaitLock> <PLLoffWhenNoSource>** <msWaitStable>: wait for the rate to stabelize, default 100 ms <msWaitLock>: timeout on JET PLL lock, default 800 ms <PLLoffWhenNoSource>: Turn clocks off when unlocked, on/off
	avsphase	Get the phase offset of AVS receiver in relation to main clock domain Use: **dal.avsphase <avs1|avs2>**

dice – DICE Clocking and Audio Interfaces

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage (DICE II) dice
dice [All targets]	sync	Select synchronizer input clock source Use: **dice.sync <0|1> <source>** <source>: 44k, 48k, ext, hpll, aes, adat, tdif
	hpll	Select hybrid pll input clock source Use: **dice.hpll <0|1> <source>** <source>: aes, adat, tdif, wc, avs1, pre, ext, dsaitx0, dsairx0
	dump	Dump register contents for module Use: **dice.dump <module>** <module>: clock, aes, router, dsai, adat, tdif, i2s, hpll1, hpll2, aesRx, aesTx
	dsai	Set DSAI communication mode Use: **dice.dsai <1bit|32bit|master|slave>** 1-bit or 32-bit frame sync, master or slave clock mode
	aes	Set AES communication mode Use: **dice.aes <master|slave>**
	apb [deprecated]	Set volume for audio routing through ARM APB interface Use: **dice.apb <0-256>**
	blockSync	Select block sync source to destination Use: **dice.blockSync <dst> <src>** <dst>: ext, aes, avs <src>: aesRx, avsRx, aesTx, avsTx, extIn
	rsmf	I2S Rx Set Master Frequency Use: **dice.rsmf <i2sId> <on|off>**
	rs192	I2S Rx Set 192KHz mode Use: **dice.rs192 <i2sId> <on|off>**
	rsc	I2S Rx enable Master Clock output and set divider Use: **dice.rsc <i2sId> <on|off> <mckOutDiv>**
	rsci	I2S Rx Invert Clock signal Use: **dice.rsci <i2sId> <on|off> <bickInv>**
	rsdf	I2S Rx Set Data Format, L/R justify, data size & data delay Use: **dice.rsdf <i2sId> <i2sRxCh> <msbfirst[on|off]> <lJustify[on|off]> <dataSize[24b|32b]> <dataDelay[on|off]>**
	tsmf	I2S Tx Set Master Frequency Use: **dice.tsmf <i2sId> <on|off>**

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage (DICE II) dice
	ts192	I2S Tx Set 192KHz mode Use: **dice.ts192 <i2sId> <on|off>**
	tsc	I2S Tx Set master Clock output and set divider Use: **dice.tsc <i2sId> <on|off> <mckOutDiv>**
	tsci	I2S Tx Invert Clock signal Use: **dice.tsci <i2sId> <on|off>**
	tsdf	I2S Tx Set Data Format, L/R justify, data size & data delay Use: **dice.tsdf <i2sId> <i2sRxCh> <msbfirst[on|off]> <lJustify[on|off]> <dataSize[24b|32b]> <dataDelay[on|off]>**
	i2s_default	Set I2S interface to default settings Use: **dice.i2s_default**

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage (DICE Jr/Mini) dice
dice [All targets]	clksrc	Select Hybrid PLL input clock source Use: **dice.clksrc <source>** <source>: aes, adat0, adat1, wc, avs1, avs2, pre, ext
	dump	Dump register contents for module Use: **dice.dump <module>** <module>: clock, aes, router, hpll, hpll1, aesRx, aesTx
	aes	Set AES communication mode Use: **dice.aes <master|slave>**
	blockSync	Select source for block sync to destination Use: **dice.blockSync <dst> <src>** <dst>: blko, aestx, txdi1, txdi2 <src>: aesRx, avsRx1, aesTx1, avsTx1, avsTx2
	sync	Set synchronizer source Use: **dice.sync <value>** <value>: aes, slave, HPLL
	mode	Set FS source for router Use: **dice.mode <value>** <value>: 1x, 2x, 4x
	eclk	Set external clock Use: **dice.eclk <ClkId> <value>** <clkID>: wclk0, wclk1, extbr <value>: off, 1fb, 2fb, 4fb, sys1fs, extfbr
	clkReset	Reset dice clock Use: **dice.clkReset**

diceaes – configure AES audio interface

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage (DICE Jr/Mini) diceaes
diceaes [dicedriver]	enable	Enable/disable aes interface Use: **diceaes.enable <on|off>**
	mclk	Set clock source for aes interface Use: **diceaes.mclk <src>** <src>: aes0, aes1, aes2, aes3, aesany
	dw	Set dual wire mode Use: **diceaes.dw <off|on>**

diceadat – configure AES audio interface

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage (DICE Jr/Mini) diceadat
diceadat [dicedriver]	enable	Enable/disable adat interface Use: **diceadat.enable <on|off>**

dicedriver – interface with Host drivers

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage dicedriver
dice [dicedriver]	dump	Dump dicedriver state Use: **dicedriver.dump <mode>** <mode>: currently not used, defaults to all

drd – Device Reference Design

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage drd
drd [_AVC target and _DRD]	power	set drd powerstate Use: **drd:power <type>** <type>: on:0, standby:1, state:2

ds – Data Stream Services

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage ds
ds [_AVC target]	show	show modes info for datastream and dsbuffer Use: **ds.show <mode>**
	test	test datastream and dsbuffer Use: **ds.test <mode>**

dsp – DSP Interfacing

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage (DICEII) dsp
dsp [All targets]	setup	Set up dsp test Use: **dsp.setup <bLoad:on|off>** <bLoad>: load dsp code (off if used with dsp debugger)
	getvalue	Get dsp memory value Use: **dsp.getvalue <mem> <addr>** <mem>: x, y, p
	setvalue	Set dsp memory value Use: **dsp.setvalue <mem> <addr> <value>** <mem>: x, y, p
	getgain	Get dsp gain value Use: **dsp.getgain**
	setgain	Set dsp gain value Use: **dsp.setgain <gain>**
	testgain	Test dsp gain by lowering/increasing gain Use: **dsp.testgain**
	getaddr	Get default dsp memory and address setting used for getgain/setgain Use: **dsp.getaddr**
	setaddr	Set default dsp memory and address setting used for getgain/setgain Use: **dsp.setaddr <mem> <addr>** <mem>: x, y, p
	gethi08	Get dsp hi08 register value Use: **dsp.gethi08 <register>** <register>: 0,..,7
	sethi08	Set dsp hi08 register value Use: **dsp.sethi08 <register> <valule>** <register>: 0,..,7
	dumphi08	Dump dsp hi08 registers Use: **dsp.dumphi08**

eds – Embedded Descriptor Services

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage
		eds
eds [_AVC target]	tree	Displays useful information about EDS. Starts from the specified handle, and displays a graphical tree in hierarchical. Use: **eds.tree <root descriptor handle>**
	catalogue	displays all the descriptors instantiated on the device Use: **eds.catalog**
	connections	displays all the open descriptor connections Use: **eds:connections**
	debug	**- 15 -** When set to true the parsed data from incoming descriptor requests is shown in the monitor. Use: **eds.Debug**
	spec	Selects descriptor specifier used for next AVC command (open,read,write) Use: **eds.spec <specifier> <arg1> <arg2> <arg3>** <specifier>: subunit listId <list_id> listType <list_type> entryPos <list_id> <entry_position> entryObj <list_id> <list_type> <object_id> entryType <entry_type> objectId <object_id> infoType <info_block_type> <instance_count> infoPos <info_bloc_position>
	open	sends AVC open descriptor Use: **eds:open <ctype> <action>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <action>: close, read, write
	read	sends AVC read descriptor Use: **eds:read <ctype> <address> <length>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <address>: Address offset starting point to read from descriptor <length>: Number of bytes to be read from the target descriptor
	write	sends AVC write descriptor Use: **eds:write <ctype> <address> <length>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <address>: Address offset starting point to be written <length>: Number of bytes in the data to be written

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage — eds
	sample	Creates a sample descriptor structure Use: **eds:sample <operation>** <operation>: c: creates sample descriptors d: deletes sample descriptors l: move list1 to be a child of list2
	readIB	sends AVC read info block Use: **eds:readIB <ctype> <address> <length>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <address>: Address offset starting point to read from info block <length>: Number of bytes to be read from the target info block

envcfg – Environment Variable Configuration

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage — envcfg
envcfg [All targets]	Init [_FIS]	Initialize environment variable Use: **envcfg.init <envname>**
	getstr	Get the value of an environment variable as string Use: **envcfg.getstr <envname>**
	getlong	Get the value of an environment variable as long int Use: **envcfg.getlong <envname>**
	getall	Get the value of all environment variables Use: **envcfg.getall**

evm – SPI, Modes, and MIDI loopback

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage (DICE Jr/Mini) evm
sys [All targets]	spi	Send SPI command to Codec or CPLD Use: **evm.spi <dst> <word>** <dst>: codec, cpld
	modes	List available modes Use: **evm.modes**
	mode	Set the current config mode Use: **evm.mode <mode>** <mode>: id of the mode to set
	midibp	Enable/Disable 1394 MIDI loopback. This is for testing purposes to avoid having to put the loop cable on the board. Use: **evm.midibp <on|off>**

fis – Flash File System

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage fis
fis [All targets]	list	Display contents of flash file system Use: **fis.list**
	create	Create a flash file from RAM contents Use: **fis.create <src_start_address> <src_length load_addr> <entry_addr> <image_name>**
	delete	Delete a flash file Use: **fis.delete <image_name>**
	init	Initialize flash memory Use: **fis.init**
	write	Write RAM contents directly to flash memory Use: **fis.write <src_start_address> <src_length> <flash_address>**
	erase	Erase flash content Use: **fis.erase <flash_start_address> <erase_length>**

gray – Gray Coder/Decoder Interface

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage
		gray
gray [All targets]	setup	Set mode polling/interrupt for gray encoder handling Use: **gray.setup <polling>** <polling>: on|off
	setmode	Set gray encoders mode Use: **gray.setmode <enc> <mode>** <enc>: 0,..,3, all <mode>: default, direct, deferred
	getmode	Get gray encoders mode Use: **gray.getmode <enc>** <enc>: 0,..,3, all
	setaccmode	Set gray encoders acc mode Use: **gray.setaccmode <enc> <accMode>** <enc>: 0,..,3,all <accMode>: default,none,type1,type2,custom custom: for adding own acc handling callback here using a fixed test interrupt version
	getaccmode	Get gray encoders acc mode Use: **gray.getaccmode <enc>** <enc> 0,..,3, all
	callback	Install/remove gray encoder callback to dsp gain Use: **gray.callback <enc> <mode>** <enc> 0,..,3 <mode>: remove - remove installed callback dspgain - install dspgain callback print - install print value callback
	intlog	Gray interrupt log Use: **gray.intlog <mode>** <mode>: dump,dumpifset,dumpifenable,clear,regs

hpll – Hybrid PLL configuration

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage (DICE Jr/Mini) hpll
dicedriver [All targets]	divider	Set HPLL Divider Use: **hpll.divider <preDiv> <addDiv >** <preDiv>: pre-divider used to bring high rates down to base rate before PLL <addDiv>: feedback divider to multiply ref_event
	phase	Set HPLL phase offset Use: **hpll.phase <value>** <value>: Phase offset in steps of 1/(256xbase rate)
	jitter	Set HPLL jitter bandwidth Use: **hpll.jitter <floor> <ceiling >**
	glider	Set HPLL max glider (slew) rate Use: **hpll.glider <locked> <unlock>**
	clock	Set HPLL clock Use: **hpll.clock <value>**
	coast	Set HPLL coast Use: **hpll.coast <on|off>**
	ref	Set HPLL reference Use: **hpll.ref <value>** <value>: 8-bit value
	enable	Enable HPLL clock Use: **hpll.enable < on|off >**
	dump	Dump HPLL status Use: **hpll.dump**
	reset	Reset HPLL Use: **hpll.reset**

i2c – Inter Integrated Circuit interface

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage i2c
i2c [All targets]	writevalue	Write value to i2c device Use: **i2c.writevalue <target_address> <value>**
	write2value	Write value1 and value2 to i2c device Use: **i2c.writevalue <target_address> <value1> <value2>**
	write	Bulk-write to an i2c device Use: **i2c.write <target_address> <start_address> <src_address src_length>** <target_address>: address of i2c device <start_address>: RAM source address <src_address src_length>: length in bytes
	read	Read from i2c device Use: **i2c.read <target_address> <src_start_address> <src_length>** <target_address>: address of i2c device <src_start_address>: RAM address to hold data <src_length>: length in bytes
	debug	Set without reading Use: i2c.debug **<op> <address> <value>** <op>: <address>: <value>:

irm – Isochronous Resource Manager

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage irm
irm [All targets]	info	Display irm info Use: **irm.info**
	channels	Display irm channel info Use: **irm.channels**
	bandwidth	Display irm bandwidth info Use: **irm.bandwidth**
	allocate	Allocate irm resources channel (64: next available) and bandwidth Use: **rm.allocate <channel> <bandwidth>**
	free	Free irm resources channel and bandwidth Use: **irm.free <channel> <bandwidth>**

kernel – IRQ and Tasks

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage kernel
kernel [All targets]	irq	Dump TCKernel irq registers Use: **kernel.irq**
	task	Dump TCKernel task info Use: **kernel.task**

lal – Link Abstraction Layer

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage lal
lal [All targets]	read1394	1394 read using node handle Use: **lal.read1394 <handle> <offset_hi> <offset_lo> <numBytes>** <numBytes>: max 1024
	write1394	1394 write using node handle Use: **lal.write1394 <handle> <offset_hi> <offset_lo> <numBytes> <quadlet>** <quadlet>: contains data <numBytes>: 1,..,4
	writetest1394	1394 write test using node handle Use: **lal.writetest1394 <handle> <offset_hi> <offset_lo> <numBytes>** where data to be sent is defined in the tool code <numBytes>: max 1024
	lock1394	1394 32-bit lock using node handle Use: **lal.lock1394 <handle> <offset_hi> <offset_lo> <arg> <data>**
	readnode	1394 read using node address Use: **lal.readnode <nodeAddr> <offset_hi> <offset_lo> <numBytes>** <numBytes>: max 1024
	writenode	1394 write using node address Use: **lal.writenode <nodeAddr> <offset_hi> <offset_lo> <numBytes> <quadlet>** <quadlet>: contains data <numBytes>: 1,..,4

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage — lal
	writetestnode	1394 write test using node address Use: **lal.writetestnode <nodeAddr> <offset_hi> <offset_lo> <numBytes>** data to be sent is defined in the tool code
	locknode	1394 lock using node address Use: **lal.locknode <nodeAddr> <offset_hi> <offset_lo> <arg> <data>**
	smr	Show max # of packet bytes from a node's Config ROM nodeCaps Use: **lal.smr <nodeAddr>**
	async1394 [_LOOSE_ISO]	1394 async stream write Use: **lal.async1394 <channel> <tag> <sy> <numBytes>** data to be sent is defined in the tool code
	masync1394 [_LOOSE_ISO]	1394 async stream read. Prints anything that arrives on the given channel 1394 async stream monitor Use: **lal.masync1394 <channel>**
	rng	Display the 1394 addresses which are allocated on this unit Use: **lal.rng**
	setwwuid	set wwuid of this unit Use: **lal.setwwuid <wwuid.high> <wwuid.low> <sw-reset>** <sw-reset>: if true or 1, also causes a bus reset
	getwwuid	get wwuid of this unit Use: **lal.getwwuid**

lhl – Link Hardware Layer

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage — lhl
lhl [All targets]	br	Initiate a bus reset Use: **lhl.br**
	lfr	Initiate a bus reset with force root bit set for this node Use: **lhl.lfr**
	phy	Display one or all Phy register settings Use: **lhl.phy <addr|'all'>**
	stats	Display node lhl statistics Use: **lhl.stats**
	intlog	lhl interrupt log lhl.intlog <mode> Use: **<mode>: dump,dumpifset,dumpifenable,clear**

meter – hardware peak detector configuration

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage (DICE Jr/Mini) meter
sys [All targets]	mode	Set Metering mode. After boot the board will always be in status mode. Use: **meter.mode <off|single|multi>** <off>: Show device status <single>: Show a single meter, see **meter.cfg** <multi>: Show 8 signal-present LED's
	cfg	Set meter 0 to 7 to show either input or output of a certain device and a certain channel. When the meter is in single mode meter=0 is shown. When in multi mode all 8 meters are shown. Default is 4 analog inputs followed by 4 analog outputs. Use: **meter.cfg <meter> <in|out> <device> <channel>** <meter>: 0..7 (0 is used for single mode) <in|out>: in, out <device>: aes, adat, analog, avs1, avs2, mixer0, mixer1, ins0, ins1, apb <channel>: 0..15

mixer8 – On-chip Software Mixer

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage (DICEII) mixer8
mixer8 [All targets]	getmaster	Get register contents for master Use: **mixer8.getmaster** <**register**> <register>: enable, cap, curconf
	setmaster	Set register contents for master Use: **mixer8.setmaster** <**register**> <**value**> <register>: enable, curconf
	getchan	Get register contents for channel registers Use: **mixer8.getchan <index> <register>** <register>: mute, solo, fader, pan, coupled
	setchan	Set register contents for channel registers Use: **mixer8.setchan <index> <register> <value>** <register>: mute, solo, fader, pan, coupled

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage (DICEII) mixer8
	setaux	Set register contents for aux registers Use: **mixer8.setaux <chindex> <auxindex> <register> <value>** **<**register>: on, pre, value
	getaux	Get register contents for aux registers Use: **mixer8.setaux <chindex> <auxindex> <register>** **<**register>: on, pre, value
	setcoef	Set mixer coefs directly Use: **mixer8.setcoef** <**index**> <**value**> <index>: 1,..,64
	dump	Dump Whole mixer Use: **mixer8.dump**
	dumpchan	Dump Whole mixer8 CHANNEL Use: **mixer8.dumpchannel <index>**
	readoverflow	Dump overflow stats

mpm – Memory Pool Manager

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage mpm
mpm [All targets]	dump	Memory Pool Manager dump block information Use: **mpm.dump <dumpType>** where <dumpType> is one of the following tokens: all: all blocks, summary allPB: all blocks, PB details inuse: blocks in use, summary inusePB : blocks in use, PB details avail: available blocks, summary availPB: available blocks, PB details
	dumpByIndex	Memory Pool Manager dump block by index Use: **mpm.dumpByIndex <block size> <block index>**
	stats	Memory Pool Manager statistics Use: **mpm.stats**
	alloc [_MPM_TEST]	allocation of a block of memory of desired size - you will need the address in order to release the block Use: **mpm:alloc <numBytes> <numBlocks>**
	release [_MPM_TEST]	releasing of a previously allocated block Use: **mpm:release <blockAddress>**

mymode – Audio Interface, Routing, Streaming configurations

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage (DICEII) mymode
mymode [dicedriver]	setmode	Set current mode. Stored in non-volatile memory. Changes will take effect on reboot. Use: **mymode.setmode** <**mode**> <mode>: integer value
	dump	Display all available modes Use: **mymode.dump**

nci – Node Controller Interface

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage nci
nci [All targets]	dump	dump handle table Use: **nci.dump**
	create	create a device handle Use: For WWUID handle: **nci.create <wwuid_hi> <wwuid_lo>** nci:create 0x0013f004 0x00401433 For node-address based handle: **nci:create 0 <nodeAddr>** nci:create 0 0xffc1
	close	close a device handle Use: **nci.close <handle>**
	gbi	get bus info Use: **nci.gbi**
	gni	get node info Use: **nci.gni <nodeIndex>**
	gli	get local info Use: **nci.gli**
	gai	get all info Use: **nci.gai**

panel – Panel Subunit

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage panel
panel [_AVC target]	gui	sends avc panel gui update command Use: **panel.gui <ctype> <subfuntion>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <subfunction>: ...
	push	sends avc panel push gui data command Use: **panel.push <ctype> <subfuntion>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <subfunction>: ...
	user	sends avc panel user action command Use: **panel.user <ctype> <subfuntion>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <subfunction>: ...
	pass	sends avc panel pass through command Use: **panel.pass <ctype> <operation_id> <arg>** <ctype>: control:c, status:s, specific inq.:i, notify:n, general inq.:g <operation_id>: [0x00;0x60] key pass-through functions <arg>: 1:key pressed 0:key released (state_flag) <operation_id>: [0x60;0x6a] deterministic functions <arg>: 0:operation specific arguments (n/a) <operation_id>: [0x71;0x75] function keys F1,..,F5 <arg>: 0:(n/a) <operation_id>: 0x7E vendor unique <arg>: 0:(n/a)
	ui	AVC Panel Control command Use: **panel.ui <user operation>** <user operation>: select: Select up: Up down: Down left: Left right: Right r-up: Right-up r-down: Right-down l-up: Left-up l-down: Left-down root: Root menu setup: Setup menu content: Contents menu favorit: Favorite menu

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage
		panel
		exit: Exit 0: 0 1: 1 2: 2 3: 3 4: 4 5: 5 6: 6 7: 7 8: 8 9: 9 dot: Dot enter: Enter clear: Clear sound: Sound Select display: Display information p-up: Page up p-down: Page Down play: Play stop: Stop pause: Pause rew: Rewind ff: Fast forward eject: Eject fward: Forward bward: Backward angle: Angle sub-p: Sub picture f1: F1 f2: F2 f3: F3 f4: F4 f5: F5 vendor: Vendor unique show: Show user operation list

pb – Packet Block Management

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage pb
pb [All targets]	create	create a packet block Use: **pb:create <numPayloadBytes>**
	duplicate	duplicates a packet block Use: **pb:duplicate <original packetBlock address>**
	done	finishes a packet block life Use: **pb:done <original packetBlock address>**

rc – Remote Calls

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage rc
rc [All targets]	id	remote call to cli using nodeid Use: **rc.id <nodeId> <str>**
[deprecated]	handle	remote call to cli using nodeid handle Use: **rc.handle <handle> <str>**
	node	remote call to cli using nodeaddr Use: **rc.node <nodeAddr> <str>**

sys – System Debug Output/Logging Management

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage sys
sys [All targets]	setmask	set debug trace mask Use: **sys.setmask <mask>**
	getmask	get debug trace mask Use: **sys.getmask**
	ismask	is debug trace mask enable Use: **sys.ismask <mask>**
	showmasks	show available debug trace masks Use: **sys.showmasks**

Category [Driver Model]	Command [Additional build flags required, if any]	Description and Usage
		sys
	Trace	Trace functions Use: **sys.trace <start | stop | dump>**
	logprintf [_SYSDEBUGPRINTF_LOG] [deprecated]	set sysPrintf logging mode Use: **sys.logprintf <on | off>**
	syslog [_SYSLOGERROR_LOG] [deprecated]	display all logged sysLogErrors Use: **sys.syslog**
	logmode [_SYSLOGERROR_LOG] [deprecated]	set sysLogErrors logging mode Use: **sys:logmode <on | off>**
	timestamp [_SYSTIMESTAMP] [deprecated]	set syslog timestamp mode Use: **sys:timestamp <on | off>**
	setverbose [deprecated]	set verbose mask Use: **sys:setverbose <mask>**
	getverbose [deprecated]	get verbose mask Use: **sys:getverbose**
	isverbose [deprecated]	is verbose mask enabled Use: **sys:isverbose <mask>**
	showverbose [deprecated]	show available verbose masks Use: **sys:showverbose**

DAL Command Interface Usage

Introduction

The Dice Abstraction Layer module encapsulates many commonly used functions into a single simple API. Your application will consist of many calls to this API, so it is explained here in detail.

Typical interface initialization

The following sequence of commands gives an example of a typical interface initialization sequence:

dal.destroy 0
dal.create 0 low_mid “aes adat” “aes adat”
dal.route 0 adat 0 aes 0 t8
dal.clock 0 aesAny any
dal.start 0

Initial configurations will not take effect until the interface is “started” using dal.start. However, after the interface is started, calls to dal.route and dal.clock will have immediate effect, unless dal.destroy is called first.

DAL Commands

Create

dal.create <interface ID> <rate mode> <input devices> <output devices>

The create function creates an initial configuration for the specified DICE router domain. It can be said that this process creates an “interface”. Note that only one interface can be created per domain. Specific source and destination devices are assigned to the interface along with a clock mode. If a device is assigned as either a source or a destination of an interface it is said to be a “member” of that interface or domain. All devices assigned will initially be muted. The routing itself will initialize to “empty” implying that all destinations will source from the system muted source.

If there is a conflict with the other domain, dal.create will return an error. The proper way to move the membership of a device between domains is to call dal.destroy on the interfaces of each domain and then create them again accordingly.

If dal.create is performed on a domain that has already been “created”, dal.destroy will automatically be called first.

The only way to change the rate mode of a domain is to call dal.create. Changing rate mode might change the channel configuration of the DICE I/O system and in some cases the channel configuration of external devices.

Note that the input and output devices must written as two separate fields each encased in double quotes, as in the following example:
dal.create 0 low “aes adat apb” “aes adat apb”

Create Function Field Descriptions

Function Field	Description
interface ID	Abstracts the concept of the Dice router 0: selects router/sync 1 1: selects router/sync 2
rate mode	Selects the rate mode for the interface low: accept nominal rates from 32k to 48k low_mid: accept nominal rates from 32k to 96k mid: accept nominal rates from 88.2k to 96k high: accept nominal rates from 176.4k to 192k slave_l: slave to interface 0 (only) in low mode slave_m: slave to interface 0 (only) in mid mode slave_h: slave to interface 0 (only) in high mode
input devices	Selects the input device membership for this interface. One or more of: aes, adat, tdif, dsai0, dsai1, dsai2, dsai3, i2s0, i2s1, i2s2, apb, avs1, avs2, avs3, avs4

output devices	Selects the output device membership for this interface. One or more of: aes, adat, tdif, dsai0, dsai1, dsai2, dsai3, i2s0, i2s1, i2s2, apb, avs1, avs2

Create Function Errors

Error	Description
NO_ERROR	No errors occurred
INTERFACE_NOT_DEFINED	The interface ID is not valid
IODEVICE_CONFLICT	The I/O device is used by the other interface

Destroy

dal.destroy <interface ID>

The destroy function destroys the specified interface if it has been created. This involves muting all receiver and transmitter devices belonging to this interface, freeing all resources and clearing the router and device membership. This will also put the clock state machine into the disabled state.

This function is automatically called by dal.create before creating a new configuration.

interface ID - abstracts the concept of the Dice router
0/1 selects Router/Sync 1 or 2 respectively

Destroy Function Field Descriptions

Function Field	Description
interface ID	Abstracts the concept of the Dice router 0: selects router/sync 1 1: selects router/sync 2

Destroy Function Errors

Error	Description
NO_ERROR	No errors occurred
INTERFACE_NOT_DEFINED	The interface ID is not valid

Route

dal.route <interface ID> <output device> <output device channel> <input device> <input device channel> <# of channels to route>

The route function adds a route to a destination of the interface. The source and destination must be members of the interface. Not all routings are possible. There are some restrictions when the AVS transmitters are used as destinations. Refer to router documentation for further details.

The same source can be routed to multiple destinations. For example:
all ADAT -> all AES dal.route 0 aes 0 ADAT 0 t8
and all ADAT -> all ADAT dal.route 0 adat 0 adat 0 t8

Later a route to an interface can be changed:
AVS1 -> all AES dal.route 0 aes 0 avs1 0 t8

Route Function Field Descriptions

Function Field	Description
interface ID	Abstracts the concept of the Dice router 0: selects router/sync 1 1: selects router/sync 2
output device	Selects an output device of the interface to set as the destination for this routing: aes, adat, tdif, dsai0, dsai1, dsai2, dsai3, i2s0, i2s1, i2s2, apb, avs1, avs2
output device channel	Specify lowest channel of the output device to be routed to: 0 - 15
input devices	Selects an input device of the interface to set as the source for this routing: aes, adat, tdif, dsai0, dsai1, dsai2, dsai3, i2s0, i2s1, i2s2, apb, avs1, avs2, avs3, avs4, mute
input device channel	Specify lowest channel of the input device to be routed from: 0-15
# of channels to route	Specify the number of channel to be routed: t1, t2, t4, t8

Route Function Errors

Error	Description

NO_ERROR	No errors occured
INTERFACE_NOT_DEFINED	The interface ID is not valid
INVALID_ROUTE	The route is conflicting with the interface configuration
INTERFACE_NOT_CREATED	dal.create has not been called

Clock

dal.clock <interface ID> <clock source> <nominal clock rate>

The clock function can be called on any interface that has already been created. If the interface is already started this command will have immediate effect, otherwise it will not have an effect until the interface is started.

If the clock source parameter specifies an internal rate the nominal clock rate must be specified. If an external source is selected “any” can be used to indicate that any rate within the selected rate mode (specified with dal.create) will be accepted. If a specific nominal rate is specified only rates which fall within that nominal rates window will be accepted.

Different mechanisms are used to detect the nominal rate depending on the clock source.

Clock Source	Description
aesAny, aes0, aes1, aes2, aes3	The detected rate is based on measuring the clock
adat	The detected rate is based on measuring the clock and inspecting the SMUX status. SMUX can either be set to auto detect (using user bit) or user selected based on dalSetAdatSmuxMode
tdif	The detected rate is based on measuring the clock
wc	The detected rate is based on measuring the clock and inspecting the WC mode. WC mode can either be set to base rate or actual rate based on dalSetWcInMode
avs1, avs2, avs3, avs4	The detected rate is based on measuring the clock and inspecting the SFC field in the CIP1 header.
ext	The detected rate is based on measuring the clock. This is illegal with rate mode set to low_mid as there is no way of determining whether the system should run low or mid.
dsai0, dsai1, dsai2, dsai3	The detected rate is based on measuring the clock
int	The detected rate is derived from the onboard crystal

The actual and nominal clock will always be calculated and are available through various functions.

If a clock is not within the legal rates of the rate mode and the nominal rate setting,

a notification will be posted once and the clock state machine will enter a state waiting for the condition to change.

Clock Function Field Descriptions

Function Field	Description
interface ID	Abstracts the concept of the Dice router 0: selects router/sync 1 1: selects router/sync 2
clock source	Sets the source to be used for clock: aesAny, aes0, aes1, aes2, aes3, adat, tdif, wc, avs1, avs2, avs3, avs4, ext, dsai0, dsai1, dsai2, dsai3, int
nominal clock rate	Sets the nominal rate to be used: 32k, 44k1, 48k, 88k2, 96k, 176k4, 192k, any

Clock Function Errors

Error	Description
NO_ERROR	No errors occurred
INTERFACE_NOT_DEFINED	The interface ID is not valid
INTERFACE_NOT_CREATED	dal.create has not been called

Start

dal.start <interface ID>

After an interface has been created with dal.create all assigned I/O devices will be muted and the clock state machine will be put in disabled state. This function starts the clock state machine and will unmute the I/O when lock is obtained.

Start Function Field Descriptions

Function Field	Description
interface ID	Abstracts the concept of the Dice router 0: selects router/sync 1 1: selects router/sync 2

Start Function Errors

Error	Description
NO_ERROR	No errors occurred
INTERFACE_NOT_DEFINED	The interface ID is not valid
INTERFACE_NOT_CREATED	dal.create has not been called

Dump

dal.dump <interface ID>

Dumps the various configuration and status information for the specified interface.

The full configuration of the specified interface is listed, including the following parameters:

Configuration Parameter	Description
rate mode	Rate mode assigned to the specified interface
clock source	Clock source assigned to the specified interface
nominal rate	Nominal rate assigned to the specified interface
input devices	Input devices assigned to the specified interface
output devices	Output devices assigned to the specified interface

The current status of the interface is also listed, including the following parameters:

Status Parameter	Description
current clock state	Describes whether the clock state machine is locked or waiting for a lock
locked rate mode	The rate mode obtained when locked. This can only be LOW, MID, or HIGH
locked nominal rate	The nominal rate obtained when locked. This rate is never NONE or ANY
locked rate Hz	The actual rate obtained when locked in Hz
bad rate Hz	When the clock state machine enters illegal rate state this field will contain that illegal rate. This is for diagnostics use.

Dump Function Field Descriptions

Function Field	Description
interface ID	Abstracts the concept of the Dice router 0: selects router/sync 1 1: selects router/sync 2