F2802x Burn In Unit

Code Reference Manual

Tue Nov 5 2013

Contents

1 Introduction **1**

1.1 General Operation 1

1.2 Digital Power Control 2

2 Unit Programming **5**

2.1 Language 5

2.2 Instrument Selection 6

2.3 The Command Tree 7

3 Command Reference **9**

3.1 CALibration Subsystem 9

3.2 CONTrol Subsystem 10

3.3 INSTrument Subsystem 10

3.4 OUTPut Subsystem 11

3.5 INPut Subsystem 11

3.6 SOURce Subsystem 11

3.7 MEASure Subsystem 14

3.8 STATus Subsystem 15

3.9 SYSTem 15

4 Data Structure Documentation **17**

4.1 acStageNets Struct Reference 17

4.2 acStageSettings Struct Reference 18

4.3 extDeviceSettings Struct Reference 19

4.4 i2cMsg Struct Reference 20

4.5 loadStageNets Struct Reference 21

4.6 loadStageSettings Struct Reference 22

4.7 xfmrStageNets Struct Reference 23

4.8 xfmrStageSettings Struct Reference 24

5 File Documentation **27**

5.1 Adc.h File Reference 27

5.2 Cntl.h File Reference 30

5.3 Common.h File Reference 36
5.4 Comparator.h File Reference 37
5.5 EnableCtrl.h File Reference 39
5.6 I2c.h File Reference 44
5.7 MacroNets.h File Reference 48
5.8 Ocp.h File Reference 53
5.9 Opp.h File Reference 58
5.10 Otp.h File Reference 61
5.11 Ovp.h File Reference 67
5.12 Pwm.h File Reference 72
5.13 Sci.h File Reference 73
5.14 SCPI_specificCmds.h File Reference 75
5.15 Settings.h File Reference 76
5.16 SineGen.h File Reference 81
5.17 slaveMode.h File Reference 85
5.18 SlewControl.h File Reference 87
5.19 Spi.h File Reference 91
5.20 StateMachine.h File Reference 94
5.21 Timers.h File Reference 95
5.22 Tmp.h File Reference 96

Index **99**

Chapter 1

Introduction

This document forms the reference manual for the Burn In Unit based on the Texas Instruments F2802x.

This system has been implemented using several Texas Instruments libraries; digital power v3.1, SGen V1.01, IQ-Math v1.5c and SQMath. Please refer to the TI documentation pertaining to those libraries and to the C28x device to understand their operation.

This system also uses the System for Standard Commands for Programmable Instruments for C28x, again, refer to the documentation pertaining to that system to understand its use.

1.1 General Operation

This system operates using the concept of a state machine with multiplexed threading. This allows important tasks, such as signal generation and the monitoring of critical parameters to take precedence while still allowing other less critical tasks to run with acceptable frequency. This is allowed by creating several threads DPL, A, B and C and iterating them at different interval periods, such that every time the thread is iterated the next single task in that thread is run.

1.1.1 Fast (DPL) Thread

The DPL thread is the fastest thread, is coded in assembly and consists of only two tasks. It is the only thread that uses an interrupt to iterate. Thus the tasks are formed by an interrupt service routine (ISR). This allows the DPL thread to interrupt the ongoing action of any of the other threads. This means that as long as the length of time each of the tasks take to execute is less than the interval time, then the threads operation will be deterministic. It is within this thread's two tasks that the critical and time-sensitive actions of the program are carried out. The thread ISR is triggered by a start of conversion (SOC) signal from the master enhanced pulse width modulation (ePWM) peripheral. This occurs at a frequency of 33 kHz. As there are two tasks on the thread, the frequency of each task is 16.5 kHz resulting in a period of 60us.

1.1.2 Sequential Iteration Threads

The remaining threads (A, B, C) iterate by checking the rollover state of different CPU timers. Their operation and iteration is sequential and so less deterministic. The outline of the handling sequential iteration threads by the state machine is illustrated by the following flow diagram, where N and n rollover to A and 0, respectively whenever they reach their maximum value.

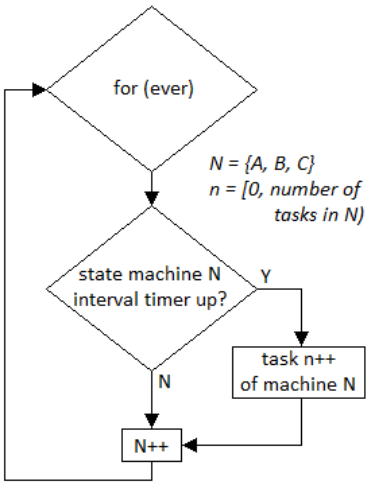


Figure 1.1: Sequential iteration thread state machine

The frequency of tasks can be determined from the period of the interval timers using the following equation, where n is the number of tasks in the thread and T_{in} is the interval timer period.

$$f_{ta} = \frac{1}{T_{in} \times n}$$

With interval periods such that $T_A < T_B < T_A$ this results in the flow of execution illustrated by the following figure.

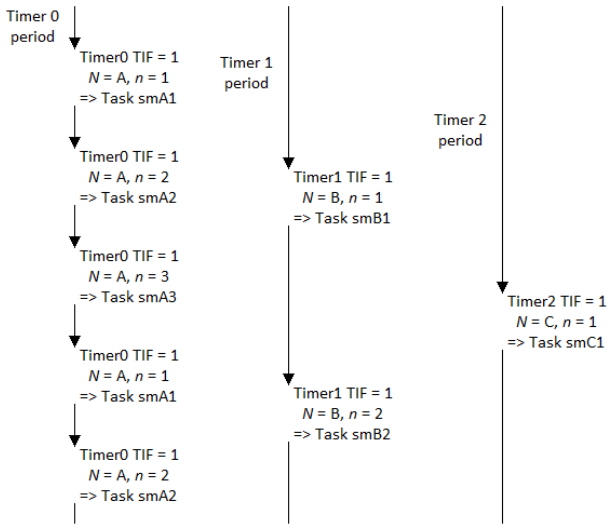


Figure 1.2: Sequence of thread tasks

1.2 Digital Power Control

The control of the power, voltage and current is managed using control loop logic, normally one loop for each stage (Transformer, AC, each of the four loads). These control loops are formed from a set of macros and helper functions.

Each of these macros acts as a block of functionality, much like a black-box electronic component, allowing their terminals to be wired together in the required order with nets. Once initialised and configured, or connected, correctly the macro blocks are executed, sequentially, by calling the appropriate code from an assembly ISR. In this case, this is what forms the DPL ISR tasks. Each block performs a precisely defined computation and delivers the

result to the appropriate net-list variables. Each control loop's net vairables and operation parameter variables are grouped together and stored in a structure (channelParameters), with one such structure for each control loop.

The basic use of macros that are available within the program can be broken down into three parts:

1. Initialisation: the macro blocks are initialised from the program by using a C callable function, DPL_Init(), which is contained within the assembly file DPL_ISR.asm.
2. Configuration: C pointers of the macro block terminals are assigned to net nodes to form the desired control structure, i.e., the control loop.
3. Execution: Macro block code is executed in the assembly ISR, DPL_ISR().

1.2.1 Boost Load Control

The following figure shows a diagram that illustrates the macro and net view of the control loop for one of the load boost converter stages. The loop depicted is current controlled. It uses a macro to retrieve a reading from an ADC peripheral to obtain a value that indicates the level of the current at the low side of the load input, CHANn I_{SNS} . This value is used as the feedback into a 2-pole 2-zero filter macro where it is compared to the user-set reference value and the output value is varied up or down, using the filter coefficients, as needed to move it towards matching the reference. This output value is then used by another macro to set the duty of the ePWM peripheral that controls the switching of the converter, CHANn PWM.

Another macro retrieves a value from a second ADC that represents the voltage level across the load input, CHANn V_{SNS} . A task in one of the sequential iteration threads compares this value to the user-set allowable voltage limit, the OVP limit. If the value is above this limit the control loop output is disabled and an over-current protection alert is raised.

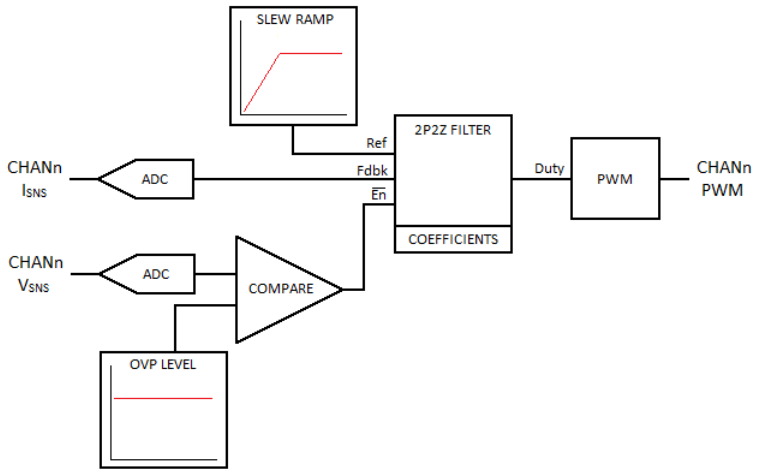


Figure 1.3: Macro view of control loop for single boost load

1.2.2 Inter-Boost Control

The following figure shows a diagram that illustrates the macro and net view of the control loop for the inter-boost, or "middle", boost converter stage. The loop depicted is voltage controlled. It uses a macro to retrieve a reading from an ADC peripheral to obtain a value that indicates the level of the voltage across the stage output, DC HV V_{SNS} . This value is used as the feedback into a 3-pole 3-zero filter macro where it is compared to the user-set reference value and the output value is varied up or down, using the filter coefficents, as needed to move towards matching the reference. This output value is then used by another macro to set the duty of the ePWM peripheral that controls the switching of the converter, INTBST PWM.

Another macro retrieves a value from a second ADC that represents the current level at the low side of the input, DC MID I_{SNS} . An on-board analogue comparator compares this value against the user-set allowable current limit,

the OCP limit. If the value is above this limit the comparator output sets the related trip-zone, quickly disabling the control loop output in software and hardware, HV EN.

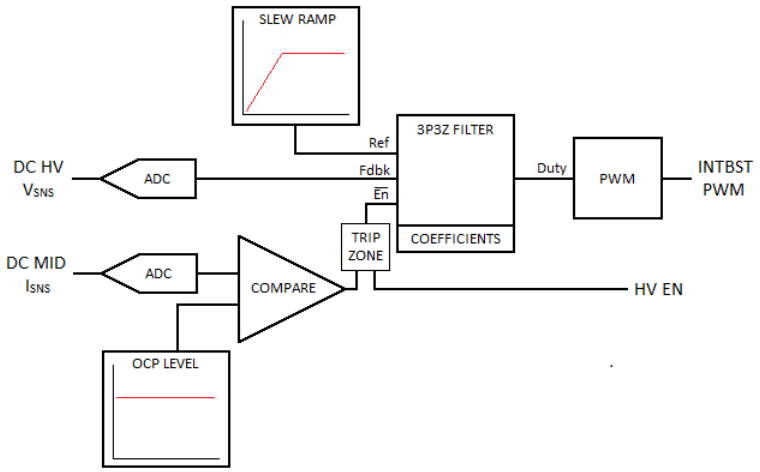


Figure 1.4: Macro view of control loop for inter-boost

1.2.3 AC Bridge Control

The following figure shows a diagram that illustrates the macro and a net view of the control loop for the AC bridge converter stage. The loop depicted is both voltage *and* current controlled. It uses a macro to retrieve a reading from an ADC peripheral to obtain a value that indicates the voltage level across the stage output, AC V_{SNS} . This value is used as the feedback into a 3-pole 3-zero filter macro where it is compared to the generated sine wave reference and the output value is varied up or down, using the filter coefficients, as needed to move towards matching the reference.

Another macro retrieves a value from a second ADC that represents the current level at the low side of the stage input, AC I_{SNS} . This value is used as the feedback into a 2-pole 2-zero filter macro where it is compared to the output of the 3-pole 3-zero filter macro and the output is varied up or down, using the filter coefficents, as needed to move towards matching the reference.

An on board analogue comparator also compares this AC I_{SNS} value against the user-set allowable current limit, the OCP limit. If the value is above the limit, the comparator output sets the related trip zone, quickly disabling the control loop output and all other stages output, STOP ALL.

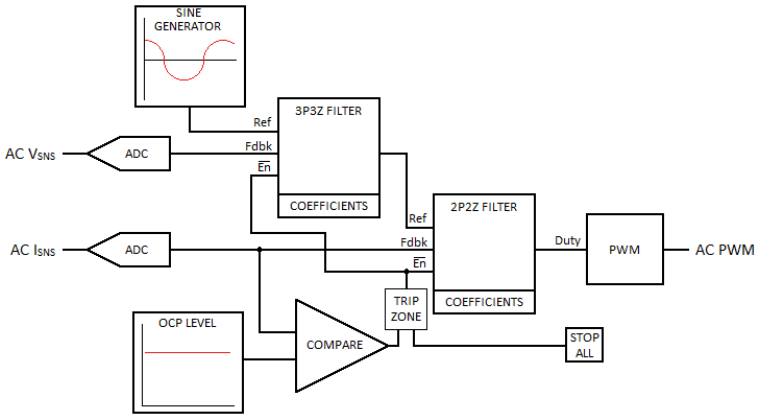


Figure 1.5: Macro view of control loop for AC bridge

Chapter 2

Unit Programming

2.1 Language

For remote programmming operations the burn in unit uses commands that follow the SCPI syntax as outlined in the System for Standard Commands for Programmable Instruments for C28x Code Reference Manual, where in the standard programmming commands that are included by this system are also outlined. Please refer to that document and, where necessray, any further referenced documents, for inforrmation on the syntax of the programming commands.

This unit programming chapter will focus on the function of the commands specific to the burn in unit

2.1.1 Format

The burn in unit uses a standard raw TCP/IP Ethernet connection to communicate with a controlling system. Once the user's control system has established a valid raw TCP/IP connection with the burn in unit commands should be sent and received as ASCII character byte string data, with the most significant byte first.

Thus the data sent that would correspond to the <COMMON QUERY PROGRAM HEADER> `*IDN`?; in transmission order would be (including the '^NL' <PROGRAM MESSAGE TERMINATOR>)

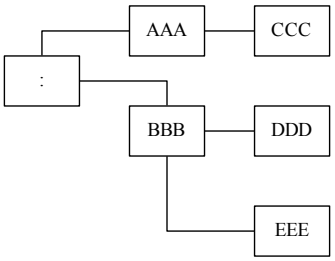
```
0x2A 0x49 0x44 0x4E 0x3F 0x3B 0x0A
```

2.1.2 Program Message Unit Creation

The user may create a <PROGRAM MESSAGE UNIT> by traversing the command tree from the root towards the desired function and concatenating the <program mnemonic>s and <COMMAND PROGRAM HEADER> or <QUERY PROGRAM HEADER> stepped through to reach that destination.

For example, using the hypothetical command tree shown in the following figure, the <PROGRAM MESSAGE UNIT> required to run the function 'EEE' would be:

```
BBB:EEE;
```



2.1.3 Queries and Responses

Each <PROGRAM MESSAGE> may contain only one <QUERY PROGRAM HEADER> which, if used, should be the last header in the message. If a <QUERY PROGRAM HEADER> is used the next operation should be to read the response generated by the query, before any further <PROGRAM MESSAGE> elements are sent to the unit.

2.2 Instrument Selection

Each burn in unit is split into several logical instruments that corresponds to the functional sections of the burn in unit. The instruement selection is controlled using the `INSTrument` sub-system These instruments are numbered according to the unit channel numbering shown in the following list. These numbers can be obtained by using the `CATalog`? query from the `INSTrument` subsystem:

```
INSTrument:CATalog?;.
```

0. Load 0
1. Load 1
2. Load 2
3. Load 3
4. AC Current Control
5. DC Stage

6. AC Stage

This means that before setting any of the instruments' settings, the relevant instrument number must first be selected. This is done using the `NSELect` command from the `INSTrument` subsystem. The default instrument selection is 0. Once selected the value is "sticky", and does not need to be set again until the user settings wishes to send commands for a different instrument. Thus, for example, the <PROGRAM MESSAGE> required to set the voltage level DC stage to 10 volts with a range set to 1V/V would be as follows:

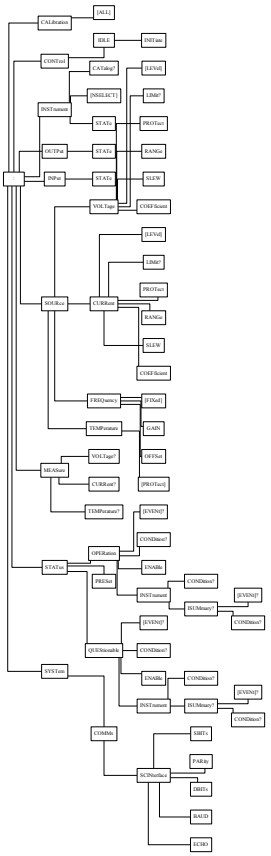
```
INSTrument:NSELect 5;:SOURce:VOLTage:LEVel 10;:SOURce:VOLTage:RA-
NGe 1;
```

2.3 The Command Tree

The command tree provides the structure of the subsystems and their commands. As shown however the diagram does not illustrate that some of the commands have no meaning to some instruments. For example, attempting to set the voltage coefficients of a current controlled instrument will result in an error.

Again, the commands shown here are in addition to those that are implemented as standard by the System for Standard Commands for Programmable Instruments for C28x.

The diagram is followed by a listing of each command. The next section, Command Reference explains the operation and details of each command.



CALibration
ALL
CONTrol
IDLE

```
        INITiate
 INSTrument
    CATalog?
    NSELect
    STATe
 OUTPut
    STATe
 INPut
    STATe
 SOURce
    VOLTage
        LEVel
        LIMit?
        PROTect
        RANGe
        SLEW
        COEFficient
    CURRent
        LEVel
        LIMit?
        PROTect
        RANGe
        SLEW
        COEFficient
    FREQuency
        FIXed
        GAIN
        OFFSet
    TEMPerature
        PROTect
 MEASure
    VOLTage?
    CURRent?
    TEMPerature?
 STATus
    OPERation
    PRESet
    QUEStionable
 SYSTem
    COMMs
        SCINterface
            SBITs
            PARity
            DBITs
            BAUD
            ECHO
```

Chapter 3

Command Reference

This section outlines the operation of each command and node on the command tree. All commands, unless otherwise specified, also have an equivalent query form that returns the value that the command is currently set to. For example, querying SOURce:VOLTage:PROTect? will return the current voltage protection level setting.

Some commands have a query form *only*, that is they have no none-query form. these commands are identified by the presence of the query terminator character, '?'.

Some of the commands are optional, or default. Such optionality is represented by the presence of square brackets enclosing the command, '[' and ']'. Where commands are optional, it is possible to use the parent node alone instead of needing to append the optional child command. For example, as the :NSELect command from the INSTrument subsystem is marked as optional, the command to select an instrument can be either:

```
INSTrument:NSELect <numeric_value>;
```

or

```
INSTrument <numeric_value>;
```

3.1 CALibration Subsystem

This subsystem has the function of performing system calibration. Currently the subsystem has only one function, the default command ALL and it's complementing query.

3.1.1 [:ALL]

```
CALibration:ALL
```

The CALibration:ALL command performs the same function as the CALibration:ALL? command except there is no response. Calibration errors are reported through the status-reporting mechanism. While this command is executing the CALibrating bit of the Operation Status Condition register will be set.

3.1.2 [:ALL]?

```
Calibration:ALL?
```

The ALL? query performs a full calibration of the instrument and responds with a <numeric_value> indicating the success of the calibration. A zero will be returned if calibration is completed successfully; otherwise a nonzero value which represents the appropriate error number shall be returned. An instrument will still report calibration errors through the status-reporting mechanism, even though an error is reported by the value of the query response.

3.2 CONTrol Subsystem

The CONTrol subsystem is used to turn on and off or control the state of the entire device.

3.2.1 :IDLE

`CONTrol:IDLE`

This node contains commands that control the IDLE state of the device.

3.2.1.1 :INITiate

`CONTrol:IDLE:INITiate`

Returns the device to the idle state. This command is an event and has no associated ∗RST condition or query command.

3.3 INSTrument Subsystem

This device uses multiple logical instruments (as described in Instrument Selection), this subsystem provides the mechanism to identify and select logical instruments by number.

3.3.1 :CATalog?

`INSTrument:CATalog?`

The CATalog query returns a comma-separated list of the numbers of all logical instruments.

3.3.2 :NSELect <numeric_value>

`INSTrument:NSELect`

This command selects an instrument according to the numeric value parameter. When a logical instrument is selected, all other logical instruments or groups are unavailable for programming until selected. When queried it shall return the selected logical instrument number.

Note that the numbering used for logical instruments directly corresponds to the numbers used in status reporting for the multiple instruments; specifically the STATus:QUEStionable:INSTrument and STATus:OPERation:INSTrument commands.

At ∗RST, instrument 0 is selected.

3.3.3 :STATe <Boolean>

`INSTrument:STATe`

Turns the selected logical instrument ON or OFF. A logical instrument does not have to be turned OFF before another logical instrument is selected. That is, several logical instruments may be active, while only one may be selected. When an instrument is active, measurements occur and signals are generated. When inactive, measurements do not occur and signals are not generated. When a logical instrument is selected, yet is inactive, all commands shall be processed so that the logical instrument shall reflect the state changes requested when the logical instrument is turned on.

At ∗RST, this value is OFF.

3.4 OUTPut Subsystem

The OUTPut subsystem controls the characteristics of the selected instrument's output port.

3.4.1 [:STATe] <Boolean>

`OUTPut:STATe`

Selects the state of the output. When STATe is ON the signal generated by the source is emitted from the output port of the selected instrument.

3.5 INPut Subsystem

The INPut subsystem controls the characteristics of the selected instrument's input port.

3.5.1 :STATe <Boolean>

`INPut:STATe`

Selects the state of the input. When STATe is ON the signal generated by the source is emitted from the output port of the selected instrument.

3.6 SOURce Subsystem

The SOURce setup commands are divided into several sections. Each section or subsystem deals with controls that directly affect device-specific settings of the selected instrument.

3.6.1 :VOLTage

This subsection controls the signal voltage characteristics of the source.

3.6.1.1 [:LEVel] <numeric_value>

`SOURce:VOLTage:LEVel`

This command sets the target voltage level of a voltage controlled instrument. The numeric value parameter is the level to be set in volts, between 0 and the voltage limit returned in response the LIMit? query.

3.6.1.2 :LIMit?

`SOURce:VOLTage:LEVel`

This command has a query form only. This is because the voltage limit value is fixed. The query response is a single numeric value in volts representing this fixed limit.

3.6.1.3 :PROTect <numeric_value>

`SOURce:VOLTage:PROTect`

This command sets the voltage protection level. This is the level at which the over-voltage protection is triggered if the voltage rises above it The numeric value parameter is the level to be set in volts, between 0 and the voltage limit returned in response to the LIMit? query.

3.6.1.4 :RANGe <numeric_value>

`SOURce:VOLTage:RANGe`

This command sets the voltage range. The numeric value parameter is the voltage scaling value in volts-per-volt from 0 to 1.

3.6.1.5 :SLEW <numeric_value>

`SOURce:VOLTage:SLEW`

This command sets the voltage slew for a voltage controlled instrument. The numeric value parameter is the voltage slew rate value in volts-per-microseconds from 0 to 1.

3.6.1.6 :COEFficient <numeric_value>, <numeric value>

`SOURce:VOLTage:COEFficient`

This command is used to specify the voltage control coefficients, of a voltage controlled instrument. The first numeric value parameter specifies the coefficient to be changed, according to the following list, while the second numeric value parameter specifies the level the selected coefficient is to be changed to.

0. Saturation Minimum
1. Saturation Maximum
2. B0
3. B1
4. A1
5. B2
6. A2
7. B3
8. A3

When queried, the response is a list of comma separated numeric values in the following order:

Saturation-Minimum-value, Saturation-Maximum-value, B0-value, B1-value, A1-value, B2-value, A2-value, B3-value, A3-value

3.6.2 :CURRent

`SOURce:CURRent`

This subsection controls the signal current characteristics of the source.

3.6.2.1 [:LEVel] <numeric_value>

`SOURce:CURRent:LEVel`

This command sets the target current level of a current controlled instrument. The numeric value parameter is the level to be set in amps, between 0 and the current limit returned in response the LIMit? query.

3.6.2.2 :LIMit?

`SOURce:CURRent:LEVel`

This command has a query form only. This is because the current limit value is fixed. The query response is a single numeric value in amps representing this fixed limit.

3.6.2.3 :PROTect <numeric_value>

`SOURce:VOLTage:PROTect`

This command sets the current protection level. This is the level at which the over-current protection is triggered if the current rises above it The numeric value parameter is the level to be set in amps, between 0 and the current limit returned in response to the LIMit? query.

3.6.2.4 :RANGe <numeric_value>

`SOURce:CURRent:RANGe`

This command sets the current range. The numeric value parameter is the current scaling value in amps-per-volt from 0 to 1. **<-—!!!-—**

3.6.2.5 :SLEW <numeric_value>

`SOURce:CURRent:SLEW`

This command sets the current slew for a current controlled instrument. The numeric value parameter is the current slew rate value in amps-per-microseconds from 0 to 1.

3.6.2.6 :COEFficient <numeric_value>, <numeric_value>

`SOURce:CURRent:COEFficient`

This command is used to specify the current control coefficients, of a current controlled instrument. The first numeric value parameter specifies the coefficient to be changed, according to the following list, while the second numeric value parameter specifies the level the selected coefficient is to be changed to.

0. Saturation Minimum
1. Saturation Maximum
2. B0
3. B1
4. A1
5. B2
6. A2

When queried, the response is a list of comma separated numeric values in the following order:

Saturation-Minimum-value, Saturation-Maximum-value, B0-value, B1-value, A1-value, B2-value, A2-value

3.6.3 :FREQuency

`SOURce:FREQuency`

The FREQuency subsystem controls the frequency characteristics of an instrument source that produces a periodic signal - such as the sine wave produced by the AC stage.

3.6.3.1 [:FIXed] <numeric_value>

`SOURce:FREQuency:FIXed`

This command is used to set the source frequency. The numeric value parameter should be the required frequency in hertz, from 0 to 1000.

3.6.3.2 :GAIN <numeric_value>

`SOURce:FREQuency:GAIN`

This command is used to set the frequency gain. The numeric value parameter should vary from 0 to 1

3.6.3.3 :OFFSet <numeric_value>

`SOURce:FREQuency:OFFSet`

This command sets the frequency offset. The numeric value parameter should vary between -0.5 and +0.5

3.6.4 :TEMPerature

`SOURce:TEMPerature`

The temperature subsystem controls the temperature of the selected logical instrument. Note that not all instruments have temperature control functionality.

3.6.4.1 [:PROTect] <numeric_value>

`SOURce:TEMPerature:PROTect`

This command sets the temperature protection level. This is the level at which the over-temperature protection is triggered if the temperature rises above it The numeric value parameter is the level to be set in degrees Celsius, between 0 and 150 ^{o} C.

3.7 MEASure Subsystem

The MEASure susbsytem allows the user to take measurements of certain aspects of the selected instruments operation.

3.7.1 :VOLTage?

`MEASure:VOLTage?` This query responds with a numeric value of the current voltage reading from the selected instrument in volts.

3.7.2 :CURRent?

`MEASure:CURRent?` This query responds with a numeric value of the current electrical current reading from the selected instrument in amps.

3.7.3 :TEMPerature?

`MEASure:TEMPerature?` This query responds with a numeric value of the current temperature reading from the selected instrument in degrees Celsius.

3.8 STATus Subsystem

3.8.1 :OPERation

3.8.2 :PRESet

3.8.3 :QUEStionable

3.9 SYSTem

The SYSTem subsystem collects the functions that are not related to instrument performance.

3.9.1 :COMMs

`SYSTem:COMMs`

This subsystem controls some aspects of the serial communications (SCI) interface.

3.9.1.1 :SBITs <numeric_value>

`SYSTem:COMMs:SBITs`

This command sets the number of stop bits used in the SCI frame. The numeric value parameter sets the number of stop bits and may be either 1 or 2. The default value is 1.

3.9.1.2 :PARity <numeric_value>

`SYSTem:COMMs:PARity`

This command sets what parity is used in the SCI frames. The defualt selection is for no parity. The numeric value parameter sets the parity in use and corresponds to parity settings as described by the following list:

0. No parity
1. Odd parity
2. Even parity

3.9.1.3 :DBITs <numeric_value>

`SYSTem:COMMs:DBITs`

This command sets how many data bits are in the SCI frame. The numeric value perameter sets the number of data bits and may vary from 1 to 8.

3.9.1.4 :BAUD <numeric_value>

SYSTem:COMMs:BAUD

This command sets the baud rate used by the SCI interface. The numeric value parameter sets the baud rate. The allowed parameters are any of the standard baud rates which are listed in the table below. Acceptance of a baud rate does not indicate that the device is capable of use such a baud rate. The default baud rate is 9600.

Accepted Baud Rates	Accepted Baud Rates	Accepted Baud Rates
110	9600	57600
300	14400	115200
600	19200	230400
1200	28800	460800
2400	38400	921600
4800	56000	

3.9.1.5 :ECHO <Boolean>

SYSTem:COMMs:ECHO

This command enables or disables the echo functionality of the SCI interface. If the boolean parameter is ON the interface echos any data it receives without passing that data on to any other part of the device. The default setting is OFF.

Chapter 4

Data Structure Documentation

4.1 acStageNets Struct Reference

`#include <MacroNets.h>`

Data Fields

- volatile int32 vRefNet
- volatile int32 vFdbkNet
- volatile int32 iFdbkNet
- volatile int32 iRefNet
- volatile int32 iFiltOutNet

4.1.1 Detailed Description

A structure that contains the nets used in the AC stage.

4.1.2 Field Documentation

4.1.2.1 volatile int32 acStageNets::iFdbkNet

Current feedback net (IQ24).

4.1.2.2 volatile int32 acStageNets::iFiltOutNet

Current IIR filter control law output net (IQ24).

4.1.2.3 volatile int32 acStageNets::iRefNet

IIR filter control law current reference net (IQ24).

4.1.2.4 volatile int32 acStageNets::vFdbkNet

Voltage feedback net (IQ24).

4.1.2.5 volatile int32 acStageNets::vRefNet

IIR filter control law voltage reference net (IQ24).

The documentation for this struct was generated from the following file:

- MacroNets.h

4.2 acStageSettings Struct Reference

`#include <MacroNets.h>`

Data Fields

- int32 gainRate
- int32 gainTarget
- int32 ocpLevel
- int32 ovpLevel
- int32 otpLevel
- int16 iMaxRms
- int16 vMaxRms
- int16 iScale
- int16 vScale
- int16 vGainLmt
- slaveMode mode
- Uint16 acFrequency
- Uint16 enable

4.2.1 Detailed Description

A structure that contains the settings that pertain to the AC stage.

4.2.2 Field Documentation

4.2.2.1 Uint16 acStageSettings::acFrequency

Sine signal generator frequency setting (Hz).

4.2.2.2 Uint16 acStageSettings::enable

Channel enable status {FALSE, TRUE}.

4.2.2.3 int32 acStageSettings::gainRate

Reference sine gain rate (IQ24).

4.2.2.4 int32 acStageSettings::gainTarget

Reference sine gain target (IQ24).

4.2.2.5 int16 acStageSettings::iMaxRms

Maximum RMS current setting limit (SQ10).

4.2.2.6 int16 acStageSettings::iScale

Current scaling setting in volts-per-amp for scaling between a voltage level measured by an ADC to a real current value (SQ14).

4.2.2.7 slaveMode acStageSettings::mode

Master or slave mode select.

4.2.2.8 int32 acStageSettings::ocpLevel

Normalised OCP limit (IQ24).

4.2.2.9 int32 acStageSettings::otpLevel

OTP limit in ° C (SQ7).

4.2.2.10 int32 acStageSettings::ovpLevel

Normalised OVP limit (IQ24).

4.2.2.11 int16 acStageSettings::vGainLmt

Sine signal generator voltage gain limit (SQ14).

4.2.2.12 int16 acStageSettings::vMaxRms

Maximum RMS voltage setting limit (SQ10).

4.2.2.13 int16 acStageSettings::vScale

Voltage scaling setting in volts-per-volts for scaling between a voltage level measured by an ADC to a real voltage value (SQ14).

The documentation for this struct was generated from the following file:

- MacroNets.h

4.3 extDeviceSettings Struct Reference

```
#include <MacroNets.h>
```

Data Fields

- int32 ext1OtpLevel
- int32 ext2OtpLevel
- Uint16 extFanEnable
- Uint16 extPsuEnable

4.3.1 Detailed Description

A structure that contains the settings that pertain to external devices.

4.3.2 Field Documentation

4.3.2.1 int32 extDeviceSettings::ext1OtpLevel

External 1 OTP limit in ° C (SQ7).

4.3.2.2 int32 extDeviceSettings::ext2OtpLevel

External 2 OTP limit in ° C (SQ7).

4.3.2.3 Uint16 extDeviceSettings::extFanEnable

External fan enable status {TRUE | FALSE}.

4.3.2.4 Uint16 extDeviceSettings::extPsuEnable

External PSU enable status {TRUE | FALSE}.

The documentation for this struct was generated from the following file:

- MacroNets.h

4.4 i2cMsg Struct Reference

`#include <I2c.h>`

Data Fields

- volatile Uint16 msgStatus
- Uint16 slaveAddress
- Uint16 numOfBytes
- Uint16 numSlavePtrBytes
- Uint16 slavePtrAddrHigh
- Uint16 slavePtrAddrLow
- Uint16 msgBuffer [I2C_MAX_BUFFER_SIZE]

4.4.1 Detailed Description

The structure used to contain all settings and values relevant to a particular I2C message.

4.4.2 Field Documentation

4.4.2.1 Uint16 i2cMsg::msgBuffer[I2C_MAX_BUFFER_SIZE]

A buffer array for message data. The maximum buffer size, MAX_BUFFER_SIZE, is 4 due to the FIFO's size.

4.4.2.2 volatile Uint16 i2cMsg::msgStatus

Indicates which state the message is in.

4.4.2.3 Uint16 i2cMsg::numOfBytes

The number of valid bytes in (or to be put in msgBuffer).

4.4.2.4 Uint16 i2cMsg::numSlavePtrBytes

The number of slave register pointer address bytes.

4.4.2.5 Uint16 i2cMsg::slaveAddress

The slave device I2C address this message is intended for.

4.4.2.6 Uint16 i2cMsg::slavePtrAddrHigh

The slave register pointer high byte.

4.4.2.7 Uint16 i2cMsg::slavePtrAddrLow

The slave register pointer low byte.

The documentation for this struct was generated from the following file:

- I2c.h

4.5 loadStageNets Struct Reference

`#include <MacroNets.h>`

Data Fields

- volatile int32 iFdbkNet
- volatile int32 vFdbkNet
- volatile int32 iRefNet
- volatile int32 iFiltOutNet

4.5.1 Detailed Description

A structure that contains the nets used in a single load stage.

4.5.2 Field Documentation

4.5.2.1 volatile int32 loadStageNets::iFdbkNet

Current feedback net (IQ24).

4.5.2.2 volatile int32 loadStageNets::iFiltOutNet

IIR filter control law output net (IQ24).

4.5.2.3 volatile int32 loadStageNets::iRefNet

IIR filter control law current reference net (IQ24).

4.5.2.4 volatile int32 loadStageNets::vFdbkNet

Voltage feedback net (IQ24).

The documentation for this struct was generated from the following file:

- MacroNets.h

4.6 loadStageSettings Struct Reference

```
#include <MacroNets.h>
```

Data Fields

- int32 slewRate
- int32 slewTarget
- int32 ocpLevel
- int32 ovpLevel
- int32 oppLevel
- int32 otpLevel
- int16 iMax
- int16 vMax
- int16 iScale
- int16 vScale
- Uint16 enable

4.6.1 Detailed Description

A structure that contains the settings that pertain to a single load stage.

4.6.2 Field Documentation

4.6.2.1 Uint16 loadStageSettings::enable

Channel enable status {FALSE, TRUE}.

4.6.2.2 int16 loadStageSettings::iMax

Maximum DC current setting limit (SQ10).

4.6.2.3 int16 loadStageSettings::iScale

Current scaling setting in volts-per-amp for scaling between a voltage level measured by an ADC to a real current value (SQ14).

4.6.2.4 int32 loadStageSettings::ocpLevel

Normalised OCP limit (IQ24).

4.6.2.5 int32 loadStageSettings::oppLevel

Normalised OPP limit (IQ22).

4.6.2.6 int32 loadStageSettings::otpLevel

OTP limit in ° C (SQ7).

4.6.2.7 int32 loadStageSettings::ovpLevel

Normalised OVP limit (IQ24).

4.6.2.8 int32 loadStageSettings::slewRate

IIR filter control law reference slew rate (IQ24).

4.6.2.9 int32 loadStageSettings::slewTarget

IIR filter control law reference slew target (IQ24).

4.6.2.10 int16 loadStageSettings::vMax

Maximum DC voltage setting limit (SQ10).

4.6.2.11 int16 loadStageSettings::vScale

Voltage scaling setting in volts-per-volts for scaling between a voltage level measured by an ADC to a real voltage value (SQ14).

The documentation for this struct was generated from the following file:

- MacroNets.h

4.7 xfmrStageNets Struct Reference

```
#include <MacroNets.h>
```

Data Fields

- volatile int32 iSnsNet
- volatile int32 midVSnsNet
- volatile int32 hvVSnsNet
- volatile int32 pwmDutyNet

4.7.1 Detailed Description

A structure that contains the nets used in the transformer stage.

4.7.2 Field Documentation

4.7.2.1 volatile int32 xfmrStageNets::hvVSnsNet

DC HV voltage sense net (IQ24).

4.7.2.2 volatile int32 xfmrStageNets::iSnsNet

Current sense net (IQ24).

4.7.2.3 volatile int32 xfmrStageNets::midVSnsNet

DC mid voltage sense net (IQ24).

4.7.2.4 volatile int32 xfmrStageNets::pwmDutyNet

Transformer PWM duty net (IQ24).

The documentation for this struct was generated from the following file:

- MacroNets.h

4.8 xfmrStageSettings Struct Reference

```
#include <MacroNets.h>
```

Data Fields

- int32 ocpLevel
- int32 hvOvpLevel
- int32 otpLevel
- int16 iMax
- int16 midVMax
- int16 hvVMax
- int16 iScale
- int16 midVScale
- int16 hvVScale
- Uint16 enable

4.8.1 Detailed Description

A structure that contains the settings that pertain to the transformer stage.

4.8.2 Field Documentation

4.8.2.1 Uint16 xfmrStageSettings::enable

Channel enable status {FALSE, TRUE}.

4.8.2.2 int32 xfmrStageSettings::hvOvpLevel

Normalised DC HV OVP limit (IQ24).

4.8.2.3 int16 xfmrStageSettings::hvVMax

Maximum DC HV voltage setting limit (SQ10).

4.8.2.4 int16 xfmrStageSettings::hvVScale

DC HV voltage scaling setting in volts-per-volts for scaling between a voltage level measured by an ADC to a real voltage value (SQ14).

4.8.2.5 int16 xfmrStageSettings::iMax

Maximum DC current setting limit (SQ10).

4.8.2.6 int16 xfmrStageSettings::iScale

Current scaling setting in volts-per-amp for scaling between a voltage level measured by an ADC to a real current value (SQ14).

4.8.2.7 int16 xfmrStageSettings::midVMax

Maximum DC mid voltage setting limit (SQ10).

4.8.2.8 int16 xfmrStageSettings::midVScale

DC mid voltage scaling setting in volts-per-volts for scaling between a voltage level measured by an ADC to a real voltage value (SQ14).

4.8.2.9 int32 xfmrStageSettings::ocpLevel

Normalised OCP limit (IQ24).

4.8.2.10 int32 xfmrStageSettings::otpLevel

OTP limit in $^{\circ}$ C (SQ7).

The documentation for this struct was generated from the following file:

- MacroNets.h

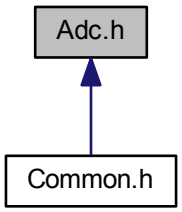
Chapter 5

File Documentation

5.1 Adc.h File Reference

ADC functions.

This graph shows which files directly or indirectly include this file:



Functions

- void initAdc (void)
- Uint16 getLoadCurrent (loadStage load, float32 *dcCurrent)
- Uint16 getLoadVoltage (loadStage load, float32 *dcVoltage)
- Uint16 getDcMidCurrent (float32 *dcCurrent)
- Uint16 getDcMidVoltage (float32 *dcVoltage)
- Uint16 getDcHvVoltage (float32 *dcVoltage)

Variables

- volatile int32 * ADCDRV_1ch_Rlt1
- volatile int32 * ADCDRV_1ch_Rlt2
- volatile int32 * ADCDRV_1ch_Rlt3
- volatile int32 * ADCDRV_1ch_Rlt4
- volatile int32 * ADCDRV_1ch_Rlt5
- volatile int32 * ADCDRV_1ch_Rlt6
- volatile int32 * ADCDRV_1ch_Rlt7

- volatile int32 * ADCDRV_1ch_Rlt8
- volatile int32 * ADCDRV_1ch_Rlt9
- volatile int32 * ADCDRV_1ch_Rlt10
- volatile int32 * ADCDRV_1ch_Rlt11
- volatile int32 * ADCDRV_1ch_Rlt12
- volatile int32 * ADCDRV_1ch_Rlt13

5.1.1 Detailed Description

ADC functions.

5.1.2 Function Documentation

5.1.2.1 Uint16 getDcHvVoltage (float32 * *dcVoltage*)

Queries the most recent voltage reading from the DC HV's associated ADC.

Parameters

out	*dcVoltage*	Address of the memory location at which to place the query result (volts).

Returns

Error status.

5.1.2.2 Uint16 getDcMidCurrent (float32 * *dcCurrent*)

Queries the most recent current reading from the DC Mid's associated ADC.

Parameters

out	*dcCurrent*	Address of the memory location at which to place the query result (amps).

Returns

Error status.

5.1.2.3 Uint16 getDcMidVoltage (float32 * *dcVoltage*)

Queries the most recent voltage reading from the DC Mid's associated ADC.

Parameters

out	*dcVoltage*	Address of the memory location at which to place the query result (volts).

Returns

Error status.

5.1.2.4 Uint16 getLoadCurrent (loadStage *load*, float32 * *dcCurrent*)

Queries the most recent current reading from the specified load's associated ADC.

Parameters

in	*load*	Specifies the load on which the reading is to be queried.
out	*dcCurrent*	Address of the memory location at which to place the query result (amps).

Returns

Error status.

5.1.2.5 Uint16 getLoadVoltage (loadStage *load*, float32 ∗ *dcVoltage*)

Queries the most recent voltage reading from the specified load's associated ADC.

Parameters

in	*load*	Specifies the load on which the reading is to be queried.
out	*dcVoltage*	Address of the memory location at which to place the query result (volts).

Returns

Error status.

5.1.2.6 void initAdc (void)

Configures the ADC's SOCs then calls pwmSocConfigure().

- SHOULD be run after initPWM().
- SHOULD be run before DPL_INIT().

5.1.3 Variable Documentation

Address of the mOOIRV_c) Errorvoltent voltrenss the-278(SO)-2775(minalef)30(poi7T)-25278(ADC)30(.)]TJ0.25 g 0.25 G/F51 8.9634

5.1.3.6 volatile int32* ADCDRV_1ch_Rlt2

Channel 1 current sense ADC terminal pointer.

5.1.3.7 volatile int32* ADCDRV_1ch_Rlt3

Channel 2 current sense ADC terminal pointer.

5.1.3.8 volatile int32* ADCDRV_1ch_Rlt4

Channel 3 current sense ADC terminal pointer.

5.1.3.9 volatile int32* ADCDRV_1ch_Rlt5

Interboost current sense ADC terminal pointer.

5.1.3.10 volatile int32* ADCDRV_1ch_Rlt6

AC stage current sense ADC terminal pointer.

5.1.3.11 volatile int32* ADCDRV_1ch_Rlt7

Channel 0 voltage sense ADC terminal pointer.

5.1.3.12 volatile int32* ADCDRV_1ch_Rlt8

Channel 1 voltage sense ADC terminal pointer.

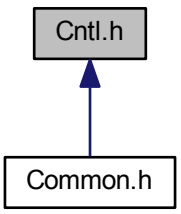
5.1.3.13 volatile int32* ADCDRV_1ch_Rlt9

Channel 2 voltage sense ADC terminal pointer.

5.2 Cntl.h File Reference

DPLib CNTL Macro related helper functions.

This graph shows which files directly or indirectly include this file:



Typedefs

- typedef enum coefNum cfType

Enumerations

- enum coefNum { ,
  cMin = firstCoef, cMax, cB0, cB1,
  cA1, cB2, cA2 }

Functions

- void initCoefs (void)
- Uint16 setLoadICoef (loadStage load, cfType coef, float32 value)
- Uint16 getLoadICoef (loadStage load, cfType coef, float32 *value)
- Uint16 setAcICoef (cfType coef, float32 val)
- Uint16 getAcICoef (cfType coef, float32 *value)
- Uint16 setAcVCoef (cfType coef, float32 value)
- Uint16 getAcVCoef (cfType coef, float32 *value)

Variables

- struct CNTL_2P2Z_CoefStruct loadICoefs [numberOfLoads]
- struct CNTL_2P2Z_CoefStruct acICoefs
- struct CNTL_2P2Z_CoefStruct acVCoefs
- volatile int32 * CNTL_2P2Z_Coef1
- volatile int32 * CNTL_2P2Z_Coef2
- volatile int32 * CNTL_2P2Z_Coef3
- volatile int32 * CNTL_2P2Z_Coef4
- volatile int32 * CNTL_2P2Z_Fdbk1
- volatile int32 * CNTL_2P2Z_Fdbk2
- volatile int32 * CNTL_2P2Z_Fdbk3
- volatile int32 * CNTL_2P2Z_Fdbk4
- volatile int32 * CNTL_2P2Z_Out1
- volatile int32 * CNTL_2P2Z_Out2
- volatile int32 * CNTL_2P2Z_Out3
- volatile int32 * CNTL_2P2Z_Out4
- volatile int32 * CNTL_2P2Z_Ref1
- volatile int32 * CNTL_2P2Z_Ref2
- volatile int32 * CNTL_2P2Z_Ref3
- volatile int32 * CNTL_2P2Z_Ref4
- volatile int32 * CNTL_2P2Z_Coef5
- volatile int32 * CNTL_2P2Z_Fdbk5
- volatile int32 * CNTL_2P2Z_Out5
- volatile int32 * CNTL_2P2Z_Ref5
- volatile int32 * CNTL_2P2Z_Coef6
- volatile int32 * CNTL_2P2Z_Fdbk6
- volatile int32 * CNTL_2P2Z_Out6
- volatile int32 * CNTL_2P2Z_Ref6

5.2.1 Detailed Description

DPLib CNTL Macro related helper functions.

5.2.2 Typedef Documentation

5.2.2.1 typedef enum coefNum cfType

A type that allows a reference to a CNTL coefficient.

5.2.3 Enumeration Type Documentation

5.2.3.1 enum coefNum

CNTL Coefficient references

Enumerator

cMin Saturation minimum reference.

cMax Saturation maximum reference.

cB0 B0 coefficient reference.

cB1 B1 coefficient reference.

cA1 A1 coefficient reference.

cB2 B2 coefficient reference.

cA2 A2 coefficient reference.

5.2.4 Function Documentation

5.2.4.1 Uint16 getAclCoef (cfType *coef,* float32 * *value*)

Queries the specified IIR filter control law coefficient for the AC current control.

Parameters

in	*coef*	Specifies the coefficient to be queried [cMin, cA3).
out	*value*	Address of the memory location at which to place the query result.

Returns

Error status.

5.2.4.2 Uint16 getAcVCoef (cfType *coef,* float32 * *value*)

Queries the specified IIR filter control law coefficient for the AC voltage control.

Parameters

in	*coef*	Specifies the coefficient to be queried [cMin, cA3).
out	*value*	Address of the memory location at which to place the query result.

Returns

Error status.

5.2.4.3 Uint16 getLoadICoef (loadStage *load,* cfType *coef,* float32 * *value*)

Queries the specified IIR filter control law coefficient for a specified load's current control.

Parameters

in	*load*	Specifies the load on which the setting is to be queried [0, NUM_CHNLS).
in	*coef*	Specifies the coefficient to be queried [cMin, cA3).
out	*value*	Address of the memory location at which to place the query result.

Returns

Error status.

5.2.4.4 void initCoefs (void)

Initialises all IIR filter control law coefficients.

Warning

This function must be called before the control macros are used.

5.2.4.5 Uint16 setAcICoef (cfType *coef,* float32 *val*)

Sets the specified IIR filter control law coefficient for the AC current control.

Parameters

in	*coef*	Specifies the coefficient to be set [cMin, cA3).
in	*val*	Specifies the coefficient value to be applied. Should be between the minimum and maximum values for the specific coefficient as defined by cfLmts[][].

Returns

Error status.

5.2.4.6 Uint16 setAcVCoef (cfType *coef,* float32 *value*)

Sets the specified IIR filter control law coefficient for the AC voltage control.

- The actual setting in use is not updated until AFTER cntlUpdateCoefs() has been called.

  Parameters

in	*coef*	Specifies the coefficient to be set [cMin, cA3).
in	*value*	Specifies the coefficient value to be applied. Should be between the minimum and maximum values for the specific coefficient as defined by cfLmts[][].

  Returns

  Error status.

5.2.4.7 Uint16 setLoadICoef (loadStage *load,* cfType *coef,* float32 *value*)

Sets the specified IIR filter control law coefficient for a specified load's current control.

Parameters

in	*load*	Specifies the load number the setting is to be applied to [0, NUM_CHNLS).
in	*coef*	Specifies the coefficient to be set [cMin, cA3).
in	*value*	Specifies the coefficient value to be applied. Should be between the minimum and maximum values for the specific coefficient as defined by cfLmts[][].

Returns

Error status.

5.2.5 Variable Documentation

5.2.5.1 struct CNTL_2P2Z_CoefStruct acICoefs

Structure that holds the AC I 2-pole 2-zero IIR filter control law coefficients currently in use.

5.2.5.2 struct CNTL_2P2Z_CoefStruct acVCoefs

Structure that holds the AC V 2-pole 2-zero IIR filter control law coefficients currently in use.

5.2.5.3 volatile int32∗ CNTL_2P2Z_Coef1

Load 1 IIR filter control law coefficient terminal pointer.

5.2.5.4 volatile int32∗ CNTL_2P2Z_Coef2

Load 2 IIR filter control law coefficient terminal pointer.

5.2.5.5 volatile int32∗ CNTL_2P2Z_Coef3

Load 3 IIR filter control law coefficient terminal pointer.

5.2.5.6 volatile int32∗ CNTL_2P2Z_Coef4

Load 4 IIR filter control law coefficient terminal pointer.

5.2.5.7 volatile int32∗ CNTL_2P2Z_Coef5

AC stage I IIR filter control law coefficient terminal pointer.

5.2.5.8 volatile int32∗ CNTL_2P2Z_Coef6

AC stage V IIR filter control law coefficient terminal pointer.

5.2.5.9 volatile int32∗ CNTL_2P2Z_Fdbk1

Load 1 IIR filter control law feedback terminal pointer.

5.2.5.10 volatile int32* CNTL_2P2Z_Fdbk2

Load 2 IIR filter control law feedback terminal pointer.

5.2.5.11 volatile int32* CNTL_2P2Z_Fdbk3

Load 3 IIR filter control law feedback terminal pointer.

5.2.5.12 volatile int32* CNTL_2P2Z_Fdbk4

Load 4 IIR filter control law feedback terminal pointer.

5.2.5.13 volatile int32* CNTL_2P2Z_Fdbk5

AC stage I IIR filter control law feedback terminal pointer.

5.2.5.14 volatile int32* CNTL_2P2Z_Fdbk6

AC stage V IIR filter control law feedback terminal pointer.

5.2.5.15 volatile int32* CNTL_2P2Z_Out1

Load 1 IIR filter control law output terminal pointer.

5.2.5.16 volatile int32* CNTL_2P2Z_Out2

Load 2 IIR filter control law output terminal pointer.

5.2.5.17 volatile int32* CNTL_2P2Z_Out3

Load 3 IIR filter control law output terminal pointer.

5.2.5.18 volatile int32* CNTL_2P2Z_Out4

Load 4 IIR filter control law output terminal pointer.

5.2.5.19 volatile int32* CNTL_2P2Z_Out5

AC stage I IIR filter control law output terminal pointer.

5.2.5.20 volatile int32* CNTL_2P2Z_Out6

AC stage V IIR filter control law output terminal pointer.

5.2.5.21 volatile int32* CNTL_2P2Z_Ref1

Load 1 IIR filter control law reference terminal pointer.

5.2.5.22 volatile int32* CNTL_2P2Z_Ref2

Load 2 IIR filter control law reference terminal pointer.

5.2.5.23 volatile int32* CNTL_2P2Z_Ref3

Load 3 IIR filter control law reference terminal pointer.

5.2.5.24 volatile int32* CNTL_2P2Z_Ref4

Load 4 IIR filter control law reference terminal pointer.

5.2.5.25 volatile int32* CNTL_2P2Z_Ref5

AC stage I IIR filter control law reference terminal pointer.

5.2.5.26 volatile int32* CNTL_2P2Z_Ref6

AC stage V IIR filter control law reference terminal pointer.

5.2.5.27 struct CNTL_2P2Z_CoefStruct loadICoefs[numberOfLoads]

Array of structures that hold the load 2-pole 2-zero IIR filter control law coefficients currently in use.

5.3 Common.h File Reference

Common include file for the project.

```
#include "Settings.h"
#include "PeripheralHeaderIncludes.h"
#include "DSP2802x_EPWM_defines.h"
#include "slaveMode.h"
#include "IQMathLib.h"
#include "SQMath.h"
#include "DPlib.h"
#include "sgen.h"
#include "I2c.h"
#include "Sci.h"
#include "Spi.h"
#include "MacroNets.h"
#include "Timers.h"
#include "Adc.h"
#include "Comparator.h"
#include "Pwm.h"
#include "Cntl.h"
#include "Ocp.h"
#include "Ovp.h"
#include "Opp.h"
#include "Otp.h"
#include "SlewControl.h"
#include "SineGen.h"
#include "tmp.h"
#include "EnableCtrl.h"
#include "StateMachine.h"
#include "../../SCPI_Build/SCPI_Build/scpi/scpi.h"
#include "SCPI_specificCmds.h"
```

Include dependency graph for Common.h:

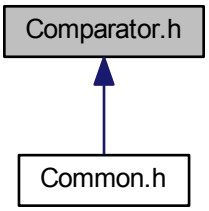
5.3.1 Detailed Description

Common include file for the project. All other header files used should be included within this file and this file should then be used to include them in the required source files.

5.4 Comparator.h File Reference

Comparator, DAC and trip zone functions.

This graph shows which files directly or indirectly include this file:



Functions

- void initTripZone (void)
- void initDcComparator (void)
- void rstDcTripzone (void)
- void initAcComparator (void)
- Uint16 setAcDac (float32 level)
- Uint16 getAcDac (float32 *level)
- void rstAcTripzone (void)

5.4.1 Detailed Description

Comparator, DAC and trip zone functions. Requires the modification of the COMP_REGS struct of the DSP2802x_GlobalVariableDefs in the file DSP2802x_Comp.h, to allow use of the DACCTL and ramp-related register unions. Use the equivalent file from the f2903x includes for reference.

5.4.2 Function Documentation

5.4.2.1 Uint16 getAcDac (float32 * *level*)

Queries the output level setting of the DAC on the inverting input of the DC comparator. iScale MUST be set before use.

Parameters

out	*level*	Pointer to location at which to place the query result (amps).

Returns

Error status.

5.4.2.2 void initAcComparator (void)

Initialises the AC comparator (COMP 1) using an internal DAC at the inverting input.

- SHOULD be called AFTER adcSocCnf().
- SHOULD be called BEFORE PWMS (SYNC) are started.
- SHOULD be called BEFORE pwmTZConfigure().

5.4.2.3 void initDcComparator (void)

Initialises the DC comparator (COMP 2) using an internal DAC at the inverting input.

- SHOULD be called AFTER adcSocCnf().
- SHOULD be called BEFORE PWMS (SYNC) are started.
- SHOULD be called BEFORE pwmTZConfigure().
- midVScale should be set before use.

5.4.2.4 void initTripZone (void)

Configures PWM trip zones for use. Requires the comparator and DAC to be configured

See Also

adc.h

5.4.2.5 void rstAcTripzone (void)

Resets the AC trip zone after an AC comparator event.

5.4.2.6 void rstDcTripzone (void)

Resets the DC trip zone after an DC comparator event.

5.4.2.7 Uint16 setAcDac (float32 *level*)

Sets the output level of the DAC on the inverting input of the AC comparator. The voltage and iScale MUST be set before use.

Parameters

in	*level*	Specifies the value of the level setting to be applied (amps).

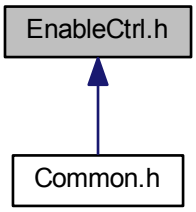
Returns

Error status.

5.5 EnableCtrl.h File Reference

Functions for enabling and disabling circuit sections via I2C.

This graph shows which files directly or indirectly include this file:



Macros

- #define IOE_SW4A_STATE 0x00
- #define IOE_SW4B_STATE 0x00
- #define IOE_I2C_BASE_ADDR 0x20
- #define IOE_IODIR_ADDR 0x00
- #define IOE_IPOL_ADDR 0x01
- #define IOE_GPINTEN_ADDR 0x02
- #define IOE_DEFVAL_ADDR 0x03
- #define IOE_INTCON_ADDR 0x04
- #define IOE_IOCON_ADDR 0x05
- #define IOE_GPPU_ADDR 0x06
- #define IOE_INTF_ADDR 0x07
- #define IOE_INTCAP_ADDR 0x08
- #define IOE_GPIO_ADDR 0x09
- #define IOE_OLAT_ADDR 0x0A
- #define IOE_MAX_CHNL 0x08
- #define IOE_I2C_ADDR (IOE_I2C_BASE_ADDR | (IOE_SW4A_STATE << 2)) | (IOE_SW4B_STATE << 3)
- #define ALL_DISABLED_WORD

Typedefs

- typedef enum ecSection circuitSection

Enumerations

- enum ecSection

Functions

- Uint16 initEnableControl (void)
- void resetEnableControl (void)
- Uint16 enableCircuit (circuitSection section)
- Uint16 disableCircuit (circuitSection section)
- Uint16 getCurrentState (circuitSection section, Uint16 *state)

5.5.1 Detailed Description

Functions for enabling and disabling circuit sections via I2C. Circuit sections are controlled via an external I/O expander (MCP23008) that is connected to the I2C bus at address 0100xx0 where 'xx' is determined by the state of switch 4 (SW4) on the Ctrl PCB, which the user should record in the macros IOE_SW4A_STATE and IOE_SW4-B_STATE.

After initialisation or reset all sections default to disabled.

Warning

Before any enable control functions can be used the I2C peripheral MUST be initialised, followed by enable control initialisation, both of which will require global interrupts to be enabled.

See Also

initI2c()

ADS7830 Ch #	Signal	Enable bit#	Enable state
GP0	CHAN1EN	0	1
GP1	CHAN2EN	1	1
GP2	CHAN3EN	2	1
GP3	CHAN4EN	3	1
GP4	HVENABLE	4	0
GP5	ACENABLE	5	1
GP6	PSUENBLE	6	1
GP7	EXTFANEN	7	1

5.5.2 Macro Definition Documentation

5.5.2.1 #define ALL_DISABLED_WORD

Value:

```
(~(0 | (CHAN1_EN_STATE << chan1) | (CHAN2_EN_STATE << chan2) | (CHAN3_EN_STATE << chan3) \
                                | (CHAN4_EN_STATE << chan4) | (XFMR_EN_STATE << xfmrCct) | (AC_EN_STATE <<
      acCct) \
                                | (PSU_EN_STATE << psu) | (EXTFAN_EN_STATE << fan))) & 0x00FF
```

Uses the defined states to define a word for the case where all circuits are disabled

5.5.2.2 #define IOE_DEFVAL_ADDR 0x03

MCP23008 I/O expander default value register address.

5.5.2.3 #define IOE_GPINTEN_ADDR 0x02

MCP23008 I/O expander interrupt on change enable register address.

5.5.2.4 #define IOE_GPIO_ADDR 0x09

MCP23008 I/O expander GPIO port register address.

5.5.2.5 #define IOE_GPPU_ADDR 0x06

MCP23008 I/O expander pull-up resistor configuration register address.

5.5.2.6 #define IOE_I2C_ADDR (IOE_I2C_BASE_ADDR | (IOE_SW4A_STATE << 2)) | (IOE_SW4B_STATE << 3)

MCP23008 I/O expander complete I2C address.

5.5.2.7 #define IOE_I2C_BASE_ADDR 0x20

MCP23008 I/O expander base I2C address.

5.5.2.8 #define IOE_INTCAP_ADDR 0x08

MCP23008 I/O expander interrupt capture register address.

5.5.2.9 #define IOE_INTCON_ADDR 0x04

MCP23008 I/O expander interrupt on change control register address.

5.5.2.10 #define IOE_INTF_ADDR 0x07

MCP23008 I/O expander interrupt flag register address.

5.5.2.11 #define IOE_IOCON_ADDR 0x05

MCP23008 I/O expander configuration register address.

5.5.2.12 #define IOE_IODIR_ADDR 0x00

MCP23008 I/O expander I/O direction register address.

5.5.2.13 #define IOE_IPOL_ADDR 0x01

MCP23008 I/O expander input polarity register address.

5.5.2.14 #define IOE_MAX_CHNL 0x08

MCP23008 I/O expander number of channels (0 - 7).

5.5.2.15 #define IOE_OLAT_ADDR 0x0A

MCP23008 I/O expander output latch register address.

5.5.2.16 #define IOE_SW4A_STATE 0x00

The state of switch 4a; ON = 0x01, OFF = 0x00.

5.5.2.17 #define IOE_SW4B_STATE 0x00

The state of switch 4b; ON = 0x01, OFF = 0x00.

5.5.3 Typedef Documentation

5.5.3.1 typedef enum ecSection circuitSection

A type that allow specification of an enable control circuit section.

5.5.4 Enumeration Type Documentation

5.5.4.1 enum ecSection

The possible enable control circuit section selections. Also defines bit order.

5.5.5 Function Documentation

5.5.5.1 Uint16 disableCircuit (circuitSection *section*)

Disables the specified circuit section enable signal.

Parameters

in	*section*	Specifies the circuit section that is to be disabled.

Returns

Error status.

5.5.5.2 Uint16 enableCircuit (circuitSection *section*)

Enables the specified circuit section enable signal.

Parameters

in	*section*	Specifies the circuit section that is to be enabled.

Returns

Error status.

5.5.5.3 Uint16 getCurrentState (circuitSection *section*, Uint16 * *state*)

Gets the current state of the specified circuit section.

Parameters

in	*section*	Specifies the circuit section for which the state is to be queried.
out	*state*	Pointer to location at which to replace query result (Disabled = 0, enabled = !0).

Returns

Error status.

5.5.5.4 Uint16 initEnableControl (void)

Initialises the enable control interface. The I2C peripheral MUST be initialised before this function is used.

See Also

i2cInit()

Returns

Error status.

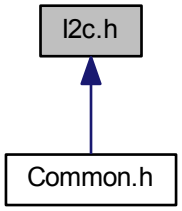
5.5.5.5 void resetEnableControl (void)

Resets and reinitialises the MCP23008 I/O Expander device.

5.6 I2c.h File Reference

Inter-integrated circuit communications functions.

This graph shows which files directly or indirectly include this file:



Data Structures

- struct i2cMsg

Macros

- #define I2C_MAX_BUFFER_SIZE 0x04
- #define I2C_MAX_PTR_SIZE 0x02

- #define I2C_CLR_AL_BIT 0x0001
- #define I2C_CLR_NACK_BIT 0x0002
- #define I2C_CLR_ARDY_BIT 0x0004
- #define I2C_CLR_RRDY_BIT 0x0008
- #define I2C_CLR_SCD_BIT 0x0020

- #define I2C_ARDY_ISRC 0x0003

- #define I2C_SCD_ISRC 0x0006

- #define I2C_MSGSTAT_INACTIVE 0x0000
- #define I2C_MSGSTAT_SEND_WITHSTOP 0x0010
- #define I2C_MSGSTAT_WRITE_BUSY 0x0011
- #define I2C_MSGSTAT_SEND_NOSTOP 0x0020
- #define I2C_MSGSTAT_SEND_NOSTOP_BUSY 0x0021
- #define I2C_MSGSTAT_RESTART 0x0022
- #define I2C_MSGSTAT_READ_BUSY 0x0023

Functions

- void initI2c (void)
- void i2cPopMsg (i2cMsg *msg, Uint16 msgStatus, Uint16 slaveAddr, Uint16 numDataBytes, Uint16 numSlavePtrBytes, Uint16 slavePtrAddrHi, Uint16 slavePtrAddrLo)
- Uint16 i2cWrite (i2cMsg *msg)
- Uint16 i2cRead (i2cMsg *msg)

5.6.1 Detailed Description

Inter-integrated circuit communications functions.

Warning

The following bridge tracks should be cut on the TI C2000 LaunchPad XL PCB before I2C (or SPI) may be used:

Bridge to Cut	GPIO	PCB Pin
JP5	GPIO32	J2-6
JP7	GPIO33	J2-7
JP8	GPIO16	J6-7
JP10	GPIO17	J6-8

This should result in the following functionality:

Function	GPIO	PCB Pin
I2C_SDAA	GPIO32	J6-7
I2C_SCLA	GPIO33	J6-8
SPI_MOSI	GPIO16	J2-6
SPI_MISO	GPIO17	J2-7

The function i2cInit() MUST be called before any other public I2C function is used. This will clear any values already in the I2C registers.
Interrupts MUST be globally enabled for the functions i2cWrite() and i2cRead() to operate correctly.

See Also

EnableCtrl.h
Tmp.h

5.6.2 Macro Definition Documentation

5.6.2.1 #define I2C_ARDY_ISRC 0x0003

I2C Interrupt Sources Register access ready condition I2C interrupt source.

5.6.2.2 #define I2C_CLR_AL_BIT 0x0001

I2C Status Clear Bits Arbitration lost status clear bit.

5.6.2.3 #define I2C_CLR_ARDY_BIT 0x0004

Register access ready status clear bit.

5.6.2.4 #define I2C_CLR_NACK_BIT 0x0002

NACK status clear bit.

5.6.2.5 #define I2C_CLR_RRDY_BIT 0x0008

Receive data ready status clear bit.

5.6.2.6 #define I2C_CLR_SCD_BIT 0x0020

Stop detected status clear bit.

5.6.2.7 #define I2C_MAX_BUFFER_SIZE 0x04

Maximum I2C message buffer size in bytes, including slave register pointer bytes.

5.6.2.8 #define I2C_MAX_PTR_SIZE 0x02

Maximum number of slave register pointer bytes.

5.6.2.9 #define I2C_MSGSTAT_INACTIVE 0x0000

I2C Message States Inactive I2C message state.

5.6.2.10 #define I2C_MSGSTAT_READ_BUSY 0x0023

State indicating the I2C is busy with a read.

5.6.2.11 #define I2C_MSGSTAT_RESTART 0x0022

Transmit a master read with a restart.

5.6.2.12 #define I2C_MSGSTAT_SEND_NOSTOP 0x0020

Transmit a write with no stop.

5.6.2.13 #define I2C_MSGSTAT_SEND_NOSTOP_BUSY 0x0021

State indicating the I2C is busy with a write with no stop.

5.6.2.14 #define I2C_MSGSTAT_SEND_WITHSTOP 0x0010

Transmit a write with stop I2C message state.

5.6.2.15 #define I2C_MSGSTAT_WRITE_BUSY 0x0011

State indicating the I2C is busy with a write with a stop.

5.6.2.16 #define I2C_SCD_ISRC 0x0006

Stop detected condition I2C interrupt source.

5.6.3 Function Documentation

5.6.3.1 void i2cPopMsg (i2cMsg * *msg,* Uint16 *msgStatus,* Uint16 *slaveAddr,* Uint16 *numDataBytes,* Uint16 *numSlavePtrBytes,* Uint16 *slavePtrAddrHi,* Uint16 *slavePtrAddrLo*)

This function can be used to validate and populate the specified settings and values into the specified I2C message structure.

Parameters

out	*msg*	The I2C message structure.
in	*msgStatus*	The initial I2C message status.
in	*slaveAddr*	The slave address.
in	*numDataBytes*	The number, if any, of data bytes, above any slave register pointer bytes, in the message.
in	*numSlavePtr-Bytes*	The number, if any, of slave register pointer bytes.
in	*slavePtrAddrHi*	The slave register pointer high byte. If only one byte, or none, (as indicated by numSlavePtrbytes) is to be used leave this at zero.
in	*slavePtrAddrLo*	The slave register pointer low byte. If no pointer bytes (as indicated by num-SlavePtrbytes) are used leave this at zero.

5.6.3.2 Uint16 i2cRead (i2cMsg * *msg*)

Starts an I2C-A read using the settings specified. Read bytes are saved to the buffer msg.msgBuffer[].

Parameters

in	*msg*	The I2C message struct.

Returns

Error status.

5.6.3.3 Uint16 i2cWrite (i2cMsg * *msg*)

Starts an I2C-A write using the settings and values specified.

Parameters

in	*msg*	The I2C message structure.

Returns

Error Status.

5.6.3.4 void initI2c (void)

Initialises the I2C-A peripheral and relevant interrupts.

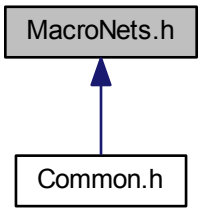
Warning

This function will clear any values already in the I2C peripheral registers.
This function MUST be called before any other public I2C function.

5.7 MacroNets.h File Reference

DPLib macro net and value control functions.

This graph shows which files directly or indirectly include this file:



Data Structures

- struct extDeviceSettings
- struct loadStageNets
- struct loadStageSettings
- struct xfmrStageNets
- struct xfmrStageSettings
- struct acStageNets
- struct acStageSettings

Macros

- #define LOAD_0 0
- #define LOAD_1 1
- #define LOAD_2 2
- #define LOAD_3 3
- #define XFMR_STAGE 4
- #define AC_STAGE 5
- #define EXT_1 6
- #define EXT_2 7

Typedefs

- typedef enum extSelect extSelect
- typedef struct extDeviceSettings extDeviceSettings
- typedef enum loadStage loadStage
- typedef struct loadStageNets loadStageNets
- typedef struct loadStageSettings loadStageSettings
- typedef struct xfmrStageNets xfmrStageNets
- typedef struct xfmrStageSettings xfmrStageSettings
- typedef struct acStageNets acStageNets
- typedef struct acStageSettings acStageSettings

Enumerations

- enum extSelect { ext1 = 0, ext2 = 1 }
- enum loadStage { load1 = 0, load2 = 1, load3 = 2, load4 = 3 }

Functions

- void setupNets (slaveMode mode)
- void stopAll (void)
- Uint16 runAll (void)

Variables

- Uint16 stopAllFlag
- Uint16 enableAllFlag
- extDeviceSettings extSettings
- loadStageNets loadNets [numberOfLoads]
- loadStageSettings loadSettings [numberOfLoads]
- xfmrStageNets xfmrNets
- xfmrStageSettings xfmrSettings
- acStageNets acNets
- acStageSettings acSettings

5.7.1 Detailed Description

DPLib macro net and value control functions.

5.7.2 Macro Definition Documentation

5.7.2.1 #define AC_STAGE 5

The index position for AC stage settings.

5.7.2.2 #define EXT_1 6

External temperature 1 channel reference.

5.7.2.3 #define EXT_2 7

External temperature 2 channel reference.

5.7.2.4 #define LOAD_0 0

The index position for Load 0 settings.

5.7.2.5 #define LOAD_1 1

The index position for Load 1 settings.

5.7.2.6 #define LOAD_2 2

The index position for Load 2 settings.

5.7.2.7 #define LOAD_3 3

The index position for Load 3 settings.

5.7.2.8 #define XFMR_STAGE 4

The index position for transformer stage settings.

5.7.3 Typedef Documentation

5.7.3.1 typedef struct acStageNets acStageNets

A type to allow use of the AC stage nets structure.

5.7.3.2 typedef struct acStageSettings acStageSettings

A type to allow use of the AC stage settings structure.

5.7.3.3 typedef struct extDeviceSettings extDeviceSettings

A type to allow use of the external device settings structure.

5.7.3.4 typedef enum extSelect extSelect

A type that allow specification of an external sensor.

5.7.3.5 typedef enum loadStage loadStage

A type that allow specification of a load.

5.7.3.6 typedef struct loadStageNets loadStageNets

A type to allow use of the load stage nets structure.

5.7.3.7 typedef struct loadStageSettings loadStageSettings

A type to allow use of the load stage settings structure.

5.7.3.8 typedef struct xfmrStageNets xfmrStageNets

A type to allow use of the transformer stage nets structure.

5.7.3.9 typedef struct xfmrStageSettings xfmrStageSettings

A type to allow use of the AC stage settings structure.

5.7.4 Enumeration Type Documentation

5.7.4.1 enum extSelect

The possible external sensor selections.

Enumerator

ext1 External sensor 1.

ext2 External sensor 2.

5.7.4.2 enum loadStage

The possible load stage selections.

Enumerator

load1 Load 1.

load2 Load 2.

load3 Load 3.

load4 Load 4.

5.7.5 Function Documentation

5.7.5.1 Uint16 runAll (void)

Enables all IIR filter control law reference inputs.

Returns

Error status.

5.7.5.2 void setupNets (slaveMode *mode*)

Initialises macro settings and nets. IIR control law coefficients are handled outside of this function.

Warning

This MUST be called after DPL_INIT()
This MUST be called after pwmMacroConfig()

Parameters

in	*mode*	Selects the test unit mode to allow correct macro net connection.

5.7.5.3 void stopAll (void)

Disables all circuit sections, zeroes all IIR references and flags each stage as disabled.

5.7.6 Variable Documentation

5.7.6.1 acStageNets acNets

A global instance of the AC nets structure.

5.7.6.2 acStageSettings acSettings

A global instance of the AC settings structure.

5.7.6.3 Uint16 enableAllFlag

Enable-all condition flag that allows status communication between the state machine tasks.

5.7.6.4 extDeviceSettings extSettings

A global instance of the external device settings structure.

5.7.6.5 loadStageNets loadNets[numberOfLoads]

A global array instance of the load nets structure that collects the nets for all 4 loads.

5.7.6.6 loadStageSettings loadSettings[numberOfLoads]

A global array instance of the load settings structure that collects the settings for all 4 loads.

5.7.6.7 Uint16 stopAllFlag

Stop-all condition flag that allows status communication between the state machine tasks.

5.7.6.8 xfmrStageNets xfmrNets

A global instance of the transformer nets structure.

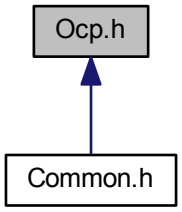
5.7.6.9 xfmrStageSettings xfmrSettings

A global instance of the transformer settings structure.

5.8 Ocp.h File Reference

Over-current protection functions.

This graph shows which files directly or indirectly include this file:



Macros

- #define LOAD1_OCP_TRIP 1
- #define LOAD2_OCP_TRIP 2
- #define LOAD3_OCP_TRIP 4
- #define LOAD4_OCP_TRIP 8
- #define DCMID_OCP_TRIP 16
- #define DCHV_OCP_TRIP 32
- #define AC_OCP_TRIP 64

Functions

- Uint16 setLoadOcpLevel (loadStage load, float32 dcLevel)
- Uint16 getLoadOcpLevel (loadStage load, float32 *dcLevel)
- Uint16 checkLoadOcp (loadStage load)
- Uint16 getLoadOcpState (loadStage load)
- Uint16 clearLoadOcp (loadStage load)
- Uint16 setDcMidOcpLevel (float32 dcLevel)
- Uint16 getDcMidOcpLevel (float32 *dcLevel)
- Uint16 checkDcMidOcp (void)
- Uint16 getDcMidOcpState (void)
- Uint16 clearDcMidOcp (void)
- Uint16 setAcOcpLevel (float32 pkLevel)
- Uint16 getAcOcpLevel (float32 *pkLevel)
- Uint16 tripAcOcp (void)
- Uint16 getAcOcpState (void)
- Uint16 clearAcOcp (void)

5.8.1 Detailed Description

Over-current protection functions. These functions require that the relevant relevant measurement scales be set before use.

5.8.2 Macro Definition Documentation

5.8.2.1 #define AC_OCP_TRIP 64

OCP flag register AC bit.

5.8.2.2 #define DCHV_OCP_TRIP 32

OCP flag register DC HV bit.

5.8.2.3 #define DCMID_OCP_TRIP 16

OCP flag register DC MID bit.

5.8.2.4 #define LOAD1_OCP_TRIP 1

OCP flag register load 1 bit.

5.8.2.5 #define LOAD2_OCP_TRIP 2

OCP flag register load 2 bit.

5.8.2.6 #define LOAD3_OCP_TRIP 4

OCP flag register load 3 bit.

5.8.2.7 #define LOAD4_OCP_TRIP 8

OCP flag register load 4 bit.

5.8.3 Function Documentation

5.8.3.1 Uint16 checkDcMidOcp (void)

Checks the current reading of the DC stage against the DC OCP limit. Raises the DC OCP flag if the reading is above the limit.

Returns

Error status

5.8.3.2 Uint16 checkLoadOcp (loadStage *load*)

Checks the current reading of the specified load against the load OCP limit. Raises the load OCP flag if the reading is above the limit.

Parameters

in	*load*	Specifies the load on which the reading is to be tested.

Returns

Error status

5.8.3.3 Uint16 clearAcOcp (void)

Clears the AC stage OCP state.

Returns

Error status.

5.8.3.4 Uint16 clearDcMidOcp (void)

Clears the DC stage OCP state.

Returns

Error status.

5.8.3.5 Uint16 clearLoadOcp (loadStage *load*)

Clears the OCP state for the specified load.

Parameters

in	*load*	Specifies the load for which the OCP state is to be cleared.

Returns

Error status.

5.8.3.6 Uint16 getAcOcpLevel (float32 * *pkLevel*)

Queries the over current protection level for the AC stage.

Parameters

out	*pkLevel*	Pointer to location at which to place the query result (amps).

Returns

Error status.

5.8.3.7 Uint16 getAcOcpState (void)

Queries the state of the AC OCP flag.

Returns

True if OCP flag has been raised.

5.8.3.8 Uint16 getDcMidOcpLevel (float32 * *dcLevel*)

Queries the over current protection level for the DC stage.

Parameters

out	*dcLevel*	Pointer to location at which to place the query result (amps).

Returns

Error status.

5.8.3.9 Uint16 getDcMidOcpState (void)

Queries the state of the DC OCP flag.

Returns

True if OCP flag has been raised.

5.8.3.10 Uint16 getLoadOcpLevel (loadStage *load*, float32 * *dcLevel*)

Queries the over current protection level for the specified load.

Parameters

in	*load*	Specifies the load on which the setting is to be queried.
out	*dcLevel*	Pointer to location at which to place the query result (amps).

Returns

Error status.

5.8.3.11 Uint16 getLoadOcpState (loadStage *load*)

Queries the state of the OCP flag for the specified load.

Parameters

in	*load*	Specifies the load on which to check the flag.

Returns

True if OCP flag has been raised.

5.8.3.12 Uint16 setAcOcpLevel (float32 *pkLevel*)

Sets the over current protection limit for the AC stage.

Parameters

in	*pkLevel*	Specifies the peak value to be applied (amps).

Returns

Error status.

5.8.3.13 Uint16 setDcMidOcpLevel (float32 *dcLevel*)

Sets the over current protection limit for the DC stage.

Parameters

in	*dcLevel*	Specifies the DC value to be applied (amps).

Returns

Error status.

5.8.3.14 Uint16 setLoadOcpLevel (loadStage *load*, float32 *dcLevel*)

< OCP flag register. Bits are set to indicate an OCP condition has been found. Sets the over current protection level for the specified load.

Parameters

in	*load*	Specifies the load the setting is to be applied to.
in	*dcLevel*	Specifies the DC value to be applied (amps).

Returns

Error status.

5.8.3.15 Uint16 tripAcOcp (void)

Trips the AC OCP. This provides an OCP function for the trip zone ISR to call when the AC comparator is triggered.

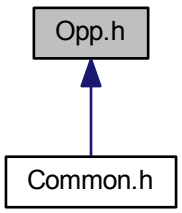
Returns

Error status

5.9 Opp.h File Reference

Over-power protection functions.

This graph shows which files directly or indirectly include this file:



Macros

- #define LOAD1_OPP_TRIP 1
- #define LOAD2_OPP_TRIP 2

- #define LOAD3_OPP_TRIP 4
- #define LOAD4_OPP_TRIP 8
- #define DCMID_OPP_TRIP 16
- #define DCHV_OPP_TRIP 32
- #define AC_OPP_TRIP 64

Functions

- Uint16 checkLoadOpp (loadStage load)
- Uint16 getLoadOppState (loadStage load)
- Uint16 clearLoadOpp (loadStage load)
- Uint16 checkAcOpp (void)
- Uint16 getAcOppState (void)
- Uint16 clearAcOpp (void)

5.9.1 Detailed Description

Over-power protection functions. These functions require that the relevant relevant measurement scales be set before use.

5.9.2 Macro Definition Documentation

5.9.2.1 #define AC_OPP_TRIP 64

OPP flag register AC bit.

5.9.2.2 #define DCHV_OPP_TRIP 32

OPP flag register DC HV bit.

5.9.2.3 #define DCMID_OPP_TRIP 16

OPP flag register DC MID bit.

5.9.2.4 #define LOAD1_OPP_TRIP 1

OPP flag register load 1 bit.

5.9.2.5 #define LOAD2_OPP_TRIP 2

OPP flag register load 2 bit.

5.9.2.6 #define LOAD3_OPP_TRIP 4

OPP flag register load 3 bit.

5.9.2.7 #define LOAD4_OPP_TRIP 8

OPP flag register load 4 bit.

5.9.3 Function Documentation

5.9.3.1 Uint16 checkAcOpp (void)

Checks the current reading of the AC stage against the AC OPP limit. Raises the AC OPP flag if the reading is above the limit.

Returns

Error status

5.9.3.2 Uint16 checkLoadOpp (loadStage *load*)

< OPP flag register. Bits are set to indicate an OPP condition has been found. Checks the power reading of the specified load against the load OPP limit. Raises the load OPP flag if the reading is above the limit.

Parameters

in	*load*	Specifies the load on which the reading is to be tested.

Returns

Error status

5.9.3.3 Uint16 clearAcOpp (void)

Clears the AC stage OPP state.

Returns

Error status.

5.9.3.4 Uint16 clearLoadOpp (loadStage *load*)

Clears the OPP state for the specified load.

Parameters

in	*load*	Specifies the load for which the OPP state is to be cleared.

Returns

Error status.

5.9.3.5 Uint16 getAcOppState (void)

Queries the state of the AC OPP flag.

Returns

True if OPP flag has been raised.

5.9.3.6 Uint16 getLoadOppState (loadStage *load*)

Queries the state of the OPP flag for the specified load.

Parameters

in	*load*	Specifies the load on which to check the flag.

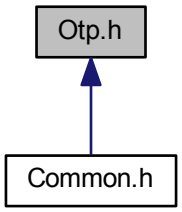
Returns

True if OPP flag has been raised.

5.10 Otp.h File Reference

Over-temperature protection functions.

This graph shows which files directly or indirectly include this file:



Macros

- #define LOAD1_OTP_TRIP 1
- #define LOAD2_OTP_TRIP 2
- #define LOAD3_OTP_TRIP 4
- #define LOAD4_OTP_TRIP 8
- #define DC_OTP_TRIP 16
- #define RSRVD_OTP_TRIP 32
- #define AC_OTP_TRIP 64
- #define EXT1_OTP_TRIP 128
- #define EXT2_OTP_TRIP 256

Functions

- Uint16 setLoadOtpLevel (loadStage load, float32 level)
- Uint16 getLoadOtpLevel (loadStage load, float32 *level)
- Uint16 checkLoadOtp (loadStage load)
- Uint16 getLoadOtpState (loadStage load)
- Uint16 clearLoadOtp (loadStage load)
- Uint16 setDcOtpLevel (float32 level)
- Uint16 getDcOtpLevel (float32 *level)
- Uint16 checkDcOtp (void)
- Uint16 getDcOtpState (void)
- Uint16 clearDcOtp (void)
- Uint16 setAcOtpLevel (float32 level)
- Uint16 getAcOtpLevel (float32 *level)

- Uint16 checkAcOtp (void)
- Uint16 getAcOtpState (void)
- Uint16 clearAcOtp (void)
- Uint16 setExtOtpLevel (extSelect ext, float32 level)
- Uint16 getExtOtpLevel (extSelect ext, float32 *level)
- Uint16 checkExtOtp (extSelect ext)
- Uint16 getExtOtpState (extSelect ext)
- Uint16 clearExtOtp (extSelect ext)

5.10.1 Detailed Description

Over-temperature protection functions.

5.10.2 Macro Definition Documentation

5.10.2.1 #define AC_OTP_TRIP 64

OTP flag register AC bit.

5.10.2.2 #define DC_OTP_TRIP 16

OTP flag register DC bit.

5.10.2.3 #define EXT1_OTP_TRIP 128

OTP flag register external 1 bit.

5.10.2.4 #define EXT2_OTP_TRIP 256

OTP flag register external 2 bit.

5.10.2.5 #define LOAD1_OTP_TRIP 1

OTP flag register load 1 bit.

5.10.2.6 #define LOAD2_OTP_TRIP 2

OTP flag register load 2 bit.

5.10.2.7 #define LOAD3_OTP_TRIP 4

OTP flag register load 3 bit.

5.10.2.8 #define LOAD4_OTP_TRIP 8

OTP flag register load 4 bit.

5.10.2.9 #define RSRVD_OTP_TRIP 32

Reserved bit.

5.10.3 Function Documentation

5.10.3.1 Uint16 checkAcOtp (void)

Checks the temperature reading of the AC stage against the AC OTP limit. Raises the AC OTP flag if the reading is above the limit.

Returns

Error status

5.10.3.2 Uint16 checkDcOtp (void)

Checks the temperature reading of the DC stage against the DC OTP limit. Raises the DC OTP flag if the reading is above the limit.

Returns

Error status

5.10.3.3 Uint16 checkExtOtp (extSelect *ext*)

Checks the temperature reading of the specified external sensor against the sensor OTP limit. Raises the load OTP flag if the reading is above the limit.

Parameters

in	*ext*	Specifies the sensor on which the setting is to be queried.

Returns

Error status

5.10.3.4 Uint16 checkLoadOtp (loadStage *load*)

Checks the temperature reading of the specified load against the load OTP limit. Raises the load OTP flag if the reading is above the limit.

Parameters

in	*load*	Specifies the load on which the reading is to be tested.

Returns

Error status

5.10.3.5 Uint16 clearAcOtp (void)

Clears the AC stage OTP state.

Returns

Error status.

5.10.3.6 Uint16 clearDcOtp (void)

Clears the DC stage OTP state.

Returns

Error status.

5.10.3.7 Uint16 clearExtOtp (extSelect *ext*)

Clears the OTP state for the specified external sensor.

Parameters

in	*ext*	Specifies the external sensor for which the OTP state is to be cleared.

Returns

Error status.

5.10.3.8 Uint16 clearLoadOtp (loadStage *load*)

Clears the OTP state for the specified load.

Parameters

in	*load*	Specifies the load for which the OTP state is to be cleared.

Returns

Error status.

5.10.3.9 Uint16 getAcOtpLevel (float32 * *level*)

Queries the over temperature protection level for the AC stage.

Parameters

out	*level*	Pointer to location at which to place the query result (° C).

Returns

Error status.

5.10.3.10 Uint16 getAcOtpState (void)

Queries the state of the AC OTP flag.

Returns

True if OTP flag has been raised.

5.10.3.11 Uint16 getDcOtpLevel (float32 * *level*)

Queries the over current protection level for the DC stage.

Parameters

out	*level*	Pointer to location at which to place the query result (° C).

Returns

Error status.

5.10.3.12 Uint16 getDcOtpState (void)

Queries the state of the DC OTP flag.

Returns

True if OTP flag has been raised.

5.10.3.13 Uint16 getExtOtpLevel (extSelect *ext,* float32 ∗ *level*)

Queries the over temperature protection level for the specified external sensor.

Parameters

in	*ext*	Specifies the external sensor on which the setting is to be queried.
out	*level*	Pointer to location at which to place the query result (° C).

Returns

Error status.

5.10.3.14 Uint16 getExtOtpState (extSelect *ext*)

Queries the state of the OTP flag for the specified external sensor.

Parameters

in	*ext*	Specifies the sensor on which to check the flag.

Returns

True if OTP flag has been raised.

5.10.3.15 Uint16 getLoadOtpLevel (loadStage *load,* float32 ∗ *level*)

Queries the over temperature protection level for the specified load.

Parameters

in	*load*	Specifies the load on which the setting is to be queried.
out	*level*	Pointer to location at which to place the query result (° C).

Returns

Error status.

5.10.3.16 Uint16 getLoadOtpState (loadStage *load*)

Queries the state of the OTP flag for the specified load.

Parameters

in	*load*	Specifies the load on which to check the flag.

Returns

True if OTP flag has been raised.

5.10.3.17 Uint16 setAcOtpLevel (float32 *level*)

Sets the over current protection limit for the AC stage.

Parameters

in	*level*	Specifies the value to be applied (° C).

Returns

Error status.

5.10.3.18 Uint16 setDcOtpLevel (float32 *level*)

Sets the over temperature protection limit for the DC stage.

Parameters

in	*level*	Specifies the value to be applied (° C).

Returns

Error status.

5.10.3.19 Uint16 setExtOtpLevel (extSelect *ext,* float32 *level*)

Sets the over temperature protection level for the specified external sensor.

Parameters

in	*ext*	Specifies the sensor the setting is to be applied to.
in	*level*	Specifies the value to be applied (° C).

Returns

Error status.

5.10.3.20 Uint16 setLoadOtpLevel (loadStage *load,* float32 *level*)

< OTP flag register. Bits are set to indicate an OTP condition has been found. Sets the over temperature protection level for the specified load.

Parameters

in	*load*	Specifies the load the setting is to be applied to.
in	*level*	Specifies the value to be applied (° C).

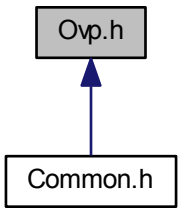
Returns

Error status.

5.11 Ovp.h File Reference

Over-voltage protection functions.

This graph shows which files directly or indirectly include this file:



Macros

- #define LOAD1_OVP_TRIP 1
- #define LOAD2_OVP_TRIP 2
- #define LOAD3_OVP_TRIP 4
- #define LOAD4_OVP_TRIP 8
- #define DCMID_OVP_TRIP 16
- #define DCHV_OVP_TRIP 32
- #define AC_OVP_TRIP 64

Functions

- Uint16 checkLoadOvp (loadStage load)
- Uint16 getLoadOvpState (loadStage load)
- Uint16 clearLoadOvp (loadStage load)
- Uint16 tripDcMidOvp (void)
- Uint16 getDcMidOvpState (void)
- Uint16 clearDcMidOvp (void)
- Uint16 setDcHvOvpLevel (float32 dcLevel)
- Uint16 getDcHvOvpLevel (float32 *dcLevel)
- Uint16 checkDcHvOvp (void)
- Uint16 getDcHvOvpState (void)
- Uint16 clearDcHvOvp (void)
- Uint16 setAcOvpLevel (float32 pkLevel)
- Uint16 getAcOvpLevel (float32 *pkLevel)
- Uint16 checkAcOvp (void)
- Uint16 getAcOvpState (void)
- Uint16 clearAcOvp (void)

5.11.1 Detailed Description

Over-voltage protection functions. These functions require that the relevant relevant measurement scales be set before use.

5.11.2 Macro Definition Documentation

5.11.2.1 #define AC_OVP_TRIP 64

OVP flag register AC bit.

5.11.2.2 #define DCHV_OVP_TRIP 32

OVP flag register DC HV bit.

5.11.2.3 #define DCMID_OVP_TRIP 16

OVP flag register DC MID bit.

5.11.2.4 #define LOAD1_OVP_TRIP 1

OVP flag register load 1 bit.

5.11.2.5 #define LOAD2_OVP_TRIP 2

OVP flag register load 2 bit.

5.11.2.6 #define LOAD3_OVP_TRIP 4

OVP flag register load 3 bit.

5.11.2.7 #define LOAD4_OVP_TRIP 8

OVP flag register load 4 bit.

5.11.3 Function Documentation

5.11.3.1 Uint16 checkAcOvp (void)

Checks the voltage reading of the AC stage against the AC OVP limit. Raises the AC OVP flag if the reading is above the limit.

Returns

Error status

5.11.3.2 Uint16 checkDcHvOvp (void)

Checks the voltage reading of the DC HV stage against the DC HV OVP limit. Raises the DC HV OVP flag if the reading is above the limit.

Returns

Error status

5.11.3.3 Uint16 checkLoadOvp (loadStage *load*)

< OVP flag register. Bits are set to indicate an OVP condition has been found. Checks the voltage reading of the specified load against the load OVP limit. Raises the load OVP flag if the reading is above the limit.

Parameters

in	*load*	Specifies the load on which the reading is to be tested.

Returns

Error status

5.11.3.4 Uint16 clearAcOvp (void)

Clears the AC stage OVP state.

Returns

Error status.

5.11.3.5 Uint16 clearDcHvOvp (void)

Clears the DC HV stage OVP state.

Returns

Error status.

5.11.3.6 Uint16 clearDcMidOvp (void)

Clears the DC Mid stage OVP state.

Returns

Error status.

5.11.3.7 Uint16 clearLoadOvp (loadStage *load*)

Clears the OVP state for the specified load.

Parameters

in	*load*	Specifies the load for which the OVP state is to be cleared.

Returns

Error status.

5.11.3.8 Uint16 getAcOvpLevel (float32 * *pkLevel*)

Queries the over voltage protection level for the AC stage.

Parameters

out	*pkLevel*	Pointer to location at which to place the query result (volts).

Returns

Error status.

5.11.3.9 Uint16 getAcOvpState (void)

Queries the state of the AC OVP flag.

Returns

True if OVP flag has been raised.

5.11.3.10 Uint16 getDcHvOvpLevel (float32 * *dcLevel*)

Queries the over voltage protection level for the DC HV stage.

Parameters

out	*dcLevel*	Pointer to location at which to place the query result (volts).

Returns

Error status.

5.11.3.11 Uint16 getDcHvOvpState (void)

Queries the state of the DC HV OVP flag.

Returns

True if OVP flag has been raised.

5.11.3.12 Uint16 getDcMidOvpState (void)

Queries the state of the DC Mid OVP flag.

Returns

True if OVP flag has been raised.

5.11.3.13 Uint16 getLoadOvpState (loadStage *load*)

Queries the state of the OVP flag for the specified load.

Parameters

in	*load*	Specifies the load on which to check the flag.

Returns

True if OVP flag has been raised.

5.11.3.14 Uint16 setAcOvpLevel (float32 *pkLevel*)

Sets the over voltage protection limit for the AC stage.

Parameters

in	*pkLevel*	Specifies the peak value to be applied (volts).

Returns

Error status.

5.11.3.15 Uint16 setDcHvOvpLevel (float32 *dcLevel*)

Sets the over voltage protection limit for the DC HV stage.

Parameters

in	*dcLevel*	Specifies the DC value to be applied (volts).

Returns

Error status.

5.11.3.16 Uint16 tripDcMidOvp (void)

Trips the DC Mid OVP. This provides an OVP function for the trip zone ISR to call when the DC comparator is triggered.

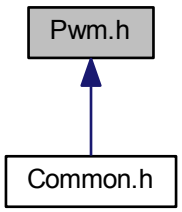
Returns

Error status

5.12 Pwm.h File Reference

PWM and related functions.

This graph shows which files directly or indirectly include this file:



Macros

- #define PWM_1_PRD 462
- #define PWM_2_PRD 462
- #define PWM_3_PRD 462

Functions

- void initPwm (void)
- interrupt void DPL_ISR (void)

Variables

- volatile int32 * PWMDRV_2ch_UpCnt_Duty1A
- volatile int32 * PWMDRV_2ch_UpCnt_Duty1B
- volatile int32 * PWMDRV_2ch_UpCnt_Duty2A
- volatile int32 * PWMDRV_2ch_UpCnt_Duty2B
- volatile int32 * PWMDRV_2ch_UpCnt_Duty3A
- volatile int32 * PWMDRV_2ch_UpCnt_Duty3B

5.12.1 Detailed Description

PWM and related functions.

5.12.2 Macro Definition Documentation

5.12.2.1 #define PWM_1_PRD 462

Defines the period setting for PWM1. 60MHz / 462 = 130kHz.

5.12.2.2 #define PWM_2_PRD 462

Defines the period setting for PWM1. 60MHz / 462 = 130kHz.

5.12.2.3 #define PWM_3_PRD 462

Defines the period setting for PWM1. 60MHz / 462 $\sim$ 130kHz.

5.12.3 Function Documentation

5.12.3.1 interrupt void DPL_ISR (void)

Digital power control loop interrupt service routine Located in the assembly file BurnInUnit_ISR.asm

5.12.3.2 void initPwm (void)

Initialises PWM macros, SOCs and ISR (DPL trigger).

5.12.4 Variable Documentation

5.12.4.1 volatile int32* PWMDRV_2ch_UpCnt_Duty1A

Channel 0 PWM terminal pointer.

5.12.4.2 volatile int32* PWMDRV_2ch_UpCnt_Duty1B

Channel 1 PWM terminal pointers.

5.12.4.3 volatile int32* PWMDRV_2ch_UpCnt_Duty2A

Channel 2 PWM terminal pointer.

5.12.4.4 volatile int32* PWMDRV_2ch_UpCnt_Duty2B

Channel 3 PWM terminal pointer.

5.12.4.5 volatile int32* PWMDRV_2ch_UpCnt_Duty3A

Interboost PWM terminal pointer.

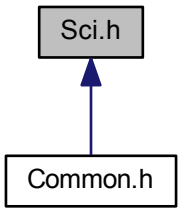
5.12.4.6 volatile int32* PWMDRV_2ch_UpCnt_Duty3B

AC stage PWM terminal pointer.

5.13 Sci.h File Reference

Serial communications interface functions.

This graph shows which files directly or indirectly include this file:



Macros

- #define SCIBAUD_MIN 29
- #define SCIFFRX_INT_LVL 1
- #define SCIFFTX_INT_LVL 0
- #define SCIFFTX_FILL_LVL 4

Functions

- Uint16 sciInit (Uint32 baud)
- void sciTx (void)

5.13.1 Detailed Description

Serial communications interface functions.

On the TI C2000 LaunchPad XL:

Function	GPIO	PCB Pin
SCI_RX	GPIO28	J1-3
SCI_TX	GPIO29	J1-4

Warning

On the TI C2000 LaunchPad XL the switch 4 (S4) should be in the OFF position to communicate with devices other than the on-board USB controller.

5.13.2 Macro Definition Documentation

5.13.2.1 #define SCIBAUD_MIN 29

Minimum allowable value of SCI Baud.

5.13.2.2 #define SCIFFRX_INT_LVL 1

Interrupt level for receiving FIFO.

5.13.2.3 #define SCIFFTX_FILL_LVL 4

Fill level for transmission FIFO.

5.13.2.4 #define SCIFFTX_INT_LVL 0

Interrupt level for transmission FIFO.

5.13.3 Function Documentation

5.13.3.1 Uint16 sciInit (Uint32 *baud*)

Initialises the SCI(A) peripheral and relevant interrupts.

Parameters

in	*baud*	The baud rate that the SCI should use minimum value is set by SCIBAUD_MIN.

Returns

Error status.

Warning

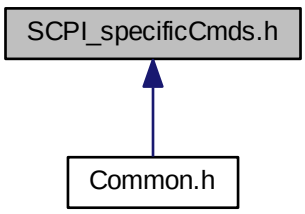
This function will clear any values already in the SCI peripheral registers.
This function MUST be called before any other SCI function.

5.13.3.2 void sciTx (void)

Transmits whatever data is on the SCPI output queue if SCI has been selected as the external communications type.

5.14 SCPI_specificCmds.h File Reference

This graph shows which files directly or indirectly include this file:



5.14.1 Detailed Description

This file includes the functions for registering the application specific commands and the related callback functions

These commands and their associated callback functions can be edited to suit the application in use, though they should follow the guidelines and conventions laid out in IEEE 488.2-1992 and SCPI-1999

Please follow the information in the parser documentation and the file SCPI_specificCmds.c on how to add or change message headers, commands and parameters

To register a new command device specific tree node, add a registerChild() call to the function registerSpecificCommands() with the form "err += registerChild(CHILD_LONG_NAME, PARENT_SHORT_NAME, BOOL_IS_HEADER_ONLY, BOOL_IS_QUERY, CALLBACK_HANDLE);", where:

- CHILD_LONG_NAME is the long form of the name of the child node to be registered.
- PARENT_SHORT_NAME is the short form of the name of the parent of the child node to be registered. This must be an already existing node.
- BOOL_IS_HEADER_ONLY is a boolean value that indicates if the node being registered is a header only (i.e. is not a command) or not.
- BOOL_IS_QUERY is a boolean value that indicates if the node being registered my be queried or not.
- CALLBACK_HANDLE is a pointer to the function to be associated with this child if it is a command, or NULL if it is a header only.

For further information on the operation of registerChild() please see the function description.

All string literals MUST be in upper case and enclosed in quotation marks.

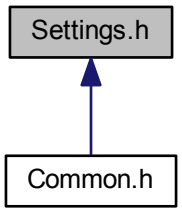
Node long and short names must conform to the standard as outlined by SCPI-99 in 6.2.1

All tree nodes are children of at least "ROOT". Changes to the number of node registered should be reflected by the user in the number defined for TREE_CHILD_LIMIT in scpi.h

5.15 Settings.h File Reference

Major build definitions and settings for the project.

This graph shows which files directly or indirectly include this file:



Macros

- #define INCR_BUILD 2
- #define DEBUG

- #define LOG_SIN
- #define DUAL_CNTL_AC
- #define VSSA 0l
- #define VDDA 3300l
- #define LOAD_I_SCALE 0.25f
- #define LOAD_V_R1 18.0f
- #define LOAD_V_R2 1.0f
- #define HV_V_R1 540.0f
- #define HV_V_R2 4.3f
- #define MID_I_SCALE 0.25f
- #define MID_V_R1 18.0f
- #define MID_V_R2 1.0f
- #define AC_I_SCALE 0.25f
- #define AC_V_R1 540.0f
- #define AC_V_R2 4.3f
- #define LOAD_IDCLVL_MAX 35
- #define LOAD_VDCLVL_FIX 60
- #define LOAD_PWRLVL_FIX 200
- #define DCMID_VDCLVL_FIX 400
- #define DCMIDV_STRTUP_LMT 10.0f
- #define DCHV_UVLCK_LMT 330.0f
- #define AC_IRMSLVL_MAX 10
- #define AC_VRMSLVL_MAX 250
- #define AC_PWRLVL_FIX 800
- #define XFMR_PWMF_FIX 130
- #define SQRT_2 1.41429
- #define RECP_SQRT_2 0.70711

- #define CHANNEL_OOB 0x10
- #define VALUE_OOB 0x11
- #define OCP_TRIP 0x12
- #define OVP_TRIP 0x13
- #define OPP_TRIP 0x14
- #define OTP_TRIP 0x15
- #define UVLKT_TRIP 0x16
- #define TRIP_ALERT 0x17
- #define I2C_READ_WRONG_MSG 0x20
- #define I2C_WRITE_WRONG_MSG 0x21
- #define I2C_STP_NOT_READY 0x22
- #define I2C_BUS_BUSY 0x23
- #define I2C_INVALID_ISRC 0x24

5.15.1 Detailed Description

Major build definitions and settings for the project.

Warning

This file is included and referenced by ISR.asm, main() and mnConnectNets().
When changes are made to this file please use rebuild all.

5.15.2 Macro Definition Documentation

5.15.2.1 #define AC_I_SCALE 0.25f

Scaling factor for AC current ADC. Volts/amp.

5.15.2.2 #define AC_IRMSLVL_MAX 10

The maximum allowable value, in amps (RMS), for the AC current level.

5.15.2.3 #define AC_PWRLVL_FIX 800

The fixed value, in watts, for the AC OPP level.

5.15.2.4 #define AC_V_R1 540.0f

Scaling voltage divider R1 resistor value for AC voltage ADC. kOhms.

5.15.2.5 #define AC_V_R2 4.3f

Scaling voltage divider R2 resistor value for AC voltage ADC. kOhms.

5.15.2.6 #define AC_VRMSLVL_MAX 250

The maximum allowable value, in volts (RMS), for the AC voltage level.

5.15.2.7 #define CHANNEL_OOB 0x10

Channel out of bounds error code.

5.15.2.8 #define DCHV_UVLCK_LMT 330.0f

The DC HV under-voltage lockout limit level.

5.15.2.9 #define DCMID_VDCLVL_FIX 400

The fixed maximum value, in volts, for the DC mid voltage level.

5.15.2.10 #define DCMIDV_STRTUP_LMT 10.0f

The maximum limit for DC MID voltage at to allow transformer stage startup.

5.15.2.11 #define DEBUG

Includes and makes functions and variables public that are used only for debugging purposes.

5.15.2.12 #define DUAL_CNTL_AC

Uses the dual CNTL AC control instead of single VCtrl. Cannot be used if PID is still in use.

5.15.2.13 #define HV_V_R1 540.0f

Scaling voltage divider R1 resistor value for HV voltage ADC. kOhms.

5.15.2.14 #define HV_V_R2 4.3f

Scaling voltage divider R2 resistor value for HV voltage ADC. kOhms.

5.15.2.15 #define I2C_BUS_BUSY 0x23

I2C bus already busy error code.

5.15.2.16 #define I2C_INVALID_ISRC 0x24

Invalid I2C interrupt source error code.

5.15.2.17 #define I2C_READ_WRONG_MSG 0x20

Incorrect type I2C message read error code.

5.15.2.18 #define I2C_STP_NOT_READY 0x22

I2C stop bit was not yet received error code.

5.15.2.19 #define I2C_WRITE_WRONG_MSG 0x21

Incorrect type I2C write message error code.

5.15.2.20 #define INCR_BUILD 2

Alters the digital power control loop between closed or open. Open-Loop: 1. Closed-loop: 2.

5.15.2.21 #define LOAD_I_SCALE 0.25f

Scaling factor for load current ADC.

5.15.2.22 #define LOAD_IDCLVL_MAX 35

The maximum allowable value, in amps, for the load current levels.

5.15.2.23 #define LOAD_PWRLVL_FIX 200

The fixed maximum value, in watts, for the load power levels.

5.15.2.24 #define LOAD_V_R1 18.0f

Scaling voltage divider R1 resistor value for load voltage ADC. kOhms.

5.15.2.25 #define LOAD_V_R2 1.0f

Scaling voltage divider R2 resistor value for load voltage ADC. kOhms.

5.15.2.26 #define LOAD_VDCLVL_FIX 60

The fixed maximum value, in volts, for the load voltage levels.

5.15.2.27 #define LOG_SIN

Logs the sine wave VOut and SignOut data into data-log buffers for debug graphing.

5.15.2.28 #define MID_I_SCALE 0.25f

Scaling factor for MID current ADC. Volts/amp.

5.15.2.29 #define MID_V_R1 18.0f

Scaling voltage divider R1 resistor value for MID voltage ADC. kOhms.

5.15.2.30 #define MID_V_R2 1.0f

Scaling voltage divider R2 resistor value for MID voltage ADC. kOhms.

5.15.2.31 #define OCP_TRIP 0x12

Over-current protection trip error code.

5.15.2.32 #define OPP_TRIP 0x14

Over-power protection trip error code.

5.15.2.33 #define OTP_TRIP 0x15

Over-temperature protection trip error code.

5.15.2.34 #define OVP_TRIP 0x13

Over-voltage protection trip error code.

5.15.2.35 #define RECP_SQRT_2 0.70711

1/sqrt(2) constant used for RMS calculations.

5.15.2.36 #define SQRT_2 1.41429

Sqrt(2) constant used for RMS calculations.

5.15.2.37 #define TRIP_ALERT 0x17

General purpose alert trip error code.

5.15.2.38 #define UVLKT_TRIP 0x16

Under-voltage lock-out trip error code.

5.15.2.39 #define VALUE_OOB 0x11

Value out of bounds error code.

5.15.2.40 #define VDDA 3300l

System VMAXREF (millivolts).

5.15.2.41 #define VSSA 0l

System VLOWREF (millivolts).

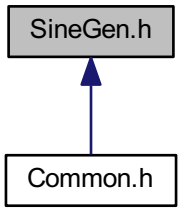
5.15.2.42 #define XFMR_PWMF_FIX 130

The fixed frequency (kHz) setting for the transformer PWM (at 50% duty).

5.16 SineGen.h File Reference

Signal generator functions.

This graph shows which files directly or indirectly include this file:



Macros

- #define SIN_F_ISR 60000000.0f/PWM_1_PRD
- #define SIN_F_ISR_DVSR 6
- #define SIN_F_MAX 200u
- #define SIN_F_REQ 50.0
- #define SIN_OFST 0

- #define SIN_PHSE 0
- #define SIN_DFLT_GAIN 0.0
- #define SIN_DFLT_RCTFY TRUE
- #define SIN_CHANNEL AC_STAGE

Functions

- void initSine (Uint16 enablePhaseOut)
- void updateSineGain (void)
- void updateSineSignal (void)
- void enableSinePhaseOut (void)
- void disableSinePhaseOut (void)
- Uint16 setSineState (Uint16 state)
- Uint16 getSineState (Uint16 *state)
- Uint16 setSineRmsTarget (float32 rmsTarget)
- Uint16 setSineGainTarget (float32 target)
- Uint16 getSineGainTarget (float32 *target)
- Uint16 setSineGainStep (float32 step)
- Uint16 getSineGainStep (float32 *step)

Variables

- volatile int32 * SGENTI_1ch_VOut

5.16.1 Detailed Description

Signal generator functions. sgInit() must be called before any other signal generator functions are used. Note that the frequency resolution is determined by the maximum frequency and the step max. For further details, see the signal generator library documentation (Texas Instruments Signal Generator Library Module user's Guide).

Warning

This file is included by the file ISR.asm.

5.16.2 Macro Definition Documentation

5.16.2.1 #define SIN_CHANNEL AC_STAGE

Defines which channel enable controls the generator output.

5.16.2.2 #define SIN_DFLT_GAIN 0.0

Initial gain setting [0.0, 1.0].

5.16.2.3 #define SIN_DFLT_RCTFY TRUE

Initial rectification setting [TRUE | FALSE).

5.16.2.4 #define SIN_F_ISR 60000000.0f/PWM_1_PRD

Sampling frequency. I.e. the frequency at which the DPL ISR task runs.

5.16.2.5 #define SIN_F_ISR_DVSR 6

Sampling frequency divisor. E.g. F_ISR = 130kHz, but the DPL ISR only runs on every third trigger plus there are two alternating tasks so, 3 x 2 = 6

5.16.2.6 #define SIN_F_MAX 200u

Maximum frequency setting (hertz).

5.16.2.7 #define SIN_F_REQ 50.0

Required frequency setting (hertz).

5.16.2.8 #define SIN_OFST 0

Offset setting [-0.5, +0.5], IQ15.

5.16.2.9 #define SIN_PHSE 0

Initial phase setting [0, 360), IQ16.

5.16.3 Function Documentation

5.16.3.1 void disableSinePhaseOut (void)

Disables the use of the AC phase output signal on GPIO12.

5.16.3.2 void enableSinePhaseOut (void)

Enables the use of the AC phase output signal on GPIO12.

5.16.3.3 Uint16 getSineGainStep (float32 * *step*)

Queries the current gain step setting.

Parameters

out	*step*	Pointer to location at which to place the query result.

Returns

Error status.

5.16.3.4 Uint16 getSineGainTarget (float32 * *target*)

Queries the current target gain setting.

Parameters

out	*target*	Pointer to location at which to place the query result.

Returns

Error status.

5.16.3.5 Uint16 getSineState (Uint16 * *state*)

Queries the current state of the generator output.

Parameters

out	*state*	Pointer to location at which to place the query result {1:ON | 0:OFF}.

Returns

Error status.

5.16.3.6 void initSine (Uint16 *enablePhaseOut*)

Sets the initial generator values and disables the output. This function MUST be called before any other signal generator function.

Parameters

in	*enablePhaseOut*	Determines if the AC phase signal is enabled as an output or disabled and switched to an input.

5.16.3.7 Uint16 setSineGainStep (float32 *step*)

Sets the gain step of the sine signal.

Parameters

in	*step*	Specifies the value of the gain step to be applied.

Returns

Error status.

5.16.3.8 Uint16 setSineGainTarget (float32 *target*)

Sets the target gain of the signal.

Parameters

in	*target*	Gain target value [0.0, 1.0).

Returns

Error status.

5.16.3.9 Uint16 setSineRmsTarget (float32 *rmsTarget*)

Sets the target RMS voltage of the signal.

Parameters

in	*rmsTarget*	RMS voltage target value.

Returns

Error status.

5.16.3.10 Uint16 setSineState (Uint16 *state*)

Enables or disables the output of the generator onto the connected net

Parameters

in	*state*	Output enable state {1:ON | 0:OFF}.

Returns

Error status.

5.16.3.11 void updateSineGain (void)

Updates the gain value to create a slow-start ramp. This should be called at the same time and similarly to the DC slew update.

5.16.3.12 void updateSineSignal (void)

Generates the next signal data point and loads it onto the VOut terminal. If the point is positive the sign terminal is set, otherwise it is cleared. If rectify is enabled, the VOut value produced will be an absolute value. This is called by the DP_LIB asm ISR.

5.16.4 Variable Documentation

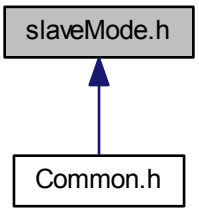
5.16.4.1 volatile int32* SGENTI_1ch_VOut

Voltage output terminal.

5.17 slaveMode.h File Reference

Functions and variables used in detection of the mode of the unit.

This graph shows which files directly or indirectly include this file:



Typedefs

- typedef enum slaveMode slaveMode

Enumerations

- enum slaveMode

Functions

- slaveMode detectSlaveMode (void)

Variables

- slaveMode slaveModeStatus

5.17.1 Detailed Description

Functions and variables used in detection of the mode of the unit. These functions determine the mode of the burn in unit and should be run after system start up. The detection is managed using GPIO6 and GPIO7. A unit may be in one of four modes: a single unit, a master unit, a slave unit or alternatively a unit may be in the undetected mode which is the initial mode of a unit upon startup.

Warning

> For the mode to be correctly detected the two units should be connected while powered off, then the slave unit should be powered on first, followed by powering on the master unit second. Once detected at startup, any further changes to the connected mode will not be detected and thus the detected mode reported will not reflect such changes to the connection mode of the unit. To change the connection mode of a unit, it, and any slave or master unit must be powered off again.

Also note that, as mentioned, the mode detection algorithm uses GPIO6. However the during initialisation and use of the SPI peripheral GPIO6 is used as a slave service request line. Thus even if the unit mode is not used in conjunction with the SPI functions they should still only be used after the slave mode has been detected.

5.17.2 Typedef Documentation

5.17.2.1 typedef enum slaveMode slaveMode

A type to allow specification of the unit mode

5.17.3 Enumeration Type Documentation

5.17.3.1 enum slaveMode

The possible unit modes

5.17.4 Function Documentation

5.17.4.1 slaveMode detectSlaveMode (void)

Detects the unit mode. This should only be run once, at startup, to allow the mode to be provided to any other function initialisations that may rely on it.

When a slave unit is powered on the state of GPIO7 is read. If the reading is LOW, this indicates that the unit is a slave and subsequently GPIO6 is set HIGH. If the GPIO7 reading indicated that the unit is a master unit the state of GPIO6 is then read and if it is LOW it determines that the unit is a single unit.

Returns

The detected slave mode. If the mode has already been detected the state will be available by referencing the slaveModeStatus variable instead.

5.17.5 Variable Documentation

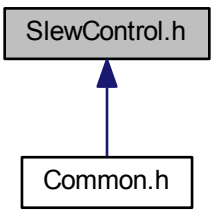
5.17.5.1 slaveMode slaveModeStatus

A global variable that holds the detected mode of the unit, allowing the mode to be queried.

5.18 SlewControl.h File Reference

Slew control functions.

This graph shows which files directly or indirectly include this file:



Functions

- void updateLoadSlew (void)
- Uint16 setLoadSlewTarget (loadStage load, float32 target)
- Uint16 setLoadSlewStep (loadStage load, float32 step)
- Uint16 setLoadState (loadStage load, Uint16 state)
- Uint16 scSetLoadStateAll (Uint16 state)
- Uint16 getLoadSlewTarget (loadStage load, float32 *target)
- Uint16 getLoadSlewStep (loadStage load, float32 *step)
- Uint16 getLoadState (loadStage load, Uint16 *state)

5.18.1 Detailed Description

Slew control functions.

5.18.2 Function Documentation

5.18.2.1 Uint16 getLoadSlewStep (loadStage *load*, float32 * *step*)

Queries the current slew step size of the specified channel.

Parameters

in	*load*	Specifies the load the setting is to be read from.
out	*step*	Pointer to location at which to place the query result (amps).

Returns

Error status.

5.18.2.2 Uint16 getLoadSlewTarget (loadStage *load*, float32 * *target*)

Queries the current slew target setting for the specified channel.

Parameters

in	*load*	Specifies the load the setting is to be read from.
out	*target*	Pointer to location at which to place the query result (amps).

Returns

Error status.

5.18.2.3 Uint16 getLoadState (loadStage *load*, Uint16 * *state*)

Queries the current reference net enable state for the specified channel.

Parameters

in	*load*	Specifies the load the setting is to be read from.
out	*state*	Pointer to location at which to place the query result (0:OFF | non-zero:ON).

Returns

Error status.

5.18.2.4 Uint16 scSetLoadStateAll (Uint16 *state*)

Sets all channels' reference net enable state.

Parameters

in	*state*	Specifies the refernce net state to be applied (0:OFF | non-zero:ON).

Returns

Error status.

5.18.2.5 Uint16 setLoadSlewStep (loadStage *load*, float32 *step*)

Sets the slew step size for the specified load.

Parameters

in	*load*	Specifies the load the setting is to be applied to.
in	*step*	Specifies the value of the slew step size to be applied (amps).

Returns

Error status.

5.18.2.6 Uint16 setLoadSlewTarget (loadStage *load*, float32 *target*)

Sets the slew target for the specified load.

Parameters

in	*load*	Specifies the load the setting is to be applied to.
in	*target*	Specifies the value of the slew target to be applied (amps).

Returns

Error status.

5.18.2.7 Uint16 setLoadState (loadStage *load*, Uint16 *state*)

Sets the reference net enable state for the specified load.

Parameters

in	*load*	Specifies the load the setting is to be applied to.
in	*state*	Specifies the reference net state to be applied (0:OFF | non-zero:ON).

Returns

Error Status.

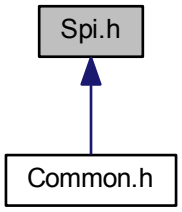
5.18.2.8 void updateLoadSlew (void)

Advances the slew ramps for all loads.

5.19 Spi.h File Reference

Serial peripheral interface communications functions.

This graph shows which files directly or indirectly include this file:



Macros

- #define SPI_DFLT_BAUD 0x3D090
- #define SPI_DFLT_CPOL 0x00
- #define SPI_DFLT_CPHA 0x01
- #define SPI_FFRX_INTLVL 0x01
- #define SPI_FFTX_INTLVL 0x00
- #define SPI_FFTX_FILLVL 0x04

Typedefs

- typedef enum spiMode spiMode
- typedef enum spiLpbk spiLpbk
- typedef enum transPol transPol
- typedef enum spiCPha spiCPha

Enumerations

- enum spiMode
- enum spiLpbk
- enum transPol
- enum spiCPha

Functions

- Uint16 spiInit (spiMode mode, Uint32 baud, spiLpbk loopback, transPol cPol, spiCPha cPha)
- void spiTx (void)

5.19.1 Detailed Description

Serial peripheral interface communications functions. As a master GPIO6 is setup as external interrupt 2 for the slave service request line, set as active low. As a master GPIO6 is setup as an active low output.

Warning

The following bridge tracks should be cut on the TI C2000 LaunchPad XL PCB before SPI (or I2C) may be used:

Bridge to Cut	GPIO	PCB Pin
JP5	GPIO32	J2-6
JP7	GPIO33	J2-7
JP8	GPIO16	J6-7
JP10	GPIO17	J6-8

This should result in the following functionality:

Function	GPIO	PCB Pin
I2C_SDAA	GPIO32	J6-7
I2C_SCLA	GPIO33	J6-8
SPI_MOSI	GPIO16	J2-6
SPI_MISO	GPIO17	J2-7
SPI_CLK	GPIO18	J1-18
SPI_STE	GPIO19	J2-1

5.19.2 Macro Definition Documentation

5.19.2.1 #define SPI_DFLT_BAUD 0x3D090

SPI default baud rate.

5.19.2.2 #define SPI_DFLT_CPHA 0x01

SPI default clock phase. 0 = Without delay, 1 = with delay.

5.19.2.3 #define SPI_DFLT_CPOL 0x00

SPI default clock polarity. 0 = Rising edge, 1 = falling edge.

5.19.2.4 #define SPI_FFRX_INTLVL 0x01

SPI Rx FIFO interrupt level.

5.19.2.5 #define SPI_FFTX_FILLVL 0x04

SPI Tx FIFO fill level.

5.19.2.6 #define SPI_FFTX_INTLVL 0x00

SPI Tx FIFO interrupt level.

5.19.3 Typedef Documentation

5.19.3.1 typedef enum spiCPha spiCPha

A type to allow specification of the SPI clock phase setting.

5.19.3.2 typedef enum spiLpbk spiLpbk

A type to allows specification of the SPI loop-back setting.

5.19.3.3 typedef enum spiMode spiMode

A type to allow specification of the SPI mode.

5.19.3.4 typedef enum transPol transPol

A type to allow specification of the edge transition polarity setting.

5.19.4 Enumeration Type Documentation

5.19.4.1 enum spiCPha

The possible SPI clock phase settings.

5.19.4.2 enum spiLpbk

The possible SPI loop-back settings.

5.19.4.3 enum spiMode

The possible SPI Modes.

5.19.4.4 enum transPol

The possible edge transition polarity settings.

5.19.5 Function Documentation

5.19.5.1 Uint16 spiInit (spiMode *mode,* Uint32 *baud,* spiLpbk *loopback,* transPol *cPol,* spiCPha *cPha*)

Initialises the SPI-A peripheral and relevant interrupts

Parameters

in	*mode*	Specifies if the device is in slave or master mode, values are master or slave.
in	*baud*	Specifies the baud rate to be used minimum value is 117,187, maximum is 3,750,000
in	*loopback*	Specifies if the SPI module should be run in loop-back mode, values are enable or disable.
in	*cPol*	Specifies the clock polarity, values are risingEdge or fallingEdge.
in	*cPha*	Specifies the clock phase, values are withDelay or withoutDelay.

Returns

Error status.

Warning

This function will clear any values already in the SPI peripheral registers.
This function MUST be called before any other public SPI function.

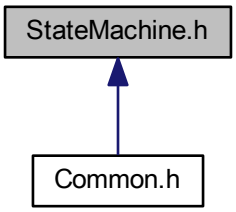
5.19.5.2 void spiTx (void)

Transmits whatever data is on the SCPI output queue if SPI has been selected as the external communications type.

5.20 StateMachine.h File Reference

State machine functions.

This graph shows which files directly or indirectly include this file:



Functions

- void initStateMachine (void)

Variables

- void(∗ Alpha_State_Ptr)(void)

5.20.1 Detailed Description

State machine functions.

5.20.2 Function Documentation

5.20.2.1 void initStateMachine (void)

Sets up the state machine (including timers) ready for use.

5.20.3 Variable Documentation

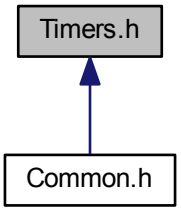
5.20.3.1 void(∗ Alpha_State_Ptr)(void)

Runs the next iteration of the state machine. Should be called from the main super-loop.

5.21 Timers.h File Reference

Real and virtual timer functions.

This graph shows which files directly or indirectly include this file:



Functions

- void timersSetupReal (void)

5.21.1 Detailed Description

Real and virtual timer functions. These functions should be run as part of the state machine setup.

See Also

StateMachine.h

5.21.2 Function Documentation

5.21.2.1 void timersSetupReal (void)

Sets up the real timers that run the state machine This should be called as part of the state machine initialisation.

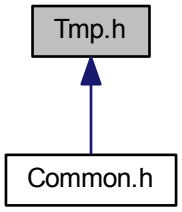
See Also

smInit()

5.22 Tmp.h File Reference

Temperature sensor functions.

This graph shows which files directly or indirectly include this file:



Macros

- #define ADC_SW3A_STATE 0x00
- #define ADC_SW3B_STATE 0x00
- #define ADC_I2C_BASE_ADDR 0x48
- #define ADC_I2C_ADDR (ADC_I2C_BASE_ADDR | ADC_SW3A_STATE) | (ADC_SW3B_STATE << 1)
- #define ADC_NUM_CHNL 0x08
- #define ADC_VREF 5.0
- #define ADC_STPS 256
- #define TMP_V0C_OFST 0.4
- #define TMP_SCL_OFST TMP_V0C_OFST * ADC_STPS / ADC_VREF
- #define TMP_E_T_COLD 1.5

Functions

- Uint16 initTemperature (void)
- Uint16 readTemperature (Uint16 chnl, float32 *tmpDest)

5.22.1 Detailed Description

Temperature sensor functions. The temperature sensor (MCP9701) output is read via an external ADC (ADS7830) that is connected to the I2C bus at address 10010xx where 'xx' is dependent upon the state of switch 3 (SW3) on the Ctrl PCB, which the user should record in the macros ADC_SW3A_STATE and ADC_SW3B_STATE.

All temperatures are in degrees Celsius.

Warning

Before any temperature functions can be used the I2C peripheral MUST be initialised, using i2cInit(), and initTemperature() must be run - initTemperature() will require the interrupts to be enabled globally.

See Also

i2cInit()

ADS7830 Ch #	Signal	Chnl selector	ADC Sel Code
CH0	TEMP1	load1 (0)	0
CH1	TEMP2	load2 (1)	4
CH2	TEMP3	load3 (2)	1
CH3	TEMP4	load4 (3)	5
CH4	TEMP5	XFMR_STAGE (4)	2
CH5	TEMP6	AC_STAGE (5)	6
CH6	TEMP_EXT1	EXT_1 (6)	3
CH7	TEMP_EXT2	EXT_2 (7)	7

5.22.2 Macro Definition Documentation

5.22.2.1 #define ADC_I2C_ADDR (ADC_I2C_BASE_ADDR | ADC_SW3A_STATE) | (ADC_SW3B_STATE << 1)

ADS7830 ADC complete I2C address.

5.22.2.2 #define ADC_I2C_BASE_ADDR 0x48

ADS7830 ADC base I2C address.

5.22.2.3 #define ADC_NUM_CHNL 0x08

ADS7830 ADC number of temperature channels.

5.22.2.4 #define ADC_STPS 256

ADS7830 ADC number of steps.

5.22.2.5 #define ADC_SW3A_STATE 0x00

The state of switch 3a; ON = 0x01, OFF = 0x00.

5.22.2.6 #define ADC_SW3B_STATE 0x00

The state of switch 3b; ON = 0x01, OFF = 0x00.

5.22.2.7 #define ADC_VREF 5.0

ADS7830 ADC reference voltage (volts).

5.22.2.8 #define TMP_E_T_COLD 1.5

MCP9701 Error at lowest operating temperature ($^\circ$ C), calculated as shown in Microchip AN1001.

5.22.2.9 #define TMP_SCL_OFST TMP_V0C_OFST * ADC_STPS / ADC_VREF

Scaled temperature offset.

5.22.2.10 #define TMP_V0C_OFST 0.4

MCP9701 Temperature sensor $V_{0^{\circ}C}$ (volts).

5.22.3 Function Documentation

5.22.3.1 Uint16 initTemperature (void)

Initialises the system for temperature readings. The I2C peripheral must be initialised before this function is used

See Also

i2cInit().

Returns

Error status.

5.22.3.2 Uint16 readTemperature (Uint16 *chnl*, float32 * *tmpDest*)

Queries the current temperature on the specified ADC channel.

Parameters

in	*chnl*	Specifies the channel the temperature is to be read from.
out	*tmpDest*	Address of the memory location at which to place the query result ($^{\circ}$ C).

Returns

Error status.

Index

AC_I_SCALE
Settings.h, 78
AC_IRMSLVL_MAX
Settings.h, 78
AC_OCP_TRIP
Ocp.h, 54
AC_OPP_TRIP
Opp.h, 59
AC_OTP_TRIP
Otp.h, 62
AC_OVP_TRIP
Ovp.h, 68
AC_PWRLVL_FIX
Settings.h, 78
AC_STAGE
MacroNets.h, 49
AC_V_R1
Settings.h, 78
AC_V_R2
Settings.h, 78
AC_VRMSLVL_MAX
Settings.h, 78
ADC_I2C_ADDR
Tmp.h, 97
ADC_I2C_BASE_ADDR
Tmp.h, 97
ADC_NUM_CHNL
Tmp.h, 97
ADC_STPS
Tmp.h, 97
ADC_SW3A_STATE
Tmp.h, 97
ADC_SW3B_STATE
Tmp.h, 97
ADC_VREF
Tmp.h, 97
ADCDRV_1ch_Rlt1
Adc.h, 29
ADCDRV_1ch_Rlt10
Adc.h, 29
ADCDRV_1ch_Rlt11
Adc.h, 29
ADCDRV_1ch_Rlt12
Adc.h, 29
ADCDRV_1ch_Rlt13
Adc.h, 29
ADCDRV_1ch_Rlt2
Adc.h, 29
ADCDRV_1ch_Rlt3
Adc.h, 30
ADCDRV_1ch_Rlt4
Adc.h, 30
ADCDRV_1ch_Rlt5
Adc.h, 30
ADCDRV_1ch_Rlt6
Adc.h, 30
ADCDRV_1ch_Rlt7
Adc.h, 30
ADCDRV_1ch_Rlt8
Adc.h, 30
ADCDRV_1ch_Rlt9
Adc.h, 30
ALL_DISABLED_WORD
EnableCtrl.h, 41
acFrequency
acStageSettings, 18
acICoefs
Cntl.h, 34
acNets
MacroNets.h, 52
acSettings
MacroNets.h, 52
acStageNets, 17
iFdbkNet, 17
iFiltOutNet, 17
iRefNet, 17
MacroNets.h, 50
vFdbkNet, 17
vRefNet, 17
acStageSettings, 18
acFrequency, 18
enable, 18
gainRate, 18
gainTarget, 18
iMaxRms, 18
iScale, 19
MacroNets.h, 50
mode, 19
ocpLevel, 19
otpLevel, 19
ovpLevel, 19
vGainLmt, 19
vMaxRms, 19
vScale, 19
acVCoefs
Cntl.h, 34
Adc.h, 27
ADCDRV_1ch_Rlt1, 29

ADCDRV_1ch_Rlt10, 29
ADCDRV_1ch_Rlt11, 29
ADCDRV_1ch_Rlt12, 29
ADCDRV_1ch_Rlt13, 29
ADCDRV_1ch_Rlt2, 29
ADCDRV_1ch_Rlt3, 30
ADCDRV_1ch_Rlt4, 30
ADCDRV_1ch_Rlt5, 30
ADCDRV_1ch_Rlt6, 30
ADCDRV_1ch_Rlt7, 30
ADCDRV_1ch_Rlt8, 30
ADCDRV_1ch_Rlt9, 30
getDcHvVoltage, 28
getDcMidCurrent, 28
getDcMidVoltage, 28
getLoadCurrent, 28
getLoadVoltage, 29
initAdc, 29
Alpha_State_Ptr
StateMachine.h, 94

cA1
Cntl.h, 32
cA2
Cntl.h, 32
cB0
Cntl.h, 32
cB1
Cntl.h, 32
cB2
Cntl.h, 32
cMax
Cntl.h, 32
cMin
Cntl.h, 32
CHANNEL_OOB
Settings.h, 78
CNTL_2P2Z_Coef1
Cntl.h, 34
CNTL_2P2Z_Coef2
Cntl.h, 34
CNTL_2P2Z_Coef3
Cntl.h, 34
CNTL_2P2Z_Coef4
Cntl.h, 34
CNTL_2P2Z_Coef5
Cntl.h, 34
CNTL_2P2Z_Coef6
Cntl.h, 34
CNTL_2P2Z_Fdbk1
Cntl.h, 34
CNTL_2P2Z_Fdbk2
Cntl.h, 34
CNTL_2P2Z_Fdbk3
Cntl.h, 35
CNTL_2P2Z_Fdbk4
Cntl.h, 35
CNTL_2P2Z_Fdbk5
Cntl.h, 35
CNTL_2P2Z_Fdbk6
Cntl.h, 35
CNTL_2P2Z_Out1
Cntl.h, 35
CNTL_2P2Z_Out2
Cntl.h, 35
CNTL_2P2Z_Out3
Cntl.h, 35
CNTL_2P2Z_Out4
Cntl.h, 35
CNTL_2P2Z_Out5
Cntl.h, 35
CNTL_2P2Z_Out6
Cntl.h, 35
CNTL_2P2Z_Ref1
Cntl.h, 35
CNTL_2P2Z_Ref2
Cntl.h, 35
CNTL_2P2Z_Ref3
Cntl.h, 36
CNTL_2P2Z_Ref4
Cntl.h, 36
CNTL_2P2Z_Ref5
Cntl.h, 36
CNTL_2P2Z_Ref6
Cntl.h, 36
cfType
Cntl.h, 32
checkAcOpp
Opp.h, 60
checkAcOtp
Otp.h, 63
checkAcOvp
Ovp.h, 68
checkDcHvOvp
Ovp.h, 68
checkDcMidOcp
Ocp.h, 54
checkDcOtp
Otp.h, 63
checkExtOtp
Otp.h, 63
checkLoadOcp
Ocp.h, 54
checkLoadOpp
Opp.h, 60
checkLoadOtp
Otp.h, 63
checkLoadOvp
Ovp.h, 69
circuitSection
EnableCtrl.h, 43
clearAcOcp
Ocp.h, 55
clearAcOpp
Opp.h, 60
clearAcOtp
Otp.h, 63

clearAcOvp
Ovp.h, 69
clearDcHvOvp
Ovp.h, 69
clearDcMidOcp
Ocp.h, 55
clearDcMidOvp
Ovp.h, 69
clearDcOtp
Otp.h, 63
clearExtOtp
Otp.h, 64
clearLoadOcp
Ocp.h, 55
clearLoadOpp
Opp.h, 60
clearLoadOtp
Otp.h, 64
clearLoadOvp
Ovp.h, 69
Cntl.h
cA1, 32
cA2, 32
cB0, 32
cB1, 32
cB2, 32
cMax, 32
cMin, 32
Cntl.h, 30
acICoefs, 34
acVCoefs, 34
CNTL_2P2Z_Coef1, 34
CNTL_2P2Z_Coef2, 34
CNTL_2P2Z_Coef3, 34
CNTL_2P2Z_Coef4, 34
CNTL_2P2Z_Coef5, 34
CNTL_2P2Z_Coef6, 34
CNTL_2P2Z_Fdbk1, 34
CNTL_2P2Z_Fdbk2, 34
CNTL_2P2Z_Fdbk3, 35
CNTL_2P2Z_Fdbk4, 35
CNTL_2P2Z_Fdbk5, 35
CNTL_2P2Z_Fdbk6, 35
CNTL_2P2Z_Out1, 35
CNTL_2P2Z_Out2, 35
CNTL_2P2Z_Out3, 35
CNTL_2P2Z_Out4, 35
CNTL_2P2Z_Out5, 35
CNTL_2P2Z_Out6, 35
CNTL_2P2Z_Ref1, 35
CNTL_2P2Z_Ref2, 35
CNTL_2P2Z_Ref3, 36
CNTL_2P2Z_Ref4, 36
CNTL_2P2Z_Ref5, 36
CNTL_2P2Z_Ref6, 36
cfType, 32
coefNum, 32
getAcICoef, 32
getAcVCoef, 32
getLoadICoef, 32
initCoefs, 33
loadICoefs, 36
setAcICoef, 33
setAcVCoef, 33
setLoadICoef, 33
coefNum
Cntl.h, 32
Common.h, 36
Comparator.h, 37
getAcDac, 38
initAcComparator, 38
initDcComparator, 38
initTripZone, 39
rstAcTripzone, 39
rstDcTripzone, 39
setAcDac, 39

DC_OTP_TRIP
Otp.h, 62
DCHV_OCP_TRIP
Ocp.h, 54
DCHV_OPP_TRIP
Opp.h, 59
DCHV_OVP_TRIP
Ovp.h, 68
DCHV_UVLCK_LMT
Settings.h, 78
DCMID_OCP_TRIP
Ocp.h, 54
DCMID_OPP_TRIP
Opp.h, 59
DCMID_OVP_TRIP
Ovp.h, 68
DCMID_VDCLVL_FIX
Settings.h, 78
DCMIDV_STRTUP_LMT
Settings.h, 78
DEBUG
Settings.h, 78
DPL_ISR
Pwm.h, 73
DUAL_CNTL_AC
Settings.h, 78
detectSlaveMode
slaveMode.h, 87
disableCircuit
EnableCtrl.h, 43
disableSinePhaseOut
SineGen.h, 83

EXT1_OTP_TRIP
Otp.h, 62
EXT2_OTP_TRIP
Otp.h, 62
EXT_1
MacroNets.h, 49
EXT_2

MacroNets.h, 49
ecSection
EnableCtrl.h, 43
enable
acStageSettings, 18
loadStageSettings, 22
xfmrStageSettings, 25
enableAllFlag
MacroNets.h, 52
enableCircuit
EnableCtrl.h, 43
EnableCtrl.h, 39
ALL_DISABLED_WORD, 41
circuitSection, 43
disableCircuit, 43
ecSection, 43
enableCircuit, 43
getCurrentState, 43
IOE_DEFVAL_ADDR, 41
IOE_GPINTEN_ADDR, 41
IOE_GPIO_ADDR, 41
IOE_GPPU_ADDR, 41
IOE_I2C_ADDR, 41
IOE_I2C_BASE_ADDR, 42
IOE_INTCAP_ADDR, 42
IOE_INTCON_ADDR, 42
IOE_INTF_ADDR, 42
IOE_IOCON_ADDR, 42
IOE_IODIR_ADDR, 42
IOE_IPOL_ADDR, 42
IOE_MAX_CHNL, 42
IOE_OLAT_ADDR, 42
IOE_SW4A_STATE, 42
IOE_SW4B_STATE, 42
initEnableControl, 43
resetEnableControl, 44
enableSinePhaseOut
SineGen.h, 83
ext1
MacroNets.h, 51
ext1OtpLevel
extDeviceSettings, 20
ext2
MacroNets.h, 51
ext2OtpLevel
extDeviceSettings, 20
extDeviceSettings, 19
ext1OtpLevel, 20
ext2OtpLevel, 20
extFanEnable, 20
extPsuEnable, 20
MacroNets.h, 50
extFanEnable
extDeviceSettings, 20
extPsuEnable
extDeviceSettings, 20
extSelect
MacroNets.h, 50, 51
extSettings
MacroNets.h, 52

gainRate
acStageSettings, 18
gainTarget
acStageSettings, 18
getAcDac
Comparator.h, 38
getAcICoef
Cntl.h, 32
getAcOcpLevel
Ocp.h, 55
getAcOcpState
Ocp.h, 55
getAcOppState
Opp.h, 60
getAcOtpLevel
Otp.h, 64
getAcOtpState
Otp.h, 64
getAcOvpLevel
Ovp.h, 69
getAcOvpState
Ovp.h, 70
getAcVCoef
Cntl.h, 32
getCurrentState
EnableCtrl.h, 43
getDcHvOvpLevel
Ovp.h, 70
getDcHvOvpState
Ovp.h, 70
getDcHvVoltage
Adc.h, 28
getDcMidCurrent
Adc.h, 28
getDcMidOcpLevel
Ocp.h, 55
getDcMidOcpState
Ocp.h, 57
getDcMidOvpState
Ovp.h, 70
getDcMidVoltage
Adc.h, 28
getDcOtpLevel
Otp.h, 64
getDcOtpState
Otp.h, 65
getExtOtpLevel
Otp.h, 65
getExtOtpState
Otp.h, 65
getLoadCurrent
Adc.h, 28
getLoadICoef
Cntl.h, 32
getLoadOcpLevel
Ocp.h, 57

getLoadOcpState
Ocp.h, 57
getLoadOppState
Opp.h, 60
getLoadOtpLevel
Otp.h, 65
getLoadOtpState
Otp.h, 65
getLoadOvpState
Ovp.h, 70
getLoadSlewStep
SlewControl.h, 88
getLoadSlewTarget
SlewControl.h, 88
getLoadState
SlewControl.h, 88
getLoadVoltage
Adc.h, 29
getSineGainStep
SineGen.h, 83
getSineGainTarget
SineGen.h, 83
getSineState
SineGen.h, 84

HV_V_R1
Settings.h, 78
HV_V_R2
Settings.h, 79
hvOvpLevel
xfmrStageSettings, 25
hvVMax
xfmrStageSettings, 25
hvVScale
xfmrStageSettings, 25
hvVSnsNet
xfmrStageNets, 24

I2C_ARDY_ISRC
I2c.h, 45
I2C_BUS_BUSY
Settings.h, 79
I2C_CLR_AL_BIT
I2c.h, 45
I2C_CLR_ARDY_BIT
I2c.h, 45
I2C_CLR_NACK_BIT
I2c.h, 46
I2C_CLR_RRDY_BIT
I2c.h, 46
I2C_CLR_SCD_BIT
I2c.h, 46
I2C_INVALID_ISRC
Settings.h, 79
I2C_MAX_BUFFER_SIZE
I2c.h, 46
I2C_MAX_PTR_SIZE
I2c.h, 46
I2C_MSGSTAT_INACTIVE
I2c.h, 46
I2C_MSGSTAT_RESTART
I2c.h, 46
I2C_READ_WRONG_MSG
Settings.h, 79
I2C_SCD_ISRC
I2c.h, 47
I2C_STP_NOT_READY
Settings.h, 79
I2C_WRITE_WRONG_MSG
Settings.h, 79
I2c.h, 44
I2C_ARDY_ISRC, 45
I2C_CLR_AL_BIT, 45
I2C_CLR_ARDY_BIT, 45
I2C_CLR_NACK_BIT, 46
I2C_CLR_RRDY_BIT, 46
I2C_CLR_SCD_BIT, 46
I2C_MAX_BUFFER_SIZE, 46
I2C_MAX_PTR_SIZE, 46
I2C_MSGSTAT_INACTIVE, 46
I2C_MSGSTAT_RESTART, 46
I2C_SCD_ISRC, 47
i2cPopMsg, 47
i2cRead, 47
i2cWrite, 47
initI2c, 47
i2cMsg, 20
msgBuffer, 21
msgStatus, 21
numOfBytes, 21
numSlavePtrBytes, 21
slaveAddress, 21
slavePtrAddrHigh, 21
slavePtrAddrLow, 21
i2cPopMsg
I2c.h, 47
i2cRead
I2c.h, 47
i2cWrite
I2c.h, 47
iFdbkNet
acStageNets, 17
loadStageNets, 22
iFiltOutNet
acStageNets, 17
loadStageNets, 22
iMax
loadStageSettings, 22
xfmrStageSettings, 25
iMaxRms
acStageSettings, 18
INCR_BUILD
Settings.h, 79
IOE_DEFVAL_ADDR
EnableCtrl.h, 41
IOE_GPINTEN_ADDR
EnableCtrl.h, 41

IOE_GPIO_ADDR
EnableCtrl.h, 41
IOE_GPPU_ADDR
EnableCtrl.h, 41
IOE_I2C_ADDR
EnableCtrl.h, 41
IOE_I2C_BASE_ADDR
EnableCtrl.h, 42
IOE_INTCAP_ADDR
EnableCtrl.h, 42
IOE_INTCON_ADDR
EnableCtrl.h, 42
IOE_INTF_ADDR
EnableCtrl.h, 42
IOE_IOCON_ADDR
EnableCtrl.h, 42
IOE_IODIR_ADDR
EnableCtrl.h, 42
IOE_IPOL_ADDR
EnableCtrl.h, 42
IOE_MAX_CHNL
EnableCtrl.h, 42
IOE_OLAT_ADDR
EnableCtrl.h, 42
IOE_SW4A_STATE
EnableCtrl.h, 42
IOE_SW4B_STATE
EnableCtrl.h, 42
iRefNet
acStageNets, 17
loadStageNets, 22
iScale
acStageSettings, 19
loadStageSettings, 23
xfmrStageSettings, 25
iSnsNet
xfmrStageNets, 24
initAcComparator
Comparator.h, 38
initAdc
Adc.h, 29
initCoefs
Cntl.h, 33
initDcComparator
Comparator.h, 38
initEnableControl
EnableCtrl.h, 43
initI2c
I2c.h, 47
initPwm
Pwm.h, 73
initSine
SineGen.h, 84
initStateMachine
StateMachine.h, 94
initTemperature
Tmp.h, 98
initTripZone
Comparator.h, 39
LOAD1_OCP_TRIP
Ocp.h, 54
LOAD1_OPP_TRIP
Opp.h, 59
LOAD1_OTP_TRIP
Otp.h, 62
LOAD1_OVP_TRIP
Ovp.h, 68
LOAD2_OCP_TRIP
Ocp.h, 54
LOAD2_OPP_TRIP
Opp.h, 59
LOAD2_OTP_TRIP
Otp.h, 62
LOAD2_OVP_TRIP
Ovp.h, 68
LOAD3_OCP_TRIP
Ocp.h, 54
LOAD3_OPP_TRIP
Opp.h, 59
LOAD3_OTP_TRIP
Otp.h, 62
LOAD3_OVP_TRIP
Ovp.h, 68
LOAD4_OCP_TRIP
Ocp.h, 54
LOAD4_OPP_TRIP
Opp.h, 59
LOAD4_OTP_TRIP
Otp.h, 62
LOAD4_OVP_TRIP
Ovp.h, 68
LOAD_0
MacroNets.h, 49
LOAD_1
MacroNets.h, 50
LOAD_2
MacroNets.h, 50
LOAD_3
MacroNets.h, 50
LOAD_I_SCALE
Settings.h, 79
LOAD_IDCLVL_MAX
Settings.h, 79
LOAD_PWRLVL_FIX
Settings.h, 79
LOAD_V_R1
Settings.h, 79
LOAD_V_R2
Settings.h, 79
LOAD_VDCLVL_FIX
Settings.h, 80
LOG_SIN
Settings.h, 80
load1
MacroNets.h, 51
load2

MacroNets.h, 51
load3
MacroNets.h, 51
load4
MacroNets.h, 51
loadICoefs
Cntl.h, 36
loadNets
MacroNets.h, 52
loadSettings
MacroNets.h, 52
loadStage
MacroNets.h, 50, 51
loadStageNets, 21
iFdbkNet, 22
iFiltOutNet, 22
iRefNet, 22
MacroNets.h, 50
vFdbkNet, 22
loadStageSettings, 22
enable, 22
iMax, 22
iScale, 23
MacroNets.h, 50
ocpLevel, 23
oppLevel, 23
otpLevel, 23
ovpLevel, 23
slewRate, 23
slewTarget, 23
vMax, 23
vScale, 23

MID_I_SCALE
Settings.h, 80
MID_V_R1
Settings.h, 80
MID_V_R2
Settings.h, 80
MacroNets.h
ext1, 51
ext2, 51
load1, 51
load2, 51
load3, 51
load4, 51
MacroNets.h, 48
AC_STAGE, 49
acNets, 52
acSettings, 52
acStageNets, 50
acStageSettings, 50
EXT_1, 49
EXT_2, 49
enableAllFlag, 52
extDeviceSettings, 50
extSelect, 50, 51
extSettings, 52
LOAD_0, 49
LOAD_1, 50
LOAD_2, 50
LOAD_3, 50
loadNets, 52
loadSettings, 52
loadStage, 50, 51
loadStageNets, 50
loadStageSettings, 50
runAll, 51
setupNets, 51
stopAll, 52
stopAllFlag, 52
XFMR_STAGE, 50
xfmrNets, 52
xfmrSettings, 52
xfmrStageNets, 50
xfmrStageSettings, 51
midVMax
xfmrStageSettings, 25
midVScale
xfmrStageSettings, 25
midVSnsNet
xfmrStageNets, 24
mode
acStageSettings, 19
msgBuffer
i2cMsg, 21
msgStatus
i2cMsg, 21

numOfBytes
i2cMsg, 21
numSlavePtrBytes
i2cMsg, 21

OCP_TRIP
Settings.h, 80
OPP_TRIP
Settings.h, 80
OTP_TRIP
Settings.h, 80
OVP_TRIP
Settings.h, 80
Ocp.h, 53
AC_OCP_TRIP, 54
checkDcMidOcp, 54
checkLoadOcp, 54
clearAcOcp, 55
clearDcMidOcp, 55
clearLoadOcp, 55
DCHV_OCP_TRIP, 54
DCMID_OCP_TRIP, 54
getAcOcpLevel, 55
getAcOcpState, 55
getDcMidOcpLevel, 55
getDcMidOcpState, 57
getLoadOcpLevel, 57
getLoadOcpState, 57
LOAD1_OCP_TRIP, 54

LOAD2_OCP_TRIP, 54
LOAD3_OCP_TRIP, 54
LOAD4_OCP_TRIP, 54
setAcOcpLevel, 57
setDcMidOcpLevel, 57
setLoadOcpLevel, 58
tripAcOcp, 58
ocpLevel
acStageSettings, 19
loadStageSettings, 23
xfmrStageSettings, 25
Opp.h, 58
AC_OPP_TRIP, 59
checkAcOpp, 60
checkLoadOpp, 60
clearAcOpp, 60
clearLoadOpp, 60
DCHV_OPP_TRIP, 59
DCMID_OPP_TRIP, 59
getAcOppState, 60
getLoadOppState, 60
LOAD1_OPP_TRIP, 59
LOAD2_OPP_TRIP, 59
LOAD3_OPP_TRIP, 59
LOAD4_OPP_TRIP, 59
oppLevel
loadStageSettings, 23
Otp.h, 61
AC_OTP_TRIP, 62
checkAcOtp, 63
checkDcOtp, 63
checkExtOtp, 63
checkLoadOtp, 63
clearAcOtp, 63
clearDcOtp, 63
clearExtOtp, 64
clearLoadOtp, 64
DC_OTP_TRIP, 62
EXT1_OTP_TRIP, 62
EXT2_OTP_TRIP, 62
getAcOtpLevel, 64
getAcOtpState, 64
getDcOtpLevel, 64
getDcOtpState, 65
getExtOtpLevel, 65
getExtOtpState, 65
getLoadOtpLevel, 65
getLoadOtpState, 65
LOAD1_OTP_TRIP, 62
LOAD2_OTP_TRIP, 62
LOAD3_OTP_TRIP, 62
LOAD4_OTP_TRIP, 62
RSRVD_OTP_TRIP, 62
setAcOtpLevel, 66
setDcOtpLevel, 66
setExtOtpLevel, 66
setLoadOtpLevel, 66
otpLevel
acStageSettings, 19
loadStageSettings, 23
xfmrStageSettings, 25
Ovp.h, 67
AC_OVP_TRIP, 68
checkAcOvp, 68
checkDcHvOvp, 68
checkLoadOvp, 69
clearAcOvp, 69
clearDcHvOvp, 69
clearDcMidOvp, 69
clearLoadOvp, 69
DCHV_OVP_TRIP, 68
DCMID_OVP_TRIP, 68
getAcOvpLevel, 69
getAcOvpState, 70
getDcHvOvpLevel, 70
getDcHvOvpState, 70
getDcMidOvpState, 70
getLoadOvpState, 70
LOAD1_OVP_TRIP, 68
LOAD2_OVP_TRIP, 68
LOAD3_OVP_TRIP, 68
LOAD4_OVP_TRIP, 68
setAcOvpLevel, 71
setDcHvOvpLevel, 71
tripDcMidOvp, 71
ovpLevel
acStageSettings, 19
loadStageSettings, 23

PWM_1_PRD
Pwm.h, 72
PWM_2_PRD
Pwm.h, 72
PWM_3_PRD
Pwm.h, 73
PWMDRV_2ch_UpCnt_Duty1A
Pwm.h, 73
PWMDRV_2ch_UpCnt_Duty1B
Pwm.h, 73
PWMDRV_2ch_UpCnt_Duty2A
Pwm.h, 73
PWMDRV_2ch_UpCnt_Duty2B
Pwm.h, 73
PWMDRV_2ch_UpCnt_Duty3A
Pwm.h, 73
PWMDRV_2ch_UpCnt_Duty3B
Pwm.h, 73
Pwm.h, 72
DPL_ISR, 73
initPwm, 73
PWM_1_PRD, 72
PWM_2_PRD, 72
PWM_3_PRD, 73
PWMDRV_2ch_UpCnt_Duty1A, 73
PWMDRV_2ch_UpCnt_Duty1B, 73
PWMDRV_2ch_UpCnt_Duty2A, 73
PWMDRV_2ch_UpCnt_Duty2B, 73

PWMDRV_2ch_UpCnt_Duty3A, 73
PWMDRV_2ch_UpCnt_Duty3B, 73
pwmDutyNet
xfmrStageNets, 24

RECP_SQRT_2
Settings.h, 80
RSRVD_OTP_TRIP
Otp.h, 62
readTemperature
Tmp.h, 98
resetEnableControl
EnableCtrl.h, 44
rstAcTripzone
Comparator.h, 39
rstDcTripzone
Comparator.h, 39
runAll
MacroNets.h, 51

SCIBAUD_MIN
Sci.h, 74
SCIFFRX_INT_LVL
Sci.h, 74
SCIFFTX_FILL_LVL
Sci.h, 74
SCIFFTX_INT_LVL
Sci.h, 75
SCPI_specificCmds.h, 75
SGENTI_1ch_VOut
SineGen.h, 85
SIN_CHANNEL
SineGen.h, 82
SIN_DFLT_GAIN
SineGen.h, 82
SIN_DFLT_RCTFY
SineGen.h, 82
SIN_F_ISR
SineGen.h, 82
SIN_F_ISR_DVSR
SineGen.h, 82
SIN_F_MAX
SineGen.h, 83
SIN_F_REQ
SineGen.h, 83
SIN_OFST
SineGen.h, 83
SIN_PHSE
SineGen.h, 83
SPI_DFLT_BAUD
Spi.h, 92
SPI_DFLT_CPHA
Spi.h, 92
SPI_DFLT_CPOL
Spi.h, 92
SPI_FFRX_INTLVL
Spi.h, 92
SPI_FFTX_FILLVL
Spi.h, 92
SPI_FFTX_INTLVL
Spi.h, 92
SQRT_2
Settings.h, 80
scSetLoadStateAll
SlewControl.h, 88
Sci.h, 73
SCIBAUD_MIN, 74
SCIFFRX_INT_LVL, 74
SCIFFTX_FILL_LVL, 74
SCIFFTX_INT_LVL, 75
sciInit, 75
sciTx, 75
sciInit
Sci.h, 75
sciTx
Sci.h, 75
setAcDac
Comparator.h, 39
setAcICoef
Cntl.h, 33
setAcOcpLevel
Ocp.h, 57
setAcOtpLevel
Otp.h, 66
setAcOvpLevel
Ovp.h, 71
setAcVCoef
Cntl.h, 33
setDcHvOvpLevel
Ovp.h, 71
setDcMidOcpLevel
Ocp.h, 57
setDcOtpLevel
Otp.h, 66
setExtOtpLevel
Otp.h, 66
setLoadICoef
Cntl.h, 33
setLoadOcpLevel
Ocp.h, 58
setLoadOtpLevel
Otp.h, 66
setLoadSlewStep
SlewControl.h, 90
setLoadSlewTarget
SlewControl.h, 90
setLoadState
SlewControl.h, 90
setSineGainStep
SineGen.h, 84
setSineGainTarget
SineGen.h, 84
setSineRmsTarget
SineGen.h, 84
setSineState
SineGen.h, 85
Settings.h, 76

AC_I_SCALE, 78
AC_IRMSLVL_MAX, 78
AC_PWRLVL_FIX, 78
AC_V_R1, 78
AC_V_R2, 78
AC_VRMSLVL_MAX, 78
CHANNEL_OOB, 78
DCHV_UVLCK_LMT, 78
DCMID_VDCLVL_FIX, 78
DCMIDV_STRTUP_LMT, 78
DEBUG, 78
DUAL_CNTL_AC, 78
HV_V_R1, 78
HV_V_R2, 79
I2C_BUS_BUSY, 79
I2C_INVALID_ISRC, 79
I2C_READ_WRONG_MSG, 79
I2C_STP_NOT_READY, 79
I2C_WRITE_WRONG_MSG, 79
INCR_BUILD, 79
LOAD_I_SCALE, 79
LOAD_IDCLVL_MAX, 79
LOAD_PWRLVL_FIX, 79
LOAD_V_R1, 79
LOAD_V_R2, 79
LOAD_VDCLVL_FIX, 80
LOG_SIN, 80
MID_I_SCALE, 80
MID_V_R1, 80
MID_V_R2, 80
OCP_TRIP, 80
OPP_TRIP, 80
OTP_TRIP, 80
OVP_TRIP, 80
RECP_SQRT_2, 80
SQRT_2, 80
TRIP_ALERT, 80
UVLKT_TRIP, 81
VALUE_OOB, 81
VDDA, 81
VSSA, 81
XFMR_PWMF_FIX, 81
setupNets
MacroNets.h, 51
SineGen.h, 81
disableSinePhaseOut, 83
enableSinePhaseOut, 83
getSineGainStep, 83
getSineGainTarget, 83
getSineState, 84
initSine, 84
SGENTI_1cine, 51

Spi.h, 93
StateMachine.h, 94
Alpha_State_Ptr, 94
initStateMachine, 94
stopAll
MacroNets.h, 52
stopAllFlag
MacroNets.h, 52

TMP_E_T_COLD
Tmp.h, 97
TMP_SCL_OFST
Tmp.h, 97
TMP_V0C_OFST
Tmp.h, 97
TRIP_ALERT
Settings.h, 80
Timers.h, 95
timersSetupReal, 95
timersSetupReal
Timers.h, 95
Tmp.h, 96
ADC_I2C_ADDR, 97
ADC_I2C_BASE_ADDR, 97
ADC_NUM_CHNL, 97
ADC_STPS, 97
ADC_SW3A_STATE, 97
ADC_SW3B_STATE, 97
ADC_VREF, 97
initTemperature, 98
readTemperature, 98
TMP_E_T_COLD, 97
TMP_SCL_OFST, 97
TMP_V0C_OFST, 97
transPol
Spi.h, 93
tripAcOcp
Ocp.h, 58
tripDcMidOvp
Ovp.h, 71

UVLKT_TRIP
Settings.h, 81
updateLoadSlew
SlewControl.h, 90
updateSineGain
SineGen.h, 85
updateSineSignal
SineGen.h, 85

VALUE_OOB
Settings.h, 81
VDDA
Settings.h, 81
vFdbkNet
acStageNets, 17
loadStageNets, 22
vGainLmt
acStageSettings, 19
vMax
loadStageSettings, 23
vMaxRms
acStageSettings, 19
vRefNet
acStageNets, 17
VSSA
Settings.h, 81
vScale
acStageSettings, 19
loadStageSettings, 23

XFMR_PWMF_FIX
Settings.h, 81
XFMR_STAGE
MacroNets.h, 50
xfmrNets
MacroNets.h, 52
xfmrSettings
MacroNets.h, 52
xfmrStageNets, 23
hvVSnsNet, 24
iSnsNet, 24
MacroNets.h, 50
midVSnsNet, 24
pwmDutyNet, 24
xfmrStageSettings, 24
enable, 25
hvOvpLevel, 25
hvVMax, 25
hvVScale, 25
iMax, 25
iScale, 25
MacroNets.h, 51
midVMax, 25
midVScale, 25
ocpLevel, 25
otpLevel, 25