F2802x Burn In Unit

Code Reference Manual

Wed Aug 14 2013

Contents

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.

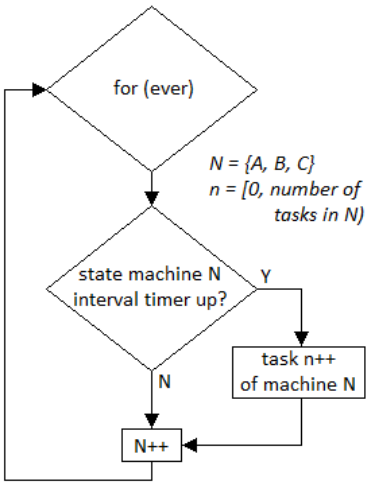


Figure 1.1: Sequential iteration thread state machine

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.

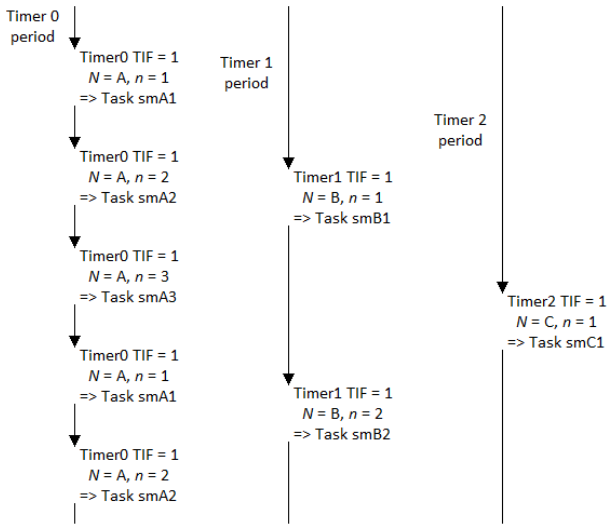


Figure 1.2: Sequence of thread tasks

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.

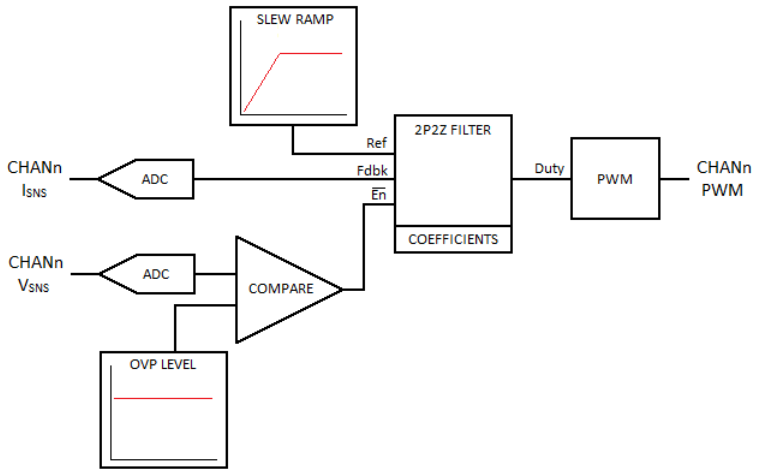


Figure 1.3: Macro view of control loop for single boost load

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.

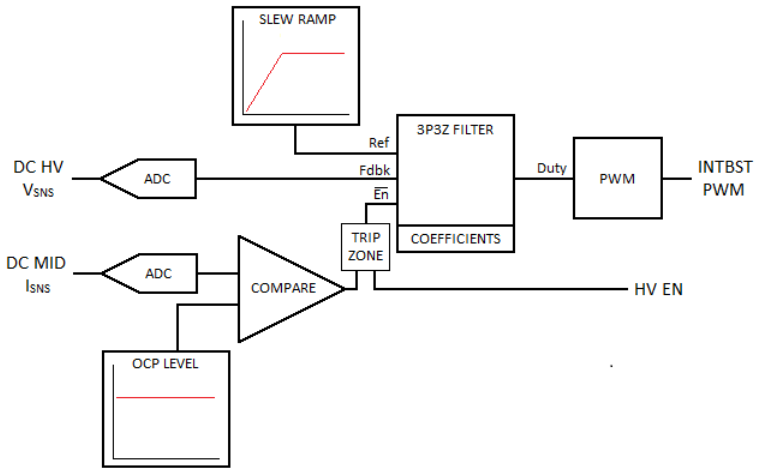


Figure 1.4: Macro view of control loop for inter-boost

1.2.3 AC Bridge Control

The following figure shows a diagram that illustrates he 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} vlaueagainst the user-set allowable current limit, the OCP limit. If the value is above the limit tha comparator output sets the relted trip zone, quickly disabling the control loop output and all other stages output, STOP ALL.

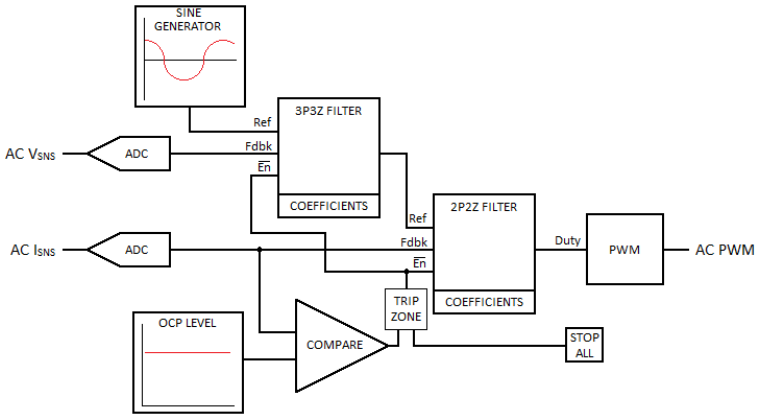


Figure 1.5: Macro view of control loop for AC bridge

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>)

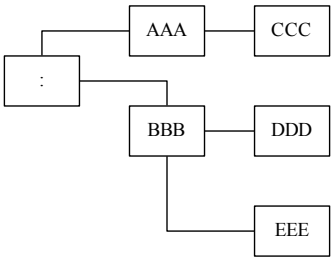
```
0x2A 0x49 0x44 0x4E 0x3F 0x3B 0x0A
```

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` subssytem 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:

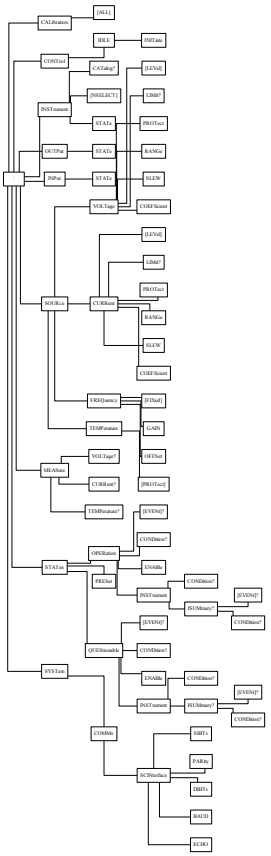
```
INSTrument:NSELect 5;:SOURce:VOLTage:LEVel 10;:SOURce:VOLTage:RA-
NGe 1;
```

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 channelParameters Struct Reference

`#include <MacroNets.h>`

Data Fields

- volatile int32 refNet
- volatile int32 iFdbkNet
- volatile int32 vFdbkNet
- volatile int32 outNet
- int32 ocp
- int32 ovp
- int32 target
- int32 slewRate
- int16 otp
- int16 iMaxRms
- int16 iMinRms
- int16 vMaxRms
- int16 vMinRms
- int16 iScale
- int16 vScale
- int16 vGainLmt
- opType opMode
- ctlType ctlMode
- Uint16 acFrequency
- Uint16 chEnable

4.1.1 Detailed Description

A structure used to represent the collection of settings pertaining to a particular channel or stage. Note that DPLib CNTL coefficient structures are handled separately to reduce complexity as DPLib expects them to be arranged in a certain manner in memory.

4.1.2 Field Documentation

4.1.2.1 Uint16 channelParameters::acFrequency

Sine signal generator frequency setting (Hz).

4.1.2.2 Uint16 channelParameters::chEnable

Channel enable status {FALSE, TRUE}.

4.1.2.3 ctlType channelParameters::ctlMode

Control mode setting {iCtrl, vCtrl}.

4.1.2.4 volatile int32 channelParameters::iFdbkNet

Current feednack net (IQ24).

4.1.2.5 int16 channelParameters::iMaxRms

Maximum RMS current setting limit (SQ10).

4.1.2.6 int16 channelParameters::iMinRms

Minimum RMS current setting limit (SQ10).

4.1.2.7 int16 channelParameters::iScale

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

4.1.2.8 int32 channelParameters::ocp

Normalised OCP limit (IQ24).

4.1.2.9 opType channelParameters::opMode

Output mode setting {dc, ac}.

4.1.2.10 int16 channelParameters::otp

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

4.1.2.11 volatile int32 channelParameters::outNet

IIR filter control law output net (IQ24).

4.1.2.12 int32 channelParameters::ovp

Normalised OVP limit (IQ24).

4.1.2.13 volatile int32 channelParameters::refNet

Net for CNTL reference (IQ24).

4.1.2.14 int32 channelParameters::slewRate

IIR filter control law reference slew rate (IQ24).

4.1.2.15 int32 channelParameters::target

IIR filter control law reference slew target (IQ24).

4.1.2.16 volatile int32 channelParameters::vFdbkNet

Voltage feedback net (IQ24).

4.1.2.17 int16 channelParameters::vGainLmt

Sine signal generator voltage gain limit (SQ14).

4.1.2.18 int16 channelParameters::vMaxRms

Maximum RMS voltage setting limit (SQ10).

4.1.2.19 int16 channelParameters::vMinRms

Minimum RMS voltage setting limit (SQ10).

4.1.2.20 int16 channelParameters::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

4.2 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]

4.2.1 Detailed Description

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

4.2.2 Field Documentation

4.2.2.1 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.

4.2.2.2 volatile Uint16 i2cMsg::msgStatus

Indicates which state the message is in.

4.2.2.3 Uint16 i2cMsg::numOfBytes

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

4.2.2.4 Uint16 i2cMsg::numSlavePtrBytes

The number of slave register pointer address bytes.

4.2.2.5 Uint16 i2cMsg::slaveAddress

The slave device I2C address this message is intended for.

4.2.2.6 Uint16 i2cMsg::slavePtrAddrHigh

The slave register pointer high byte.

4.2.2.7 Uint16 i2cMsg::slavePtrAddrLow

The slave register pointer low byte.

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

- I2c.h

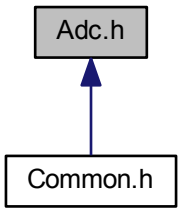
Chapter 5

File Documentation

5.1 Adc.h File Reference

ADC, DAC, comparator and related functions.

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



Functions

- void adcSocCnf (void)
- void adcMacroConfigure (void)
- void adcCompConfigure (void)
- Uint16 adcCheckOcp (void)
- Uint16 adcCheckOvp (void)
- Uint16 adcSetDac (Uint16 chnl, float32 dacLvl)
- Uint16 adcSetIScale (Uint16 chnl, float32 scaleSetting)
- Uint16 adcSetVScale (Uint16 chnl, float32 scaleSetting)
- Uint16 adcSetOcp (Uint16 chnl, float32 ocpSetting)
- Uint16 adcSetOvp (Uint16 chnl, float32 ovpSetting)
- Uint16 adcGetDac (Uint16 chnl, float32 ∗dacDest)
- Uint16 adcGetIScale (Uint16 chnl, float32 ∗sclDest)
- Uint16 adcGetVScale (Uint16 chnl, float32 ∗sclDest)
- Uint16 adcGetVoltage (Uint16 chnl, float32 ∗vDest)
- Uint16 adcGetCurrent (Uint16 chnl, float32 ∗iDest)
- Uint16 adcGetOcp (Uint16 chnl, float32 ∗ocpDest)
- Uint16 adcGetOvp (Uint16 chnl, float32 ∗ovpDest)

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, DAC, comparator and related 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.

5.1.2 Function Documentation

5.1.2.1 Uint16 adcCheckOcp (void)

Checks the current current sense ADC readings against the OCP limits.

Returns

Error status

5.1.2.2 Uint16 adcCheckOvp (void)

Checks the current voltage sense ADC readings against the OVP limits.

Returns

Error status

5.1.2.3 void adcCompConfigure (void)

Configures the COMP 1 & 2 comparators using the internal DACs at inverting inputs.

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

5.1.2.4 Uint16 adcGetCurrent (Uint16 *chnl*, float32 * *iDest*)

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

Parameters

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

Returns

Error status.

5.1.2.5 Uint16 adcGetDac (Uint16 *chnl,* float32 * *dacDest*)

Queries the output level setting of the DAC on the inverting input of the comparators.

Parameters

in	*chnl*	Specifies the channel number on which the setting is to be queried.
out	*dacDest*	Address of the memory location at which to place the query result (volts or amps).

Returns

Error status.

5.1.2.6 Uint16 adcGetIScale (Uint16 *chnl,* float32 * *sclDest*)

Queries the current current scaling setting of the specified channel.

Parameters

in	*chnl*	Specifies the channel number on which the setting is to be queried.
out	*sclDest*	Address of the memory location at which to place the query result (amps).

Returns

Error status.

5.1.2.7 Uint16 adcGetOcp (Uint16 *chnl,* float32 * *ocpDest*)

Queries the over current protection setting for the specified channel.

Parameters

in	*chnl*	Specifies the channel number on which the setting is to be queried.
out	*ocpDest*	Address of the memory location at which to place the query result (amps).

Returns

Error status.

5.1.2.8 Uint16 adcGetOvp (Uint16 *chnl,* float32 * *ovpDest*)

Queries the over current protection setting for the specified channel.

Parameters

in	*chnl*	Specifies the channel number on which the setting is to be queried.
out	*ovpDest*	Address of the memory location at which to place the query result (volts).

Returns

Error status.

5.1.2.9 Uint16 adcGetVoltage (Uint16 *chnl*, float32 * *vDest*)

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

Parameters

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

Returns

Error status.

5.1.2.10 Uint16 adcGetVScale (Uint16 *chnl*, float32 * *sclDest*)

Queries the current voltage scaling setting of the specified channel.

Parameters

in	*chnl*	Specifies the channel number on which the setting is to be queried.
out	*sclDest*	Address of the memory location at which to place the query result (volts).

Returns

Error status.

5.1.2.11 void adcMacroConfigure (void)

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

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

5.1.2.12 Uint16 adcSetDac (Uint16 *chnl*, float32 *dacLvl*)

Sets the output levels of the DACs on the inverting input of the comparators. The function will determine the scaling to be applied by testing the ctrlMode setting of the channel specified. The respective channel's current or voltage MUST be set previously.

Parameters

in	*chnl*	Specifies the channel number the setting is to be applied to.
in	*dacLvl*	Specifies the value of the level setting to be applied (volts or amps).

Returns

Error status.

5.1.2.13 Uint16 adcSetIScale (Uint16 *chnl*, float32 *scaleSetting*)

Sets the current scaling for the specified channel.

Parameters

in	*chnl*	Specifies the channel number the setting is to be applied to.
in	*scaleSetting*	Specifies the value of the scaling setting to be applied (amps/volts).

Returns

Error status.

5.1.2.14 Uint16 adcSetOcp (Uint16 *chnl*, float32 *ocpSetting*)

Sets the over current protection limit for the specified channel.

Parameters

in	*chnl*	Specifies the channel number the setting is to be applied to.
in	*ocpSetting*	Specifies the value of the limit to be applied (Amps).

Returns

Error status.

5.1.2.15 Uint16 adcSetOvp (Uint16 *chnl*, float32 *ovpSetting*)

Sets the over voltage protection limit for the specified channel.

Parameters

in	*chnl*	Specifies the channel number the setting is to be applied to.
in	*ovpSetting*	Specifies the value of the limit to be applied (volts).

Returns

Error status.

5.1.2.16 Uint16 adcSetVScale (Uint16 *chnl*, float32 *scaleSetting*)

Sets the voltage scaling for the specified channel.

Parameters

in	*chnl*	Specifies the channel number the setting is to be applied to.
in	*scaleSetting*	Specifies the value of the scaling setting to be applied (volts/volts).

Returns

Error status.

5.1.2.17 void adcSocCnf (void)

Configures ADC SOC for ADC macro

5.1.3 Variable Documentation

5.1.3.1 volatile int32* ADCDRV_1ch_Rlt1

Channel 0 current sense ADC terminal pointer.

5.1.3.2 volatile int32* ADCDRV_1ch_Rlt10

Channel 3 voltage sense ADC terminal pointer.

5.1.3.3 volatile int32* ADCDRV_1ch_Rlt11

Interboost voltage sense ADC terminal pointer.

5.1.3.4 volatile int32* ADCDRV_1ch_Rlt12

AC stage voltage sense ADC terminal pointer.

5.1.3.5 volatile int32* ADCDRV_1ch_Rlt13

VMid voltage sense ADC terminal pointer.

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.

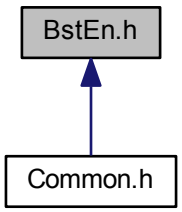
5.1.3.13 volatile int32* ADCDRV_1ch_Rlt9

Channel 2 voltage sense ADC terminal pointer.

5.2 BstEn.h File Reference

Functions for enabling and disabling the boost converter stages via I2C.

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



Macros

- #define IOE_I2C_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 BST_NUM_CHNL 0x04
- #define IOE_NUM_CHNL BST_NUM_CHNL

Functions

- Uint16 bcInit (void)
- Uint16 bcEnable (Uint16 chnl)
- Uint16 bcDisable (Uint16 chnl)

5.2.1 Detailed Description

Functions for enabling and disabling the boost converter stages via I2C. The converters are controlled via an external I/O expander (MCP23008) that is connected to the I2C bus at address 0100x-x-x where 'x-x-x' is dependent upon the configuration of resistors R60 - 61 & R70 - R74.

After bcInit() all converters default to disabled.

Warning

Before any converter control functions can be used the I2C peripheral MUST be initialised and EITHER bcInit() or fcInit() MUST be run - bcInit() will require the interrupts to be enabled globally.

See Also

i2cInit()
fcInit()

5.2.2 Macro Definition Documentation

5.2.2.1 #define BST_NUM_CHNL 0x04

Number of boost converter channels.

5.2.2.2 #define IOE_DEFVAL_ADDR 0x03

MCP23008 I/O expander default value register address.

5.2.2.3 #define IOE_GPINTEN_ADDR 0x02

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

5.2.2.4 #define IOE_GPIO_ADDR 0x09

MCP23008 I/O expander GPIO port register address.

5.2.2.5 #define IOE_GPPU_ADDR 0x06

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

5.2.2.6 #define IOE_I2C_ADDR 0x20

MCP23008 I/O expander I2C address (slave, 32d, 8-bit I/O expander).

5.2.2.7 #define IOE_INTCAP_ADDR 0x08

MCP23008 I/O expander interrupt capture register address.

5.2.2.8 #define IOE_INTCON_ADDR 0x04

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

5.2.2.9 #define IOE_INTF_ADDR 0x07

MCP23008 I/O expander interrupt flag register address.

5.2.2.10 #define IOE_IOCON_ADDR 0x05

MCP23008 I/O expander configuration register address.

5.2.2.11 #define IOE_IODIR_ADDR 0x00

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

5.2.2.12 #define IOE_IPOL_ADDR 0x01

MCP23008 I/O expander input polarity register address.

5.2.2.13 #define IOE_NUM_CHNL BST_NUM_CHNL

Total number of MCP I/O expander channels.

5.2.2.14 #define IOE_OLAT_ADDR 0x0A

MCP23008 I/O expander output latch register address.

5.2.3 Function Documentation

5.2.3.1 Uint16 bcDisable (Uint16 *chnl*)

Disables the specified channel's boost converter. The I2C peripheral and the boost converter enable controller interface MUST be initialised before this function is used.

See Also

i2cInit()
bcInit()

Parameters

in	*chnl*	Specifies the channel boost that is to be disabled.

Returns

Error status.

5.2.3.2 Uint16 bcEnable (Uint16 *chnl*)

Enables the specified channel's boost converter. The I2C peripheral and the boost converter enable controller interface MUST be initialised before this function is used.

See Also

i2cInit()
bcInit()

Parameters

in	*chnl*	Specifies the channel boost that is to be enabled.

Returns

Error status.

5.2.3.3 Uint16 bcInit (void)

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

See Also

i2cInit()

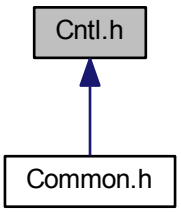
Returns

Error status.

5.3 Cntl.h File Reference

DPLib CNTL Macro related helper functions.

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



Macros

- #define SATMAX_MAX 0.9f

Typedefs

- typedef enum coefNum cfType

Enumerations

- enum coefNum { ,
  cMin = firstCoef, cMax, cB0, cB1,
  cA1, cB2, cA2, cB3,
  cA3 }

Functions

- void cntlUpdateCoefs (void)
- Uint16 cntlGetCoef (Uint16 chnl, cfType coef, float32 *valDest)
- Uint16 cntlSetCoef (Uint16 chnl, cfType coef, float32 val)

Variables

- volatile int32 * CNTL_2P2Z_Coef1
- volatile int32 * CNTL_2P2Z_Coef2
- volatile int32 * CNTL_2P2Z_Coef3
- volatile int32 * CNTL_2P2Z_Coef4
- volatile int32 * CNTL_2P2Z_Coef5
- volatile int32 * CNTL_2P2Z_Fdbk1
- volatile int32 * CNTL_2P2Z_Fdbk2
- volatile int32 * CNTL_2P2Z_Fdbk3
- volatile int32 * CNTL_2P2Z_Fdbk4
- volatile int32 * CNTL_2P2Z_Fdbk5
- volatile int32 * CNTL_2P2Z_Out1
- volatile int32 * CNTL_2P2Z_Out2
- volatile int32 * CNTL_2P2Z_Out3
- volatile int32 * CNTL_2P2Z_Out4
- volatile int32 * CNTL_2P2Z_Out5
- volatile int32 * CNTL_2P2Z_Ref1
- volatile int32 * CNTL_2P2Z_Ref2
- volatile int32 * CNTL_2P2Z_Ref3
- volatile int32 * CNTL_2P2Z_Ref4
- volatile int32 * CNTL_2P2Z_Ref5
- volatile int32 * CNTL_3P3Z_Coef1
- volatile int32 * CNTL_3P3Z_Coef2
- volatile int32 * CNTL_3P3Z_Fdbk1
- volatile int32 * CNTL_3P3Z_Fdbk2
- volatile int32 * CNTL_3P3Z_Out1
- volatile int32 * CNTL_3P3Z_Out2
- volatile int32 * CNTL_3P3Z_Ref1
- volatile int32 * CNTL_3P3Z_Ref2
- struct CNTL_2P2Z_CoefStruct coefs2 [NUM_ICTRL_CHNLS]
- struct CNTL_3P3Z_CoefStruct coefs3 [NUM_VCTRL_CHNLS]

5.3.1 Detailed Description

DPLib CNTL Macro related helper functions.

5.3.2 Macro Definition Documentation

5.3.2.1 #define SATMAX_MAX 0.9f

The maximum allowable value for the IIR filter control law's maximum saturation.

5.3.3 Typedef Documentation

5.3.3.1 typedef enum coefNum cfType

A type that allows a reference to a CNTL coefficient.

5.3.4 Enumeration Type Documentation

5.3.4.1 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.

cB3 B3 coefficient reference.

cA3 A3 coefficient reference.

5.3.5 Function Documentation

5.3.5.1 Uint16 cntlGetCoef (Uint16 *chnl*, cfType *coef*, float32 * *valDest*)

Queries the specified IIR filter control law coefficient for the specified channel.

Parameters

in	*chnl*	Specifies the channel number on which the setting is to be queried.
in	*coef*	Specifies the coefficient to be queried.
out	*valDest*	Address of the memory location at which to place the query result.

Returns

Error status.

5.3.5.2 Uint16 cntlSetCoef (Uint16 *chnl*, cfType *coef*, float32 *val*)

Sets the specified IIR filter control law coefficient for the specified channel.

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

Parameters

in	*chnl*	Specifies the channel number the setting is to be applied to [0, NUM_CHNLS).
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[coef] and cfLmts[coef + cA3].

Returns

Error status.

5.3.5.3 void cntlUpdateCoefs (void)

Updates the IIR filter control law's coefficients that are being used to those values set by the use of the other functions within this file.

5.3.6 Variable Documentation

5.3.6.1 volatile int32* CNTL_2P2Z_Coef1

Channel 0 IIR filter control law coefficient terminal pointer.

5.3.6.2 volatile int32* CNTL_2P2Z_Coef2

Channel 1 IIR filter control law coefficient terminal pointer.

5.3.6.3 volatile int32* CNTL_2P2Z_Coef3

Channel 2 IIR filter control law coefficient terminal pointer.

5.3.6.4 volatile int32* CNTL_2P2Z_Coef4

Channel 3 IIR filter control law coefficient terminal pointer.

5.3.6.5 volatile int32* CNTL_2P2Z_Coef5

Channel 4 IIR filter control law coefficient terminal pointer.

5.3.6.6 volatile int32* CNTL_2P2Z_Fdbk1

Channel 0 IIR filter control law feedback terminal pointer.

5.3.6.7 volatile int32* CNTL_2P2Z_Fdbk2

Channel 1 IIR filter control law feedback terminal pointer.

5.3.6.8 volatile int32* CNTL_2P2Z_Fdbk3

Channel 2 IIR filter control law feedback terminal pointer.

5.3.6.9 volatile int32* CNTL_2P2Z_Fdbk4

Channel 3 IIR filter control law feedback terminal pointer.

5.3.6.10 volatile int32* CNTL_2P2Z_Fdbk5

Channel 4 IIR filter control law feedback terminal pointer.

5.3.6.11 volatile int32* CNTL_2P2Z_Out1

Channel 0 IIR filter control law output terminal pointer.

5.3.6.12 volatile int32* CNTL_2P2Z_Out2

Channel 1 IIR filter control law output terminal pointer.

5.3.6.13 volatile int32* CNTL_2P2Z_Out3

Channel 2 IIR filter control law output terminal pointer.

5.3.6.14 volatile int32* CNTL_2P2Z_Out4

Channel 3 IIR filter control law output terminal pointer.

5.3.6.15 volatile int32* CNTL_2P2Z_Out5

Channel 4 IIR filter control law output terminal pointer.

5.3.6.16 volatile int32* CNTL_2P2Z_Ref1

Channel 0 IIR filter control law reference terminal pointer.

5.3.6.17 volatile int32* CNTL_2P2Z_Ref2

Channel 1 IIR filter control law reference terminal pointer.

5.3.6.18 volatile int32* CNTL_2P2Z_Ref3

Channel 2 IIR filter control law reference terminal pointer.

5.3.6.19 volatile int32* CNTL_2P2Z_Ref4

Channel 3 IIR filter control law reference terminal pointer.

5.3.6.20 volatile int32* CNTL_2P2Z_Ref5

Channel 4 IIR filter control law reference terminal pointer.

5.3.6.21 volatile int32* CNTL_3P3Z_Coef1

Interboost IIR filter control law coefficient terminal pointer.

5.3.6.22 volatile int32* CNTL_3P3Z_Coef2

AC stage IIR filter control law coefficient terminal pointer.

5.3.6.23 volatile int32* CNTL_3P3Z_Fdbk1

Interboost IIR filter control law feedback terminal pointer.

5.3.6.24 volatile int32* CNTL_3P3Z_Fdbk2

AC stage IIR filter control law feedback terminal pointer.

5.3.6.25 volatile int32* CNTL_3P3Z_Out1

Interboost IIR filter control law output terminal pointer.

5.3.6.26 volatile int32* CNTL_3P3Z_Out2

AC stage IIR filter control law output terminal pointer.

5.3.6.27 volatile int32* CNTL_3P3Z_Ref1

Interboost IIR filter control law reference terminal pointer.

5.3.6.28 volatile int32* CNTL_3P3Z_Ref2

AC stage IIR filter control law reference terminal pointer.

5.3.6.29 struct CNTL_2P2Z_CoefStruct coefs2[NUM_ICTRL_CHNLS]

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

5.3.6.30 struct CNTL_3P3Z_CoefStruct coefs3[NUM_VCTRL_CHNLS]

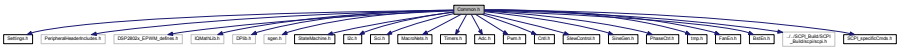
Array of structures that hold the 3-pole 3-zero IIR filter control law coefficient currently in use.

5.4 Common.h File Reference

Common include file for the project.

```
#include "Settings.h"
#include "PeripheralHeaderIncludes.h"
#include "DSP2802x_EPWM_defines.h"
#include "IQMathLib.h"
#include "SQMath.h"
#include "DPlib.h"
#include "sgen.h"
#include "StateMachine.h"
#include "I2c.h"
#include "Sci.h"
#include "MacroNets.h"
#include "Timers.h"
#include "Adc.h"
#include "Pwm.h"
#include "Cntl.h"
#include "SlewControl.h"
#include "SineGen.h"
#include "PhaseCtrl.h"
#include "tmp.h"
#include "FanEn.h"
#include "BstEn.h"
#include "../../SCPI_Build/SCPI_Build/scpi/scpi.h"
#include "SCPI_specificCmds.h"
```

Include dependency graph for Common.h:



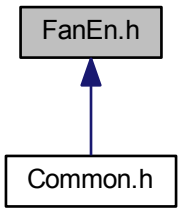
5.4.1 Detailed Description

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.

5.5 FanEn.h File Reference

Functions for enabling and disabling the external fans via I2C.

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



Macros

- #define IOE_I2C_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 FAN_NUM_CHNL 0x04
- #define FAN_CHNL_OFST 0x04

Functions

- Uint16 fcInit (void)
- Uint16 fcEnable (Uint16 chnl)
- Uint16 fcDisable (Uint16 chnl)

5.5.1 Detailed Description

Functions for enabling and disabling the external fans via I2C. The fans are controlled via an external I/O expander (MCP23008) that is connected to the I2C bus at address 0100x-x-x where 'x-x-x' is dependent upon the configuration of resistors R60 - 61 & R70 - R74.

After fcInit() all fans default to disabled.

Warning

Before any fan control functions can be used the I2C peripheral MUST be initialised and EITHER fcInit() or bcInit() must be run - fcInit() will require the interrupts to be enabled globally.

See Also

i2cInit()
bcInit()

5.5.2 Macro Definition Documentation

5.5.2.1 #define FAN_CHNL_OFST 0x04

Fan channel numbering offset

5.5.2.2 #define FAN_NUM_CHNL 0x04

Number of fan channels

5.5.2.3 #define IOE_DEFVAL_ADDR 0x03

MCP23008 I/O expander default value register address

5.5.2.4 #define IOE_GPINTEN_ADDR 0x02

MCP23008 I/O expander interrupt on change enable register address

5.5.2.5 #define IOE_GPIO_ADDR 0x09

MCP23008 I/O expander GPIO port register address

5.5.2.6 #define IOE_GPPU_ADDR 0x06

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

5.5.2.7 #define IOE_I2C_ADDR 0x20

MCP23008 I/O expander I2C address (slave, 32d, 8-bit I/O expander)

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_OLAT_ADDR 0x0A

MCP23008 I/O expander output latch register address

5.5.3 Function Documentation

5.5.3.1 Uint16 fcDisable (Uint16 *chnl*)

Disables the specified channel's fan The I2C peripheral and the fan enable controller interface MUST be initialised before this function is used.

See Also

i2cInit()
fcInit()

Parameters

in	*chnl*	Specifies the channel fan that is to be disabled

Returns

Error status

5.5.3.2 Uint16 fcEnable (Uint16 *chnl*)

Enables the specified channel's fan The I2C peripheral and the fan enable controller interface MUST be initialised before this function is used.

See Also

i2cInit()
fcInit()

Parameters

in	*chnl*	Specifies the channel fan that is to be enabled

Returns

Error status

5.5.3.3 Uint16 fcInit (void)

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

See Also

i2cInit()

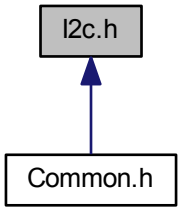
Returns

Error status

5.6 I2c.h File Reference

I2C communication 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 i2cInit (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

I2C communication 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

BstEn.h
FanEn.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 i2cInit (void)

Initialises the I2C-A peripheral and relevant interrupts. This function will clear any values already in the I2C peripheral registers. This function MUST be called before any other public I2C function.

5.6.3.2 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.

5.6.3.3 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.

5.6.3.4 Uint16 i2cWrite (**i2cMsg** ∗ *msg*)

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

Parameters

in	*msg*	The I2C message structure.

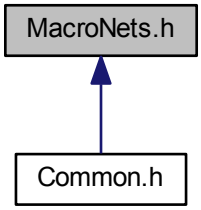
Returns

Error Status.

5.7 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 channelParameters

Macros

- #define LOAD_0 0
- #define LOAD_1 1
- #define LOAD_2 2
- #define LOAD_3 3
- #define AC_I_CNTL 4
- #define DC_STAGE 5
- #define AC_STAGE 6
- #define V_MID_CH 7

Typedefs

- typedef enum acOrDc opType
- typedef enum iOrVCtl ctlType

Enumerations

- enum acOrDc { dc = 0, ac = 1 }
- enum iOrVCtl { iCtrl = 0, vCtrl = 1 }

Functions

- void mnSetupChannels (void)
- void mnConnectNets (void)
- void mnStopAll (void)
- void mnRunAll (void)

Variables

- Uint16 stopAll
- Uint16 enableAll
- channelParameters channel [NUM_CHNLS+1]

5.7.1 Detailed Description

DPLib macro net and value control functions.

5.7.2 Macro Definition Documentation

5.7.2.1 #define AC_I_CNTL 4

The index position for AC I control settings.

5.7.2.2 #define AC_STAGE 6

The index position for AC stage settings.

5.7.2.3 #define DC_STAGE 5

The index position for DC stage settings.

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 V_MID_CH 7

The index position for VMid settings.

5.7.3 Typedef Documentation

5.7.3.1 typedef enum iOrVCtl ctlType

A type that allow specification of a channel's control mode setting.

5.7.3.2 typedef enum acOrDc opType

A type that allow specification of a channel's output mode setting.

5.7.4 Enumeration Type Documentation

5.7.4.1 enum acOrDc

The possible settings for channel output settings.

Enumerator

dc DC channel setting (0).

ac AC channel setting (1 or not-zero).

5.7.4.2 enum iOrVCtl

The possible settings for channel control setting.

Enumerator

iCtrl Current control setting (0).

vCtrl Voltage control setting (1 or not-zero).

5.7.5 Function Documentation

5.7.5.1 void mnConnectNets (void)

Connects the macro terminals to the relevant nets. This SHOULD be called AFTER DPL_Init()

5.7.5.2 void mnRunAll (void)

Enables all IIR filter control law reference inputs.

5.7.5.3 void mnSetupChannels (void)

Initialises all channel settings structures with their default values.

Warning

This MUST be called AFTER pwmMacroConfigure()

5.7.5.4 void mnStopAll (void)

Disables and zeros all IIR filter control law reference inputs, thus causing their outputs to ramp down to zero.

5.7.6 Variable Documentation

5.7.6.1 channelParameters channel[NUM_CHNLS+1]

A collection of the individual channel structures.

5.7.6.2 Uint16 enableAll

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

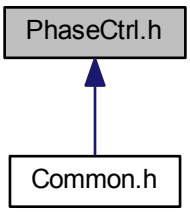
5.7.6.3 Uint16 stopAll

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

5.8 PhaseCtrl.h File Reference

Signal generator phase (ACFBPHASE) control function.

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



Functions

- void pcUpdate (void)

Variables

- volatile int32 ∗ PHASE_CTRL_In

5.8.1 Detailed Description

Signal generator phase (ACFBPHASE) control function.

Warning

This file is included by the file ISR.asm and thus any dependencies this file has should also be included there (e.g. PeripheralHeaderIncludes.h).

5.8.2 Function Documentation

5.8.2.1 void pcUpdate (void)

Updates GPIO19 based on state of ∗PHASE_CTRL_In terminal. Expects 0 (GPIO19 set) or non-zero (GPIO19 cleared). This is generally called by the DPL_ISR.asm

5.8.3 Variable Documentation

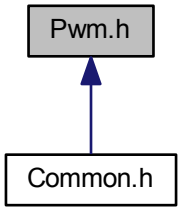
5.8.3.1 volatile int32∗ PHASE_CTRL_In

Phase control module signal input terminal.

5.9 Pwm.h File Reference

PWM and related functions.

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



Macros

- #define PERIOD 600

Functions

- void pwmTzConfigure (void)
- void pwmRstTz (void)
- void pwmMacroConfigure (void)
- void pwmSocConfigure (void)
- void pwmDPLTrigInit (void)
- Uint16 pwmSetFreq (Uint32 frq)
- Uint16 pwmGetFreq (Uint32 *frqDest)
- 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

5.9.1 Detailed Description

PWM and related functions.

5.9.2 Macro Definition Documentation

5.9.2.1 #define PERIOD 600

Defines the initial PWM period setting = 60MHz / 600 = 100.

5.9.3 Function Documentation

5.9.3.1 interrupt void DPL_ISR (void)

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

5.9.3.2 void pwmDPLTrigInit (void)

Initialises and enables PWM1 (master) to trigger the DPL ISR.

5.9.3.3 Uint16 pwmGetFreq (Uint32 * *frqDest*)

Queries the current PWM frequency setting.

Parameters

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

Returns

Error status.

5.9.3.4 void pwmMacroConfigure (void)

Configures each of the PWM macros for use.

5.9.3.5 void pwmRstTz (void)

Resets the trip zone after a comparator event.

5.9.3.6 Uint16 pwmSetFreq (Uint32 *frq*)

Sets the frequency of the PWMs.

Parameters

in	*frq*	Specifies the required frequency (hertz).

Returns

Error status.

5.9.3.7 void pwmSocConfigure (void)

Configures PWM1 (master) to generate ADC SOC start for ADC macro - configure before initialisation.

5.9.3.8 void pwmTzConfigure (void)

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

See Also

adc.h

5.9.4 Variable Documentation

5.9.4.1 volatile int32* PWMDRV_2ch_UpCnt_Duty1A

Channel 0 PWM terminal pointer.

5.9.4.2 volatile int32* PWMDRV_2ch_UpCnt_Duty1B

Channel 1 PWM terminal pointers.

5.9.4.3 volatile int32* PWMDRV_2ch_UpCnt_Duty2A

Channel 2 PWM terminal pointer.

5.9.4.4 volatile int32* PWMDRV_2ch_UpCnt_Duty2B

Channel 3 PWM terminal pointer.

5.9.4.5 volatile int32* PWMDRV_2ch_UpCnt_Duty3A

Interboost PWM terminal pointer.

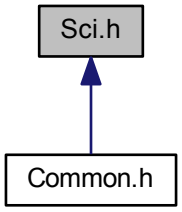
5.9.4.6 volatile int32* PWMDRV_2ch_UpCnt_Duty3B

AC stage PWM terminal pointer.

5.10 Sci.h File Reference

SCI communications 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)

5.10.1 Detailed Description

SCI communications 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.

5.10.2 Macro Definition Documentation

5.10.2.1 #define SCIBAUD_MIN 29

Minimum allowable value of SCI Baud.

5.10.2.2 #define SCIFFRX_INT_LVL 1

Interrupt level for receiving FIFO.

5.10.2.3 #define SCIFFTX_FILL_LVL 4

Fill level for transmission FIFO.

5.10.2.4 #define SCIFFTX_INT_LVL 0

Interrupt level for transmission FIFO.

5.10.3 Function Documentation

5.10.3.1 Uint16 sciInit (Uint32 *baud*)

Initialises the SCI(A) peripheral and relate interrupts.

Parameters

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

Returns

Error status.

Warning

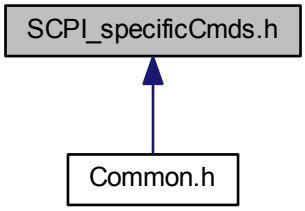
This function MUST be called before any other SCI(A) function.
This function will clear any values placed in the associated SCI(A) registers prior to calling.

5.10.3.2 void sciTx (void)

Transmits whatever data is on the SCPI output queue.

5.11 SCPI_specificCmds.h File Reference

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



5.11.1 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.

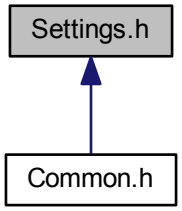
Node long and short names must conform to the standard as outlined by SCPI-99 in 6.2.1

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

5.12 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 DUAL_CNTL_AC
- #define VSSA 0l
- #define VMID_R1 540.0
- #define VMID_R2 4.3
- #define VAC_R1 540.0
- #define VAC_R2 4.3
- #define NUM_ICTRL_CHNLS 5
- #define NUM_VCTRL_CHNLS 2
- #define NUM_CHNLS NUM_ICTRL_CHNLS + NUM_VCTRL_CHNLS
- #define SQRT_2 1.41429
- #define RECP_SQRT_2 0.70711
- #define VDDA 3300l
- #define uSec100 6000

- #define CHANNEL_OOB 0x10
- #define VALUE_OOB 0x11
- #define OCP_TRIP 0x12
- #define OVP_TRIP 0x13
- #define OTP_TRIP 0x14
- #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

5.12.1 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.

5.12.2 Macro Definition Documentation

5.12.2.1 #define CHANNEL_OOB 0x10

Channel out of bounds error code.

5.12.2.2 #define DEBUG

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

5.12.2.3 #define DUAL_CNTL_AC

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

5.12.2.4 #define I2C_BUS_BUSY 0x23

I2C bus already busy error code.

5.12.2.5 #define I2C_INVALID_ISRC 0x24

Invalid I2C interrupt source error code.

5.12.2.6 #define I2C_READ_WRONG_MSG 0x20

Incorrect type I2C message read error code.

5.12.2.7 #define I2C_STP_NOT_READY 0x22

I2C stop bit was not yet received error code.

5.12.2.8 #define I2C_WRITE_WRONG_MSG 0x21

Incorrect type I2C write message error code.

5.12.2.9 #define INCR_BUILD 2

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

5.12.2.10 #define NUM_CHNLS NUM_ICTRL_CHNLS + NUM_VCTRL_CHNLS

Total number of IIR filter control law macros used (doesn't include VMID semi-channel).

5.12.2.11 #define NUM_ICTRL_CHNLS 5

The number of current, or 2-pole 2-zero, IIR filter control law macros used.

5.12.2.12 #define NUM_VCTRL_CHNLS 2

The number of voltage, or 3-pole 3-zero, IIR filter control law macros used.

5.12.2.13 #define OCP_TRIP 0x12

Over-current protection trip error code.

5.12.2.14 #define OTP_TRIP 0x14

Over-temperature protection trip error code.

5.12.2.15 #define OVP_TRIP 0x13

Over-voltage protection trip error code.

5.12.2.16 #define RECP_SQRT_2 0.70711

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

5.12.2.17 #define SQRT_2 1.41429

Sqrt(2) constant used for RMS calculations.

5.12.2.18 #define uSec100 6000

100us - System define.

5.12.2.19 #define VAC_R1 540.0

Scaling voltage divider R1 resistor value for VAC ADC.

5.12.2.20 #define VAC_R2 4.3

Scaling voltage divider R2 resistor value for VAC ADC.

5.12.2.21 #define VALUE_OOB 0x11

Value out of bounds error code.

5.12.2.22 #define VDDA 3300l

System VMAXREF (millivolts).

5.12.2.23 #define VMID_R1 540.0

Scaling voltage divider R1 resistor value for VMID ADC.

5.12.2.24 #define VMID_R2 4.3

Scaling voltage divider R2 resistor value for VMID ADC.

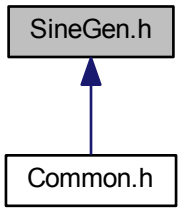
5.12.2.25 #define VSSA 0l

System VLOWREF (millivolts).

5.13 SineGen.h File Reference

Signal generator functions.

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



Macros

- #define SIN_DFLT_RCTFY TRUE
- #define SIN_DFLT_OFST 0
- #define SIN_DFLT_PHSE 0
- #define SIN_DFLT_GAIN 0.9
- #define SIN_DFLT_F 50.0
- #define SIN_DFLT_F_MAX 1000u
- #define SIN_CHANNEL AC_STAGE
- #define SIN_F_SPL 8250u

Functions

- void sgInit (void)
- void sgUpdate (void)
- void sgGainUpdate (void)
- Uint16 sgSetState (Uint16 stt)
- Uint16 sgSetRectify (Uint16 rfy)
- Uint16 sgSetOffset (float32 ofst)
- Uint16 sgSetInitialPhase (float32 phs)

- Uint16 sgSetGainTarget (float32 gnt)
- Uint16 sgSetFreq (Uint16 frq)
- Uint16 sgGetState (Uint16 *sttDest)
- Uint16 sgGetRectify (Uint16 *rfyDest)
- Uint16 sgGetOffset (float32 *oftDest)
- Uint16 sgGetGainTarget (float32 *gntDest)
- Uint16 sgGetFreq (Uint16 *frqDest)
- Uint16 sgGetResolution (float32 *rslDest)

Variables

- volatile int32 * SGENTI_1ch_VOut
- volatile int32 * SGENTI_1ch_Sign

5.13.1 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.

5.13.2 Macro Definition Documentation

5.13.2.1 #define SIN_CHANNEL AC_STAGE

Defines which channel enable controls the generator output.

5.13.2.2 #define SIN_DFLT_F 50.0

Initial frequency setting (hertz).

5.13.2.3 #define SIN_DFLT_F_MAX 1000u

Initial maximum frequency setting (hertz).

5.13.2.4 #define SIN_DFLT_GAIN 0.9

Initial gain setting [0.0, 1.0].

5.13.2.5 #define SIN_DFLT_OFST 0

Initial offset setting [-0.5, +0.5], IQ15.

5.13.2.6 #define SIN_DFLT_PHSE 0

Initial initial phase setting [0, 360), IQ16.

5.13.2.7 #define SIN_DFLT_RCTFY TRUE

Initial rectification setting [TRUE | FALSE).

5.13.2.8 #define SIN_F_SPL 8250u

Signal sampling frequency, i.e. the frequency that sgen.calc() is called at. This is dependent on ISR frequency, currently 1/4 of f_ISR, full ISR speed is 33,000Hz.

5.13.3 Function Documentation

5.13.3.1 void sgGainUpdate (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.

See Also

scSlewUpdate()

5.13.3.2 Uint16 sgGetFreq (Uint16 * *frqDest*)

Queries the current frequency setting.

Parameters

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

Returns

Error status.

5.13.3.3 Uint16 sgGetGainTarget (float32 * *gntDest*)

Queries the current target gain setting.

Parameters

out	*gntDest*	Address of the memory location at which to place the query result.

Returns

Error status.

5.13.3.4 Uint16 sgGetOffset (float32 * *oftDest*)

Queries the current signal DC offset setting.

Parameters

out	*oftDest*	Address of the memory location at which to place the query result.

Returns

Error status.

5.13.3.5 Uint16 sgGetRectify (Uint16 * *rfyDest*)

Queries the current state of the signal generator rectification enable.

Parameters

out	*rfyDest*	Address of the memory location at which to place the query result {1:ON | 0:OF}.

Returns

Error status.

5.13.3.6 Uint16 sgGetResolution (float32 * *rslDest*)

Queries the current frequency resolution. This is equal to f_{max} / step_max.

Parameters

out	*rslDest*	Address of the memory location at which to place the query result.

Returns

Error status.

5.13.3.7 Uint16 sgGetState (Uint16 * *sttDest*)

Queries the current state of the generator output.

Parameters

out	*sttDest*	Address of the memory location at which to place the query result {1:ON | 0:OFF}.

Returns

Error status.

5.13.3.8 void sgInit (void)

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

5.13.3.9 Uint16 sgSetFreq (Uint16 *frq*)

Sets the signal frequency.

Parameters

in	*frq*	Frequency value [0, f_{max}] (hertz).

Returns

Error status.

5.13.3.10 Uint16 sgSetGainTarget (float32 *gnt*)

Sets the target gain of the signal.

Parameters

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

Returns

Error status.

5.13.3.11 Uint16 sgSetInitialPhase (float32 *phs*)

Sets the signal initial phase value

Parameters

in	*phs*	Initial phase value [0, 360) (degrees).

Returns

Error status

5.13.3.12 Uint16 sgSetOffset (float32 *ofst*)

Sets the signal DC offset

Parameters

in	*ofst*	DC offset value [-0.5, +0.5].

Returns

Error status.

5.13.3.13 Uint16 sgSetRectify (Uint16 *rfy*)

Enables or disables the rectification of the generator output

Parameters

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

Returns

Error status.

5.13.3.14 Uint16 sgSetState (Uint16 *stt*)

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

Parameters

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

Returns

Error status.

5.13.3.15 void sgUpdate (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 value produced will be an absolute value.

5.13.4 Variable Documentation

5.13.4.1 volatile int32* SGENTI_1ch_Sign

Voltage sign (pre-rectification) output terminal.

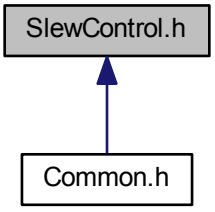
5.13.4.2 volatile int32* SGENTI_1ch_VOut

Voltage output terminal.

5.14 SlewControl.h File Reference

Slew control functions.

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



Functions

- void scSlewUpdate (void)
- Uint16 scSetTarget (Uint16 chnl, float32 trgt)
- Uint16 scSetStep (Uint16 chnl, float32 stp)
- Uint16 scSetState (Uint16 chnl, Uint16 stt)
- Uint16 scSetTargetAll (float32 trgt)
- Uint16 scSetStepAll (float32 stp)
- Uint16 scSetStateAll (Uint16 stt)

- Uint16 scGetTarget (Uint16 chnl, float32 *trgtDest)
- Uint16 scGetStep (Uint16 chnl, float32 *stpDest)
- Uint16 scGetState (Uint16 chnl, Uint16 *sttDest)

5.14.1 Detailed Description

Slew control functions.

5.14.2 Function Documentation

5.14.2.1 Uint16 scGetState (Uint16 *chnl*, Uint16 * *sttDest*)

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

Parameters

in	*chnl*	Specifies the channel the setting is to be read from.
out	*sttDest*	Address of the memory location at which to place the query result (0:OFF | non-zero:ON).

Returns

Error status.

5.14.2.2 Uint16 scGetStep (Uint16 *chnl*, float32 * *stpDest*)

Queries the current slew step size of the specified channel.

Parameters

in	*chnl*	Specifies the channel the setting is to be read from.
out	*stpDest*	Address of the memory location at which to place the query result (amps or volts).

Returns

Error status.

5.14.2.3 Uint16 scGetTarget (Uint16 *chnl*, float32 * *trgtDest*)

Queries the current slew target setting for the specified channel.

Parameters

in	*chnl*	Specifies the channel the setting is to be read from.
out	*trgtDest*	Address of the memory location at which to place the query result (amps or volts).

Returns

Error status.

5.14.2.4 Uint16 scSetState (Uint16 *chnl*, Uint16 *stt*)

Sets the reference net enable state for the specified channel.

Parameters

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

Returns

Error Status.

5.14.2.5 Uint16 scSetStateAll (Uint16 *stt*)

Sets all channels' reference net enable state.

Parameters

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

Returns

Error status.

5.14.2.6 Uint16 scSetStep (Uint16 *chnl,* float32 *stp*)

Sets the slew step size for the specified channel.

Parameters

in	*chnl*	Specifies the channel the setting is to be applied to.
in	*stp*	Specifies the value of the slew step size to be applied (amps or volts).

Returns

Error status.

5.14.2.7 Uint16 scSetStepAll (float32 *stp*)

Sets all channels' slew step size.

Parameters

in	*stp*	Specifies the value of the slew step size to be applied (amps or volts).

Returns

Error status.

5.14.2.8 Uint16 scSetTarget (Uint16 *chnl,* float32 *trgt*)

Sets the slew target for the specified channel.

Parameters

in	*chnl*	Specifies the channel the setting is to be applied to.
in	*trgt*	Specifies the value of the slew target to be applied (amps or volts).

Returns

Error status.

5.14.2.9 Uint16 scSetTargetAll (float32 *trgt*)

Sets all channels' slew target

Parameters

in	*trgt*	Specifies the value of the slew target to be applied (amps or volts).

Returns

Error status.

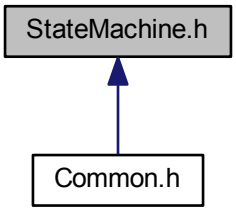
5.14.2.10 void scSlewUpdate (void)

Advances the slew ramps for all relevant channels. Does not apply to channels that use sine references

5.15 StateMachine.h File Reference

State machine functions.

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



Functions

- void smInit (void)

Variables

- void(* Alpha_State_Ptr)(void)

5.15.1 Detailed Description

State machine functions.

5.15.2 Function Documentation

5.15.2.1 void smInit (void)

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

5.15.3 Variable Documentation

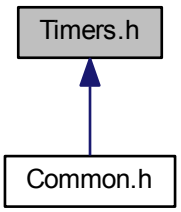
5.15.3.1 void(* Alpha_State_Ptr)(void)

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

5.16 Timers.h File Reference

Real and virtual timer functions.

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



Functions

- void timersSetupReal (void)

5.16.1 Detailed Description

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

See Also

StateMachine.h

5.16.2 Function Documentation

5.16.2.1 void timersSetupReal (void)

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

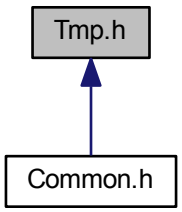
See Also

smInit()

5.17 Tmp.h File Reference

Temperature sensor functions.

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



Macros

- #define ADC_I2C_ADDR 0x48
- #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 tmpInit (void)
- Uint16 tmpSetOtp (Uint16 chnl, float32 tmp)
- Uint16 tmpGetOtp (Uint16 chnl, float32 ∗tmpDest)
- Uint16 tmpCheckOtp (void)
- Uint16 tmpRead (Uint16 chnl, float32 ∗tmpDest)

5.17.1 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 configuration of resistors R75 - R78. All temperatures are in degrees Celcius.

Warning

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

See Also

i2cInit()

5.17.2 Macro Definition Documentation

5.17.2.1 #define ADC_I2C_ADDR 0x48

Slave I2C address (ADS7830 8-channel ADC - A0 = 0, A1 = 0).

5.17.2.2 #define ADC_NUM_CHNL 0x08

Number of temperature channels. The program expects a 50/50 split with first half for the on-board temperature channels.

5.17.2.3 #define ADC_STPS 256

Number of ADS7830 ADC steps.

5.17.2.4 #define ADC_VREF 5.0

ADS7830 ADC reference voltage (volts).

5.17.2.5 #define TMP_E_T_COLD 1.5

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

5.17.2.6 #define TMP_SCL_OFST TMP_V0C_OFST $*$ ADC_STPS / ADC_VREF

Scaled temperature offset.

5.17.2.7 #define TMP_V0C_OFST 0.4

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

5.17.3 Function Documentation

5.17.3.1 Uint16 tmpCheckOtp (void)

Tests the current on-board temperature sensor readings against the OTP limits.

Returns

Error status.

5.17.3.2 Uint16 tmpGetOtp (Uint16 *chnl,* float32 * *tmpDest*)

Queries the on-board over temperature limit for the specified channel. The I2C peripheral and temperature reading interface MUST be initialised before this function is used.

Parameters

in	*chnl*	Specifies the channel the setting is to be read from.
out	*tmpDest*	Address of the memory location at which to place the query result (° C).

Returns

Error status.

5.17.3.3 Uint16 tmpInit (void)

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

See Also

i2cInit().

Returns

Error status.

5.17.3.4 Uint16 tmpRead (Uint16 *chnl*, float32 * *tmpDest*)

Queries the current on-board temperature of the specified 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 (° C).

Returns

Error status.

5.17.3.5 Uint16 tmpSetOtp (Uint16 *chnl*, float32 *tmp*)

Sets the on-board over temperature limit for the specified channel. The I2C peripheral and temperature reading interface MUST be initialised before this function is used.

Parameters

in	*chnl*	Specifies the channel the setting is to be applied to.
in	*tmp*	Specifies the value of the limit to be applied (° C).

Returns

Error status.

Index