DSP Shield Documentation

1.0

Generated by Doxygen 1.8.8

Sun Nov 9 2014 19:05:28

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 AudioClass Class Reference

Audio Class.

`#include <Audio.h>`

Public Member Functions

- int Audio (void)
- int Audio (int process)
- int close ()
- void attachIntr (void *function)
- void detachIntr (void)
- int read (void)
- int write (void)
- void isrDma (void)
- int setInputGain (int lgain, int rgain)
- int setOutputVolume (int volume)
- int setOutputVolume (int lvolume, int rvolume)
- int audioMute (void)
- int audioUnmute (void)
- int setSamplingRate (long)
- int HPL_RConF_Routing (int left)
- int HPR_RConF_Routing (int left, int right)
- int LOL_RConF_Routing (int left, int right)
- int LOR_RConF_Routing (int right)

Public Attributes

- Uint16 * audioInLeft [2]
- Uint16 * audioInRight [2]
- Uint16 * audioOutLeft [2]
- Uint16 * audioOutRight [2]
- unsigned short activeInBuf
- unsigned short activeOutBuf
- int sampleLeft
- int sampleRight

3.1.1 Detailed Description

Audio Class.

Contains prototypes for functions in Audio library

Address of audio data buffers is stored in the below pointers audioInLeft - Holds the adress of Audio input (read from codec) data buffers for left channel audioInRight - Holds the adress of Audio input (read from codec) data buffers for right channel audioOutLeft - Holds the adress of Audio output (write to codec) data buffers for left channel audioInRight - Holds the adress of Audio output (write to codec) data buffers for right channel

All the components described above are array of two pointers pointing to the two data buffers of given audio channel. When one data buffer is being used by audio module, other buffer can be used for data processing.

Index of the buffers being used by audio module are indicated by activeInBuf - Index of the buffer being used for audio input audioInLeft[activeInBuf] indcates left channel buffer audioInRight[activeInBuf] indcates right channel buffer activeOutBuf - Index of the buffer being used for audio output audioOutLeft[activeOutBuf] indcates left channel buffer audioOutRight[activeOutBuf] indcates right channel buffer

Data read from codec can accessed using the data buffer pointers 'audioInLeft', audioInRight and index 'activeInBuf'

When Audio library is configured to operate with same data buffers for read and write operation (initialized using Audio(void)), audioInLeft and audioOutLeft points to same buffers and audioInRight and audioInRight points to same buffers.

When Audio library is configured to operate with independent data buffers for read and write operation (initialized using Audio(int process)), applications need to take care of copying the data from audio input buffers (audioInLeftxx) to audio output buffers (audioOutxx) in DMA ISR.

3.1.2 Member Function Documentation

3.1.2.1 void AudioClass::attachIntr (void * *function*)

==========

attachInterrupt

Description
Function to assign ISR fot DMA interrupt.

Arguments

```
*        function - ISR function pointer
```

Return Value
None

Start the Audio module

3.1.2.2 int AudioClass::Audio (void)

==========

Audio

Description
Function initializes Audio Module.

This function is for enabling the audio loopback for routing data received from audio input to the audio output without much processing of the audio data received. Calling this function enables audio library to use same data buffers for audio input and output channels. While audio input process is filling one buffer, audio output process will empty other buffer. Using this function for audio module configuration is recommended for the applications which requires little or no processing of the audio input data. Calling this function requires NO buffer copying in the DMA ISR for routing the audio input data to audio output which will be taken care by audio library internally.

Arguments

Return Value CSL_Status

- CSL_SOK - Initializations are successful
- CSL_ESYS_FAIL - Initializations are not successful

Initialize the DMA transfer buffers

3.1.2.3 int AudioClass::Audio (int *process*)

==========

Audio

Description
Function initializes Audio Module.

This function is for enabling the audio loopback for routing data received from audio input to the audio output with processing involved on the audio input data. Calling this function with 'process' parameter set to 'TRUE' enables audio library to use independent data buffers for audio input and output channels. Using this function for audio module configuration is recommended for the applications which requires complex processing of the audio input data. Calling this function requires copying of the processed audio data to audio output buffers in the DMA ISR. Application should read the audio input data on the DMA read right channel interrupt and copy the processed audio data to audio output data buffers on DMA write right channel interrupt.

Note: Calling this function with 'process' parameter set to 'FALSE' is same as calling the function Audio(void).

Arguments

Return Value CSL_Status

- CSL_SOK - Initializations are successful
- CSL_ESYS_FAIL - Initializations are not successful

Initialize the DMA transfer buffers

3.1.2.4 int AudioClass::audioMute (void)

==========

audioMute

Description
Function to mute audio.

Arguments

Return Value CSL_Status

- CSL_SOK - Muting successful
- CSL_ESYS_FAIL - Muting is not successful

Select Page 0

Unmute audio

Small delay to allow the mute setting

3.1.2.5 int AudioClass::audioUnmute (void)

==========

audioUnmute

Description
Function to unmute audio.

Arguments

Return Value CSL_Status

- CSL_SOK - Muting successful
- CSL_ESYS_FAIL - Muting is not successful

Select Page 0

Unmute audio

Small delay to allow the mute setting

3.1.2.6 int AudioClass::close (void)

==========

close

Description
Function to close Audio module.

Arguments

Return Value CSL_Status

- CSL_SOK - close is successful
- CSL_ESYS_FAIL - close is not successful

Disable I2S

Reset codec

3.1.2.7 void AudioClass::detachIntr (void)

==========

detachInterrupt

Description
Function to detach ISR function pointer from the DMA interrupt.

Arguments

Return Value
None

3.1.2.8 int AudioClass::HPL_RConF_Routing (int *left*)

==========

HPL_RConF_Routing

Description
Function to select the HPL Routing of the Audio Codec.

Arguments

```
*      left - value of the left channel configuration that is to be set for the
*             HPL routing register in order to select the desired routing to
*             HPL
```

Return Value CSL_Status

- CSL_SOK - Selecting the desired HPL routing successful
- CSL_ESYS_FAIL - Selecting the desired HPL routing is not successful

Select Page 1

3.1.2.9 int AudioClass::HPR_RConF_Routing (int *left,* int *right*)

==========

HPR_RConF_Routing

Description
Function to select the HPR Routing of the Audio Codec.

Arguments

```
*      left - value of the left channel configuration that is to be set for the
*             HPR routing register in order to select the desired routing to
*             HPR
*      right - value of the right channel configuration that is to be set for
*              the HPR routing register in order to select the desired routing
*              to HPR
```

Return Value CSL_Status

- CSL_SOK - Selecting the desired HPR routing successful
- CSL_ESYS_FAIL - Selecting the desired HPR routing is not successful

Select Page 1

3.1.2.10 void AudioClass::isrDma (void)

==========

isrDma

Description
DMA ISR for DMA interrupts.

Arguments

Return Value
None

Read the DMA interrupt

Clear DMA interrupt status

Reset interrupt value stored

Configure DMA for Read left channel

Configure DMA for Read right channel

Configure DMA for Write left channel

Configure DMA for Write right channel

3.1.2.11 int AudioClass::LOL_RConF_Routing (int *left,* int *right*)

```
*****************************************************************************
```

LOL_RConF_Routing

Description
Function to select the LOL Routing of the Audio Codec.

Arguments

```
*      left - value of the left channel configuration that is to be set for the
*             LOL routing register in order to select the desired routing to
*             LOL
*      right - value of the right channel configuration that is to be set for
*              the LOL routing register in order to select the desired routing
*              to LOL
```

Return Value CSL_Status

- CSL_SOK - Selecting the desired LOL routing successful
- CSL_ESYS_FAIL - Selecting the desired LOL routing is not successful

Select Page 1

3.1.2.12 int AudioClass::LOR_RConF_Routing (int *right*)

```
*****************************************************************************
```

LOR_RConF_Routing

Description
Function to select the LOR Routing of the Audio Codec.

Arguments

```
*      right - value of the right channel configuration that is to be set for
*              the LOR routing register in order to select the desired routing
*              to LOR
```

Return Value CSL_Status

- CSL_SOK - Selecting the desired LOR routing successful
- CSL_ESYS_FAIL - Selecting the desired LOR routing is not successful

Select Page 1

3.1.2.13 int AudioClass::read (void)

```
*****************************************************************************
```

read

Description
Function to perform DMA read of left and right channels.

Arguments

Return Value
None

3.1.2.14 int AudioClass::setInputGain (int *lgain,* int *rgain*)

==========
setInputGain

Description
Function to set ADC gain for left and right channels independently.

The ADC gain value will be in the range of 0.0dB to 47.5dB, a increase in 1% ADC gain (left/right) will result in increasing the ADC gain (left/right) by 0.5dB or 1dB

Arguments

```
*       lgain - ADC gain value for left channel in terms of percentage(0 - 100)
*       rgain - ADC gain value for right channel in terms of percentage(0 - 100)
```

Return Value CSL_Status

- CSL_SOK - Gain setting successful
- CSL_ESYS_FAIL - Gain setting is not successful

ADC gain register values will be in the range of 0 to 95(i.e. 0.0dB to 47.5dB respectively), hence multiply the lgain and rgain percentage values (range 0-100) by 0.95 to get the corresponding left and right ADC gain register values

Select Page 1

Write the gain registers

Small delay to allow the gain change

3.1.2.15 int AudioClass::setOutputVolume (int *volume*)

==========
setOutputVolume

Description
Function to set DAC volume with left channel controlling both left and right.

The DAC volume value will be in the range of : -63.5dB, -63.0dB, ..., -0.5dB, 0.0dB, +0.5dB, ..., +23.5dB, +24dB. A increase in 1% Volume (left/right) will result in increasing the DAC Volume (left/right) by 1.5dB or 2dB

Arguments

```
*       volume - DAC volume value in terms of percentage(0 - 100)
```

Return Value CSL_Status

- CSL_SOK - Volume setting successful
- CSL_ESYS_FAIL - Volume setting is not successful

DAC volume register values will be in the range of 0 to 48(for 0.0dB to +24dB respectively) and 255 down to 129 (for -0.5dB to -63.5dB respectively). Hence the percentages will be in the range of 73% to 100% for 0.0dB to +24dB respectively and 0% to 72% for -0.5dB to -63.5dB respectively. The range of DAC volume (left or right) register values consists of a total 176 values (i.e, 0-48: 49 nos and 129-255: 127 nos), hence divide the percentage with 0.568(i.e, 100/176 = 0.568).

Select Page 0

Left channel controlling both channels

Small delay to allow the gain change

3.1.2.16 int AudioClass::setOutputVolume (int *lvolume,* int *rvolume*)

==========

setOutputVolume

Description
Function to set DAC volume with left and right independent control.

Arguments

```
*       lvolume - DAC volume value for left channel
*       rvolume - DAC volume value for right channel
```

Return Value CSL_Status

- CSL_SOK - Volume setting successful
- CSL_ESYS_FAIL - Volume setting is not successful

DAC volume register values will be in the range of 0 to 48(for 0.0dB to +24dB respectively) and 255 down to 129 (for -0.5dB to -63.5dB respectively). Hence the percentages will be in the range of 73% to 100% for 0.0dB to +24dB respectively and 0% to 72% for -0.5dB to -63.5dB respectively. The range of DAC volume (left or right) register values consists of a total 176 values (i.e, 0-48: 49 nos and 129-255: 127 nos), hence divide the percentage with 0.568(i.e, 100/176 = 0.568).

Select Page 0

Left and right channels have independent control

Small delay to allow the gain change

3.1.2.17 int AudioClass::setSamplingRate (long *samplingRate*)

==========

setSamplingRate

Description
Function to set Sampling Rate for Audio IN/OUT.

Arguments

```
*       samplingRate - Sampling Rate that is to be set
```

Return Value CSL_Status

- CSL_SOK - Setting Sampling rate successful
- CSL_ESYS_FAIL - Setting Sampling rate is not successful

3.1.2.18 int AudioClass::write (void)

==========

write

Description
Function to perform DMA writes of left and right channels.

Arguments

Return Value
None

3.1.3 Member Data Documentation

3.1.3.1 unsigned short AudioClass::activeInBuf

Active input buffer

3.1.3.2 unsigned short AudioClass::activeOutBuf

Active output buffer

3.1.3.3 Uint16* AudioClass::audioInLeft[2]

Audio input - left channel

3.1.3.4 Uint16* AudioClass::audioInRight[2]

Audio input - right channel

3.1.3.5 Uint16* AudioClass::audioOutLeft[2]

Audio output - left channel

3.1.3.6 Uint16* AudioClass::audioOutRight[2]

Audio output - right channel

3.1.3.7 int AudioClass::sampleLeft

Left sample

3.1.3.8 int AudioClass::sampleRight

Right sample

The documentation for this class was generated from the following files:

- C:/DSPShieldDoc/Audio/Audio.h
- C:/DSPShieldDoc/Audio/Audio.cpp

3.2 BR_struct Struct Reference

`#include <chk_mmc.h>`

Public Attributes

- unsigned int short_jump_instr_byte_1:8
- unsigned int short_jump_instr_byte_2:8
- unsigned int short_jump_instr_byte_3:8
- unsigned int oem_name_byte_1:8
- unsigned int oem_name_byte_2:8
- unsigned int oem_name_byte_3:8
- unsigned int oem_name_byte_4:8
- unsigned int oem_name_byte_5:8
- unsigned int oem_name_byte_6:8
- unsigned int oem_name_byte_7:8
- unsigned int oem_name_byte_8:8
- unsigned int LB_bytes_per_sector:8
- unsigned int UB_bytes_per_sector:8
- unsigned int sectors_per_cluster:8
- unsigned int LB_reserved_sectors:8
- unsigned int UB_reserved_sectors:8
- unsigned int no_of_fats:8
- unsigned int LB_no_of_root_dir_entries:8
- unsigned int UB_no_of_root_dir_entries:8
- unsigned int LB_no_of_sectors_on_partition:8
- unsigned int UB_no_of_sectors_on_partition:8
- unsigned int media_descriptor:8
- unsigned int LB_sectors_per_fat:8
- unsigned int UB_sectors_per_fat:8
- unsigned int LB_sectors_per_track:8
- unsigned int UB_sectors_per_track:8
- unsigned int LB_no_of_heads:8
- unsigned int UB_no_of_heads:8
- unsigned int byte1_no_hidden_sectors:8
- unsigned int byte2_no_hidden_sectors:8
- unsigned int byte3_no_hidden_sectors:8
- unsigned int byte4_no_hidden_sectors:8
- unsigned int byte1_extended_no_of_sectors_on_partition:8
- unsigned int byte2_extended_no_of_sectors_on_partition:8
- unsigned int byte3_extended_no_of_sectors_on_partition:8
- unsigned int byte4_extended_no_of_sectors_on_partition:8
- unsigned int drive_number:8
- unsigned int reserved:8
- unsigned int extended_boot_signature:8
- unsigned int vol_serial_number_byte_1:8
- unsigned int vol_serial_number_byte_2:8
- unsigned int vol_serial_number_byte_3:8
- unsigned int vol_serial_number_byte_4:8
- unsigned int byte1_vol_label:8
- unsigned int byte2_vol_label:8
- unsigned int byte3_vol_label:8
- unsigned int byte4_vol_label:8
- unsigned int byte5_vol_label:8

- unsigned int byte6_vol_label:8
- unsigned int byte7_vol_label:8
- unsigned int byte8_vol_label:8
- unsigned int byte9_vol_label:8
- unsigned int byte10_vol_label:8
- unsigned int byte11_vol_label:8
- unsigned int byte1_fs_type:8
- unsigned int byte2_fs_type:8
- unsigned int byte3_fs_type:8
- unsigned int byte4_fs_type:8
- unsigned int byte5_fs_type:8
- unsigned int byte6_fs_type:8
- unsigned int byte7_fs_type:8
- unsigned int byte8_fs_type:8
- unsigned int opt_partition_boot_code [224]
- unsigned int signature

3.2.1 Detailed Description

Structure to hold Boot Record (BR)

3.2.2 Member Data Documentation

3.2.2.1 unsigned int BR_struct::byte10_vol_label

Volume label byte 10

3.2.2.2 unsigned int BR_struct::byte11_vol_label

Volume label byte 11

3.2.2.3 unsigned int BR_struct::byte1_extended_no_of_sectors_on_partition

Extended no of sectors byte 1

3.2.2.4 unsigned int BR_struct::byte1_fs_type

FS type byte 1

3.2.2.5 unsigned int BR_struct::byte1_no_hidden_sectors

Number of hidden sectors byte 1

3.2.2.6 unsigned int BR_struct::byte1_vol_label

Volume label byte 1

3.2.2.7 unsigned int BR_struct::byte2_extended_no_of_sectors_on_partition

Extended no of sectors byte 2

3.2.2.8 unsigned int BR_struct::byte2_fs_type

FS type byte 2

3.2.2.9 unsigned int BR_struct::byte2_no_hidden_sectors

Number of hidden sectors byte 2

3.2.2.10 unsigned int BR_struct::byte2_vol_label

Volume label byte 2

3.2.2.11 unsigned int BR_struct::byte3_extended_no_of_sectors_on_partition

Extended no of sectors byte 3

3.2.2.12 unsigned int BR_struct::byte3_fs_type

FS type byte 3

3.2.2.13 unsigned int BR_struct::byte3_no_hidden_sectors

Number of hidden sectors byte 3

3.2.2.14 unsigned int BR_struct::byte3_vol_label

Volume label byte 3

3.2.2.15 unsigned int BR_struct::byte4_extended_no_of_sectors_on_partition

Extended no of sectors byte 4

3.2.2.16 unsigned int BR_struct::byte4_fs_type

FS type byte 4

3.2.2.17 unsigned int BR_struct::byte4_no_hidden_sectors

Number of hidden sectors byte 4

3.2.2.18 unsigned int BR_struct::byte4_vol_label

Volume label byte 4

3.2.2.19 unsigned int BR_struct::byte5_fs_type

FS type byte 5

3.2.2.20 unsigned int BR_struct::byte5_vol_label

Volume label byte 5

3.2.2.21 unsigned int BR_struct::byte6_fs_type

FS type byte 6

3.2.2.22 unsigned int BR_struct::byte6_vol_label

Volume label byte 6

3.2.2.23 unsigned int BR_struct::byte7_fs_type

FS type byte 7

3.2.2.24 unsigned int BR_struct::byte7_vol_label

Volume label byte 7

3.2.2.25 unsigned int BR_struct::byte8_fs_type

FS type byte 8

3.2.2.26 unsigned int BR_struct::byte8_vol_label

Volume label byte 8

3.2.2.27 unsigned int BR_struct::byte9_vol_label

Volume label byte 9

3.2.2.28 unsigned int BR_struct::drive_number

Drive number

3.2.2.29 unsigned int BR_struct::extended_boot_signature

Extended boot signature

3.2.2.30 unsigned int BR_struct::LB_bytes_per_sector

Bytes per sector lower byte

3.2.2.31 unsigned int BR_struct::LB_no_of_heads

Number of heads lower byte

3.2.2.32 unsigned int BR_struct::LB_no_of_root_dir_entries

Number of root directories lower byte

3.2.2.33 unsigned int BR_struct::LB_no_of_sectors_on_partition

Sectors in partition lower byte

3.2.2.34 unsigned int BR_struct::LB_reserved_sectors

Reserved sectors lower byte

3.2.2.35 unsigned int BR_struct::LB_sectors_per_fat

Sectors per FAT lower byte

3.2.2.36 unsigned int BR_struct::LB_sectors_per_track

Sectors per track lower byte

3.2.2.37 unsigned int BR_struct::media_descriptor

Media descriptor

3.2.2.38 unsigned int BR_struct::no_of_fats

Number of fats

3.2.2.39 unsigned int BR_struct::oem_name_byte_1

OEM name byte 1

3.2.2.40 unsigned int BR_struct::oem_name_byte_2

OEM name byte 2

3.2.2.41 unsigned int BR_struct::oem_name_byte_3

OEM name byte 3

3.2.2.42 unsigned int BR_struct::oem_name_byte_4

OEM name byte 4

3.2.2.43 unsigned int BR_struct::oem_name_byte_5

OEM name byte 5

3.2.2.44 unsigned int BR_struct::oem_name_byte_6

OEM name byte 6

3.2.2.45 unsigned int BR_struct::oem_name_byte_7

OEM name byte 7

3.2.2.46 unsigned int BR_struct::oem_name_byte_8

OEM name byte 8

3.2.2.47 unsigned int BR_struct::opt_partition_boot_code[224]

Partition boot code

3.2.2.48 unsigned int BR_struct::reserved

Reserved sectors

3.2.2.49 unsigned int BR_struct::sectors_per_cluster

Sectors per cluster

3.2.2.50 unsigned int BR_struct::short_jump_instr_byte_1

Short jump instruction byte 1

3.2.2.51 unsigned int BR_struct::short_jump_instr_byte_2

Short jump instruction byte 2

3.2.2.52 unsigned int BR_struct::short_jump_instr_byte_3

Short jump instruction byte 3

3.2.2.53 unsigned int BR_struct::signature

Signature

3.2.2.54 unsigned int BR_struct::UB_bytes_per_sector

Bytes per sector upper byte

3.2.2.55 unsigned int BR_struct::UB_no_of_heads

Number of heads upper byte

3.2.2.56 unsigned int BR_struct::UB_no_of_root_dir_entries

Number of root directories upper byte

3.2.2.57 unsigned int BR_struct::UB_no_of_sectors_on_partition

Sectors in partition upper byte

3.2.2.58 unsigned int BR_struct::UB_reserved_sectors

Reserved sectors upper byte

3.2.2.59 unsigned int BR_struct::UB_sectors_per_fat

Sectors per FAT upper byte

3.2.2.60 unsigned int BR_struct::UB_sectors_per_track

Sectors per track upper byte

3.2.2.61 unsigned int BR_struct::vol_serial_number_byte_1

Volume serial number byte 1

3.2.2.62 unsigned int BR_struct::vol_serial_number_byte_2

Volume serial number byte 2

3.2.2.63 unsigned int BR_struct::vol_serial_number_byte_3

Volume serial number byte 3

3.2.2.64 unsigned int BR_struct::vol_serial_number_byte_4

Volume serial number byte 4

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

- C:/DSPShieldDoc/SD/chk_mmc.h

3.3 FFTClass Class Reference

FFT Class.

`#include <FFT.h>`

Public Member Functions

- void FFT_init (void)
- void FFT_filter (short *input, short *output, short *overlap, int length)

3.3.1 Detailed Description

FFT Class.

Contains prototypes for functions in FFT library

3.3.2 Member Function Documentation

3.3.2.1 void FFTClass::FFT_filter (short * *input,* short * *output,* short * *overlap,* int *length*)

==========

FFT_filter()

Description
API to perform FFT-filter(lpf)-IFFT on the samples passed as input

Arguments

```
input   - Buffer containing the Input samples
output  - Buffer to hold the FFT-filtered output samples
overlap - Buffer to hold the overlap data, which will be used in next
          FFT-filtering
length  - Number of input data samples
```

Return Value
None

pointer to keep track of input data, and output data

number of frames to be processed

Preparing the data for the FFT function. The real and imaginary parts of the signal are alternated. Since the input signal is real, the imaginary part is zero. Moreover since FFTLEN = 2*WINLEN, zero padding the input buffer.

real part

imag part

zero padding

real part

imag part

Taking the FFT of the signal. Using the scaled version of the FFT so that FFT values do not overflow. Bit-reversing the output.

Filtering the data. Filter the data in the frequency domain is equal to a complex multiplication.

Need to perform only filtering on half the spectrum, the other half of the spectrum will be generated as the complex conjugate of the first half. The FFT spectrum of a real signal satisfies the following symmetry: X(k) = complex conjugate of X(N-k)., where N = FFTLEN, and k is the frequency index. If N is even, X(0) & X(N/2) are real valued.

Need pointer to only real values

Splitting the 32-bit result into the real parts and imaginary parts.

The 0-th index and the FFTLEN/2 index are only real values. 2*FFTLEN since the real and imaginary parts of the complex values are alternated.

Downscale data to avoid overflow/saturation

Taking the inverse FFT. Using the NOSCALE version to get the original scale of the signal back.

Reconstructing the output using overlap and add technique.

real part

preserving the second half of the buffer for over lap and add.

real part

imag part

3.3.2.2 void FFTClass::FFT_init (void)

==========

FFT_init()

Description
API to initialize the buffers used by the FFT algorithm

Arguments

Return Value
None

Initialize output buffer with zeroes

Converting the filter impulse response to its frequency response. FFT - NOSCALE, to maintain the dynamic range and resolution.

The documentation for this class was generated from the following files:

- C:/DSPShieldDoc/FFT/FFT.h
- C:/DSPShieldDoc/FFT/FFT.cpp

3.4 File Class Reference

File Class.

`#include <SD.h>`

Public Member Functions

- operator bool () const
- File ()
  File fileClass.
- File (const File &fileObj)
  File fileClass.
- unsigned long available ()
  available()
- void close ()
  close()
- void flush ()
  flush()
- char peek ()
  peek()
- unsigned long position ()
  position()
- int print (char character)
  print(character)
- int print (char *printString)
  print(printString)
- int print (int integer)

print(integer)

- int print (long integer)

  print(integer)
- int print (float value)

  print(value)
- int print (double value)

  print(value)
- int print (long integer, NUMBER_FORMAT_BASE base)

  print(integer, base)
- int println (void)

  println()
- int println (char character)

  println(character)
- int println (char *printString)

  println(printString)
- int println (int integer)

  println(integer)
- int println (long integer)

  println(integer)
- int println (float value)

  print(value)
- int println (double value)

  print(value)
- int println (long integer, NUMBER_FORMAT_BASE base)

  println(integer, base)
- Bool seek (unsigned long posValue)

  seek(posValue)
- unsigned long size ()

  size()
- char read ()

  read()
- int read (char *buffer, int length)

  read(buffer, length)
- int read (int *buffer, int length)

  read(buffer, length)
- int write (char character)

  write(character)
- int write (long integer)

  write(integer)
- int write (char *printString)

  write(printString)
- int write (char *buffer, int length)

  write(buffer, length)
- int write (int *buffer, int length)

  write(buffer, length)
- Bool isDirectory ()

  isDirectory()
- File openNextFile ()

  openNextFile()
- void rewindDirectory ()

  rewindDirectory()
- void getName (char *fileName)

  getName(fileName)

Friends

- class SD_Class

3.4.1 Detailed Description

File Class.

The File class allows for reading from and writing to individual files on the SD card

3.4.2 Constructor & Destructor Documentation

3.4.2.1 File::File ( )

File fileClass.

File Class Constructor.

Returns

NONE

3.4.2.2 File::File (const File & *fileObj*)

File fileClass.

File Class Copy Constructor.

Parameters

fileObj	[IN] An existing File Object

Returns

NONE

3.4.3 Member Function Documentation

3.4.3.1 unsigned long File::available ( )

available()

copy constructor

API to Check if there are any bytes available for reading from the file

Returns

Number of bytes available, that can be read

Bytes available to read is: File size minus current file seek position

3.4.3.2 void File::close (void)

close()

API to Close an opened file

Returns

NONE

Flush any data which is pending to be written

Delete the file node from the list of opened file nodes

3.4.3.3 void File::flush ( )

flush()

API to ensure that any bytes written to the file are physically saved to the SD card

Returns

NONE

Setting Size as current File Pointer position, it will be valid when User seeks to some position (other than EOF) of the file

Since we are writing only 1 byte, decrement file size by 1. ATA_write with a single data element will increment the file size by 2bytes

3.4.3.4 void File::getName (char * *fileName*)

getName(fileName)

API to get the name of the File

Parameters

fileName	[IN] Buffer to hold the name of the file

Returns

NONE

3.4.3.5 Bool File::isDirectory ( )

isDirectory()

API to report if the current file is a directory or not

Returns

TRUE - If the current file is a Directory FALSE - If the current file is not a Directory

3.4.3.6 File File::openNextFile ( )

openNextFile()

API to report the next file or folder in a directory

Returns

Pointer to the 'File' class of the next file

Initializing the Next File Handle

Removing spaces are the end of file name

Since its a directory, go inside the directory and find the 1st file under it, other than '.' and '..' files

Since its file append the file extension to the file name

Removing spaces are the end of file extension

Find the next file under the Parent directory, which triggered the current 'openNextFile' API call

Next file not found

3.4.3.7 File::operator bool ( ) const [inline]

To check file open status

3.4.3.8 char File::peek ( )

peek()

API to Read a byte from the file without advancing to the next one

Returns

The character that is read from the file

Obtain the current seek position

Seek to the current seek position obtained previously

Return the data that was read from previous User Read Operation

3.4.3.9 unsigned long File::position ( )

position()

API to Get the current position to which the next byte will be read from or written to

Returns

The location to which the next byte will be read from or written to

Get the current file seek position

3.4.3.10 int File::print (char *character*)

print(character)

API to Print an ASCII character to the file

Parameters

character	[IN] The ASCII character to be printed to the file

Returns

The number of bytes printed to the file Else zero, if the file is not opened or for invalid file operation

Setting Size as current File Pointer position, it will be valid when User seeks to some position (other than EOF) of the file

Check whether data is already there to be flushed or not

Store the character in the flush data variable

3.4.3.11 int File::print (char * *printString*)

print(printString)

API to Print a String(character array) to the file

Parameters

printString	[IN] The String to be printed to the file

Returns

The number of bytes printed to the file Else zero, if the file is not opened or for invalid file operation

Check whether data is already there to be flushed

Setting Size as current File Pointer position, it will be valid when User seeks to some position (other than EOF) of the file

Pack the bytes in a 16-bit integer Array

If the user requests odd no of bytes to write, hold the last byte in the 'flushData' variable

3.4.3.12 int File::print (int *integer*)

print(integer)

API to Print a 16 Bit integer(int) to the file

Parameters

integer	[IN] The 16 Bit integer(int) to be printed to the file

Returns

The number of bytes printed to the file Else zero, if the file is not opened or for invalid file operation

3.4.3.13 int File::print (long *integer*)

print(integer)

API to Print a 32 Bit integer(long) to the file

Parameters

integer	[IN] The 32 Bit integer(long) to be printed to the file

Returns

The number of bytes printed to the file Else zero, if the file is not opened or for invalid file operation

Convert the integer to character array

3.4.3.14 int File::print (float *value*)

print(value)

API to Print a Floating Point Value to the file as human-readable ASCII text

Parameters

value	[IN] Floating Point value that is to be written to the file

Returns

The number of bytes printed to the file Else zero, if the file is not opened or for invalid file operation

Convert the float to character array

3.4.3.15 int File::print (double *value*)

print(value)

API to Print a Double Precision Floating Point Value to the file as human-readable ASCII text

Parameters

value	[IN] Double Precision Floating Point value that is to be written to the file

Returns

The number of bytes printed to the file Else zero, if the file is not opened or for invalid file operation

Convert the float to character array

3.4.3.16 int File::print (long *integer,* NUMBER_FORMAT_BASE *base*)

print(integer, base)

API to Print an integer in a specified base format to the file

Parameters

integer	[IN] The integer to be printed to the file
base	[IN] The base in which to print numbers

Returns

The number of bytes printed to the file Else zero, if the file is not opened or for invalid file operation

3.4.3.17 int File::println (void)

println()

API to Print a carriage return and newline to the file

Returns

The number of bytes printed to the file Else zero, if the file is not opened or for invalid file operation

3.4.3.18 int File::println (char *character*)

println(character)

API to Print an ASCII character, followed by a carriage return and newline to the file

Parameters

character	[IN] The ASCII character to be printed to the file

Returns

The number of bytes printed to the file Else zero, if the file is not opened or for invalid file operation

3.4.3.19 int File::println (char * *printString*)

println(printString)

API to Print a String(character array), followed by a carriage return and newline to the file

Parameters

printString	[IN] The String to be printed to the file

Returns

The number of bytes printed to the file Else zero, if the file is not opened or for invalid file operation

3.4.3.20 int File::println (int *integer*)

println(integer)

API to Print an integer, followed by a carriage return and newline to the file

Parameters

integer	[IN] The integer to be printed to the file

Returns

The number of bytes printed to the file Else zero, if the file is not opened or for invalid file operation

3.4.3.21 int File::println (long *integer*)

println(integer)

API to Print an integer, followed by a carriage return and newline to the file

Parameters

integer	[IN] The integer to be printed to the file

Returns

The number of bytes printed to the file Else zero, if the file is not opened or for invalid file operation

3.4.3.22 int File::println (float *value*)

print(value)

API to Print a Floating Point Value to the file as human-readable ASCII text, followed by a carriage return and newline to the file

Parameters

value	[IN] Floating Point value that is to be written to the file

Returns

The number of bytes printed to the file Else zero, if the file is not opened or for invalid file operation

3.4.3.23 int File::println (double *value*)

print(value)

API to Print a Double Precision Floating Point Value to the file as human-readable ASCII text, followed by a carriage return and newline to the file

Parameters

value	[IN] Double Precision Floating Point value that is to be written to the file

Returns

The number of bytes printed to the file Else zero, if the file is not opened or for invalid file operation

3.4.3.24 int File::println (long *integer,* NUMBER_FORMAT_BASE *base*)

println(integer, base)

API to Print an integer in a specified base format, followed by a carriage return and newline to the file

Parameters

integer	[IN] The integer to be printed to the file
base	[IN] The base in which to print numbers

Returns

The number of bytes printed to the file Else zero, if the file is not opened or for invalid file operation

3.4.3.25 char File::read (void)

read()

API to Read a byte from the file

Returns

The next byte (or character) -1 if no data is available

ATA_read() will return 2 bytes as a single data element, so check whether any data is already read in previous calls to ATA_read(which will be stored in holdCharFromRead variable)

3.4.3.26 int File::read (char * *buffer,* int *length*)

read(buffer, length)

API to Read an array of bytes from the file

Parameters

buffer	[IN] The Buffer to hold the data read from the file
length	[IN] The length of the data to be read, in no of bytes

Returns

The number of bytes read from the file Else zero, if the file is not opened

Check whether any data is already read in previous calls(which will be stored in holdCharFromRead variable)

Incrementing the buffer pointer, so that the data in 'holdCharFromRead' variable will be stored later

Revert back to previous read state

Unpacking the 16-bit Integer to individual bytes

If length is an odd number(No of bytes to read is odd)

3.4.3.27 int File::read (int * *buffer,* int *length*)

read(buffer, length)

API to Read an array of Words (16 bits) from the file

Parameters

buffer	[IN] The Buffer to hold the data read from the file
length	[IN] The length of the data to be read, in no of Words (16 bits)

Returns

The number of Words read from the file Else zero, if the file is not opened

Check whether any data is already read in previous calls(which will be stored in holdCharFromRead variable)

Incrementing the buffer pointer, so that the data in 'holdCharFromRead' variable will be stored later

Revert back to previous read state

3.4.3.28 void File::rewindDirectory ( )

rewindDirectory()

API to go back to the first file in the directory

Returns

NONE

3.4.3.29 Bool File::seek (unsigned long *posValue*)

seek(posValue)

API to Seek to a new position in the file, which must be between 0 and the size of the file

Parameters

posValue	[IN] The position to which to seek

Returns

TRUE - Successful in seeking to the requested position FALSE - Unsuccessful in seeking to the requested position

Flush any remaining data

If request is to Seek odd no of bytes

Read the next word and hold the LSB for next read operation, also hold the MSB for next write operation

3.4.3.30 unsigned long File::size ( )

size()

API to Get the size of the file

Returns

The size of the file in bytes

3.4.3.31 int File::write (char *character*)

write(character)

API to Write an ASCII character to the file

Parameters

character	[IN] The ASCII character to be written to the file

Returns

The number of bytes written to the file Else zero, if the file is not opened or for invalid file operation

3.4.3.32 int File::write (long *integer*)

write(integer)

API to Write an integer to the file

Parameters

integer	[IN] The integer to be written to the file

Returns

The number of bytes written to the file Else zero, if the file is not opened or for invalid file operation

3.4.3.33 int File::write (char * *printString*)

write(printString)

API to Write a String(character array) to the file

Parameters

printString	[IN] The String to be written to the file

Returns

The number of bytes written to the file Else zero, if the file is not opened or for invalid file operation

3.4.3.34 int File::write (char * *buffer,* int *length*)

write(buffer, length)

API to Write a String(character array) of specified length to the file

Parameters

buffer	[IN] The String to be written to the file
length	[IN] The length of the String to be written

Returns

The number of bytes written to the file Else zero, if the file is not opened or for invalid file operation

Setting Size as current File Pointer position, it will be valid when User seeks to some position (other than EOF) of the file

Writing data in chunks of 512 bytes

Write the left over bytes, which are lesser than 512 bytes

3.4.3.35 int File::write (int * *buffer,* int *length*)

write(buffer, length)

API to Write an Array of words, of specified length to the file

Parameters

buffer	[IN] The Buffer (containing array of words) to be written to the file
length	[IN] No of Words (16 bits) to be written

Returns

The number of Words written to the file Else zero, if the file is not opened or for invalid file operation

Setting Size as current File Pointer position, it will be valid when User seeks to some position (other than EOF) of the file

3.4.4 Friends And Related Function Documentation

3.4.4.1 friend class SD_Class [friend]

Making 'SD_Class' class friend of 'File' class, so that member functions of 'SD_Class' can access private member fields of 'File'

The documentation for this class was generated from the following files:

- C:/DSPShieldDoc/SD/SD.h
- C:/DSPShieldDoc/SD/SD.cpp

3.5 fileNodesList Class Reference

Node of the Linked List to hold the details of all the Opened files.

Public Attributes

- unsigned long startCluster
- struct fileNodesList * nextFileNode

3.5.1 Detailed Description

Node of the Linked List to hold the details of all the Opened files.

3.5.2 Member Data Documentation

3.5.2.1 struct fileNodesList* fileNodesList::nextFileNode

Pointer to next file

3.5.2.2 unsigned long fileNodesList::startCluster

Starting cluster of file

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

- C:/DSPShieldDoc/SD/SD.cpp

3.6 FONT_CHAR_INFO Struct Reference

This structure describes a single character's display information.

`#include <bitmapDb.h>`

Public Attributes

- const Uint8 widthBits
- const Uint8 heightBits
- const Uint16 offset

3.6.1 Detailed Description

This structure describes a single character's display information.

3.6.2 Member Data Documentation

3.6.2.1 const Uint8 FONT_CHAR_INFO::heightBits

height, in bits (or pixels), of the character

3.6.2.2 const Uint16 FONT_CHAR_INFO::offset

offset of the character's bitmap, in bytes, into the the FONT_INFO's data array

3.6.2.3 const Uint8 FONT_CHAR_INFO::widthBits

width, in bits (or pixels), of the character

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

- C:/DSPShieldDoc/OLED/bitmapDb.h

3.7 FONT_INFO Struct Reference

This structure describes a single font.

`#include <bitmapDb.h>`

Public Attributes

- const Uint8 heightPages
- const Uint8 startChar
- const Uint8 endChar
- const FONT_CHAR_INFO * charInfo

  const Uint8 spacePixels; /!< number of pixels that a space character takes up */*
- const Uint8 * data

3.7.1 Detailed Description

This structure describes a single font.

3.7.2 Member Data Documentation

3.7.2.1 const FONT_CHAR_INFO* FONT_INFO::charInfo

const Uint8 spacePixels; /*!< number of pixels that a space character takes up */

pointer to array of char information

3.7.2.2 const Uint8* FONT_INFO::data

pointer to generated array of character visual representation

3.7.2.3 const Uint8 FONT_INFO::endChar

the last character in the font

3.7.2.4 const Uint8 FONT_INFO::heightPages

height, in pages (8 pixels), of the font's characters

3.7.2.5 const Uint8 FONT_INFO::startChar

the first character in the font (e.g. in charInfo and data)

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

- C:/DSPShieldDoc/OLED/bitmapDb.h

3.8 GPT_Config Struct Reference

Configuration structure.

`#include <Timers.h>`

Public Attributes

- Bool autoLoad
- unsigned short preScaleDiv
- Bool ctrlTim
- unsigned short prdLow
- unsigned short prdHigh

3.8.1 Detailed Description

Configuration structure.

Contains the Information of a GPT configuration.

3.8.2 Member Data Documentation

3.8.2.1 Bool GPT_Config::autoLoad

Auto reload

3.8.2.2 Bool GPT_Config::ctrlTim

control timer

3.8.2.3 unsigned short GPT_Config::prdHigh

period high

3.8.2.4 unsigned short GPT_Config::prdLow

period low

3.8.2.5 unsigned short GPT_Config::preScaleDiv

Prescale division

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

- C:/DSPShieldDoc/Timers/Timers.h

3.9 MBR_struct Struct Reference

`#include <chk_mmc.h>`

Public Attributes

- unsigned int mbr_code [216]
- unsigned int mmc_id_entry [7]
- PARTITION_TABLE partition_one
- PARTITION_TABLE partition_two
- PARTITION_TABLE partition_three
- PARTITION_TABLE partition_four
- unsigned int signature

3.9.1 Detailed Description

Structure to hold Master Boot Record (MBR)

3.9.2 Member Data Documentation

3.9.2.1 unsigned int MBR_struct::mbr_code[216]

MBR code 432 bytes

3.9.2.2 unsigned int MBR_struct::mmc_id_entry[7]

MMC ID entry 14 bytes

3.9.2.3 PARTITION_TABLE MBR_struct::partition_four

Partition 4 entry 16 bytes

3.9.2.4 PARTITION_TABLE MBR_struct::partition_one

Partition 1 entry 16 bytes

3.9.2.5 PARTITION_TABLE MBR_struct::partition_three

Partition 3 entry 16 bytes

3.9.2.6 PARTITION_TABLE MBR_struct::partition_two

Partition 2 entry 16 bytes

3.9.2.7 unsigned int MBR_struct::signature

FS signature 2 bytes

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

- C:/DSPShieldDoc/SD/chk_mmc.h

3.10 OLED Class Reference

OLED Class.

`#include <OLED.h>`

Public Member Functions

- void oledInit (void)
- void init ()
- void begin ()
- void clear ()
- void clear (int)
- int printchar (unsigned char)
- int print (char string[])
- int print (long value)
- int print (char character)
- int print (long value, int base)
- void write (unsigned char)
- void noDisplay ()
- void display ()
- void scrollDisplayLeft ()
- void scrollDisplayLeft (int line)
- void scrollDisplayRight ()
- void scrollDisplayRight (int line)
- void flip ()
- void setOrientation (int newDir)
- void autoscroll ()
- void noAutoscroll ()
- void setline (int line)
- void setRolling (int row, int status)
- void resetCursor (int page)

3.10.1 Detailed Description

OLED Class.

Contains prototypes for functions in OLED library

3.10.2 Member Function Documentation

3.10.2.1 void OLED::autoscroll (void)

==========

autoscroll()

Description
Turns on automatic scrolling of the LCD

Arguments

Return Value
None

For multibyte commands

Set vertical and horizontal scrolling

Vertical and Right Horizontal Scroll 26 = left, 27 = right, 29 = vl, 2A = vr

Dummy byte

Define start page address

Set time interval between each scroll step as 5 frames

Define end page address

Vertical scrolling offset

Keep first 8 rows from vertical scrolling

Set Vertical Scroll Area

Set No. of rows in top fixed area

Set No. of rows in scroll area

3.10.2.2 void OLED::begin (void)

==========

begin()

Description
Specifies the dimensions (width and height) of the display.

Arguments

```
*       Uint8 cols      <- Number of columns
*       Uint8 rows      <- Number of rows
```

Return Value
None

3.10.2.3 void OLED::clear (void)

==========

clear()

Description
Clears the LCD screen and positions the cursor in the upper-left corner

Arguments

Return Value
None

This part is to initialize and blank the LCD.

Initialize OLED display

Deactivate Scrolling

Write to page 0

Set low column address

Set high column address

Set page for page 0 to page 5

Write to page 1

Set low column address

Set high column address

Set page for page 0 to page 5

Set cursor to upper left corner

Set low column address

Set high column address

Set page for page 0 to page 5

3.10.2.4 void OLED::clear (int *page*)

==========

clear(page)

Description
Clears the LCD screen and positions the cursor in the upper-left corner

Arguments

```
*       int page    <- Page number
```

Return Value
None

This part is to initialize and blank the LCD.

Initialize OLED display

Deactivate Scrolling

Write to page 0

Set low column address

Set high column address

Set page for page 0 to page 5

Write to page 1

Set low column address

Set high column address

Set page for page 0 to page 5

Write to page 0

Set low column address

Set high column address

Set page for page 0 to page 5

3.10.2.5 void OLED::display ( )

==========

display()

Description
Turns on the LCD display

Arguments

Return Value
None

Turn on oled panel

3.10.2.6 void OLED::flip ( )

==========

flip()

Description
Flips the screen vertically

Arguments

Return Value
None

Set divide ratios

Set divide ratios

3.10.2.7 void OLED::init (void)

==========

init()

Description
Function initializes I2C Module and also LCD.

Arguments

Return Value
None

Note: Wire initialization will be done in main.cpp

Initialize OSD9616 display

Set low column address

Set high column address

Turn off oled panel

Set display clock divide ratio/oscillator frequency

Set divide ratio

Set multiplex ratio(1 to 16)

Set display offset

Not offset

Set Display start line address

–set DC-DC enable

Set Charge Pump

Set segment re-map 95 to 0

Set COM Output Scan Direction

Set com pins hardware configuration

Set contrast control register

Set pre-charge period

Set vcomh

0.83∗vref

Set entire display on/off

Set normal display

Turn on oled panel

3.10.2.8 void OLED::noAutoscroll (void)

==========

autoscroll()

Description
Turns off automatic scrolling of the LCD

Arguments

Return Value
None

2 Spaces

Deactivate Scrolling

3.10.2.9 void OLED::noDisplay ( )

==========

noDisplay()

Description
Turns off the LCD display

Arguments

Return Value
None

Turn off oled panel

3.10.2.10 void OLED::oledInit (void)

==========

oledInit()

Description
Function initializes I2C Module and also LCD.

Arguments

Return Value
None

Font data for Century Gothic 8pt

Character bitmaps for Century Gothic 8pt

3.10.2.11 int OLED::print (char *string[]*)

==========

print(string[])

Description
Prints a string to the LCD

Arguments

```
*       char *printString      <- Pointer to the string to be printed
```

Return Value Returns the number of bytes written to LCD

- strLen - Length of the string written to LCD

3.10.2.12 int OLED::print (long *value*)

==========

print(value)

Description
Prints a value to the LCD

Arguments

```
*       long value      <- Value to be printed
```

Return Value Returns the number of bytes written to LCD

- strLen - Length of the string written to LCD

3.10.2.13 int OLED::print (char *character*)

==========

print(character)

Description
Prints a character to the LCD

Arguments

```
*       char character      <- Character to be printed
```

Return Value Returns the number of bytes written to LCD

- 1 - In case of char only 1byte is written to LCD

3.10.2.14 int OLED::print (long *value,* int *base*)

==========

print(value, base)

Description
Prints a value to the LCD

Arguments

```
*      long value      <- Value to be printed
*      int base        <- Base of the value printed
```

Return Value Returns the number of bytes written to LCD

- strLen - Length of the string written to LCD

3.10.2.15 int OLED::printchar (unsigned char *a*)

==========

printchar(character)

Description
Prints a character to the LCD

Arguments

```
*     unsigned char character      <- Character to be printed
```

Return Value
None

2 Spaces

Deactivate Scrolling

3.10.2.16 void OLED::resetCursor (int *line*)

==========

resetCursor(line)

Description
Resets the cursor position to the beginning in a given line

Arguments

```
*      int line    <- line number
```

Return Value
None

Write to line 0

Set low column address

Set high column address

Set page for page 0 to page 5

Write to line 1

Set low column address

Set high column address

Set page for page 0 to page 5

3.10.2.17 void OLED::scrollDisplayLeft (void)

==========

scrollDisplayLeft()

Description
Scrolls the contents of the display (text and cursor) to the left

Arguments

Return Value
None

For multibyte commands

These commands scroll the display without changing the RAM

Set vertical and horizontal scrolling

Vertical and Right Horizontal Scroll 26 = left, 27 = right, 29 = vl, 2A = vr

Dummy byte

Define start page address

Set time interval between each scroll step as 5 frames

Define end page address

Dummy byte

Dummy byte

3.10.2.18 void OLED::scrollDisplayLeft (int *line*)

==========

scrollDisplayLeft()

Description
Scrolls the contents of a particular line, of the display (text and cursor) to the left

Arguments

```
*     int line      <- Line number
```

Return Value
None

For multibyte commands

These commands scroll the display without changing the RAM

Set vertical and horizontal scrolling

Vertical and Right Horizontal Scroll 26 = left, 27 = right, 29 = vl, 2A = vr

Dummy byte

Define start page address

Set time interval between each scroll step as 5 frames

Define end page address

Dummy byte

Dummy byte

3.10.2.19 void OLED::scrollDisplayRight (void)

==========

scrollDisplayRight()

Description
Scrolls the contents of the display (text and cursor) to the right

Arguments

Return Value
None

For multibyte commands

Set horizontal scrolling

Vertical and Left Horizontal Scroll 26 = left, 27 = right, 29 = vl, 2A = vr

Dummy byte

Define start page address

Set time interval between each scroll step as 5 frames

Define end page address

Dummy byte

Dummy byte

3.10.2.20 void OLED::scrollDisplayRight (int *line*)

==========

scrollDisplayRight(line)

Description
Scrolls the contents of a particular line, of the display (text and cursor) to the right

Arguments

```
*     int line      <- Line number
```

Return Value
None

For multibyte commands

Set horizontal scrolling

Vertical and Left Horizontal Scroll 26 = left, 27 = right, 29 = vl, 2A = vr

Dummy byte

Define start page address

Set time interval between each scroll step as 5 frames

Define end page address

Dummy byte

Dummy byte

3.10.2.21 void OLED::setline (int *line*)

==========

setline(line)

Description
Sets the start line for the display
Arguments

```
*     int line      <- Line number
```

Return Value
None

Write to page 1

Set low column address

Set high column address

Set page for page 1

Write to page 0

Set low column address

Set high column address

Set page for page 0

3.10.2.22 void OLED::setOrientation (int *newDir*)

==========
setOrientation()

Description
Function to set the orientation of LCD.
Arguments

```
*     int newDir      <- Direction of orientation
```

Return Value
None

Set divide ratio

Set divide ratio

3.10.2.23 void OLED::setRolling (int *row,* int *status*)

==========
setRolling(int row, int status)

Description
Sets the rolling parameters
Arguments

```
*     int row       <- Row number
*     int status    <- Status of rolling
```

Return Value
None

3.10.2.24 void OLED::write (unsigned char *byte*)

==========
write(character)

Description
Writes a character to the LCD
Arguments

```
*       char character      <- Character to be printed
```

Return Value Returns the number of bytes written to LCD

- 1 - In case of char only 1 byte is written to LCD

Set low column address

Set high column address

Set page for page 0 to page 5

The documentation for this class was generated from the following files:

- C:/DSPShieldDoc/OLED/OLED.h
- C:/DSPShieldDoc/OLED/OLED.cpp

3.11 PARTITION_TABLE Struct Reference

```
#include <chk_mmc.h>
```

Public Attributes

- unsigned int boot_descriptor:8
- unsigned int partition_start_head:8
- unsigned int partition_start_sector:6
- unsigned int partition_start_cylinder:10
- unsigned int fs_descriptor:8
- unsigned int partition_end_head:8
- unsigned int partition_end_sector:6
- unsigned int partition_end_cylinder:10
- unsigned int byte1_first_sector_position:8
- unsigned int byte2_first_sector_position:8
- unsigned int byte3_first_sector_position:8
- unsigned int byte4_first_sector_position:8
- unsigned int byte1_no_of_sectors_in_partition:8
- unsigned int byte2_no_of_sectors_in_partition:8
- unsigned int byte3_no_of_sectors_in_partition:8
- unsigned int byte4_no_of_sectors_in_partition:8

3.11.1 Detailed Description

Structure to hold Partition table

3.11.2 Member Data Documentation

3.11.2.1 unsigned int PARTITION_TABLE::boot_descriptor

Boot descriptor 1 byte

3.11.2.2 unsigned int PARTITION_TABLE::byte1_first_sector_position

Byte 1 first sector position

3.11.2.3 unsigned int PARTITION_TABLE::byte1_no_of_sectors_in_partition

Byte 1 sectors in partition

3.11.2.4 unsigned int PARTITION_TABLE::byte2_first_sector_position

Byte 2 first sector position

3.11.2.5 unsigned int PARTITION_TABLE::byte2_no_of_sectors_in_partition

Byte 2 sectors in partition

3.11.2.6 unsigned int PARTITION_TABLE::byte3_first_sector_position

Byte 3 first sector position

3.11.2.7 unsigned int PARTITION_TABLE::byte3_no_of_sectors_in_partition

Byte 3 sectors in partition

3.11.2.8 unsigned int PARTITION_TABLE::byte4_first_sector_position

Byte 4 first sector position

3.11.2.9 unsigned int PARTITION_TABLE::byte4_no_of_sectors_in_partition

Byte 4 sectors in partition

3.11.2.10 unsigned int PARTITION_TABLE::fs_descriptor

FS descriptor 1 byte

3.11.2.11 unsigned int PARTITION_TABLE::partition_end_cylinder

Partition end cylinder 10 bits

3.11.2.12 unsigned int PARTITION_TABLE::partition_end_head

Partition end head 1 byte

3.11.2.13 unsigned int PARTITION_TABLE::partition_end_sector

Partition end sector 6 bits

3.11.2.14 unsigned int PARTITION_TABLE::partition_start_cylinder

Partition start cylinder 10 bits

3.11.2.15 unsigned int PARTITION_TABLE::partition_start_head

Partition start head 1 byte

3.11.2.16 unsigned int PARTITION_TABLE::partition_start_sector

Partition start sector 6 bits

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

- C:/DSPShieldDoc/SD/chk_mmc.h

3.12 Pipe_Config Class Reference

Pipe confuguration Class.

`#include <USB.h>`

Public Attributes

- USB_xferType xferType
- unsigned short maxPktSize

3.12.1 Detailed Description

Pipe confuguration Class.

3.12.2 Member Data Documentation

3.12.2.1 unsigned short Pipe_Config::maxPktSize

Max packet size

3.12.2.2 USB_xferType Pipe_Config::xferType

Transfer type

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

- C:/DSPShieldDoc/USB/USB.h

3.13 PLL_Class Class Reference

PLL Class.

`#include <pll.h>`

Public Member Functions

- PLL_Class ()
- int configure (PLL_Config *pConfigInfo)
- int getConfigure (PLL_Config *pConfigInfo)
- long getSystemClock ()

3.13.1 Detailed Description

PLL Class.

Contains prototypes for functions in PLL library

3.13.2 Constructor & Destructor Documentation

3.13.2.1 PLL_Class::PLL_Class ( )

PLL pllClass

PLL Class Constructor.

3.13.3 Member Function Documentation

3.13.3.1 int PLL_Class::configure (PLL_Config * *pConfigInfo*)

configure(&pllConfig)

Configure PLL PLL_Config *pConfigInfo <- Config Structure, used to configure PLL

3.13.3.2 int PLL_Class::getConfigure (PLL_Config * *pConfigInfo*)

getConfigure(&pllConfig)

Get PLL Configuration Values PLL_Config *pConfigInfo <- Config Structure, used to store the PLL configurations

3.13.3.3 long PLL_Class::getSystemClock ( )

getSystemClock()

Get System Clock frequency, it will return the CPU clock in terms of kHz

The documentation for this class was generated from the following files:

- C:/DSPShieldDoc/PLL/pll.h
- C:/DSPShieldDoc/PLL/pll.cpp

3.14 SAR_Class Class Reference

SAR Class.

`#include <SAR.h>`

Public Member Functions

- int begin ()
- int end ()
- int configChannel (int opmode, int chanNum, int conversion)
- int startConversion ()
- int stopConversion ()
- Bool getStatus ()
- unsigned int readData ()

3.14.1 Detailed Description

SAR Class.

Contains prototypes for functions in SAR library

3.14.2 Member Function Documentation

3.14.2.1 int SAR_Class::begin (void)

==========

begin()

Description
API to Initialize the SAR Module

Arguments

Return Value

- CSL_SOK - SAR init successful
- CSL_ESYS_INVPARAMS - Invalid parameters

3.14.2.2 int SAR_Class::configChannel (int *opmode,* int *chanNum,* int *conversion*)

==========

configChannel()

Description
API to Configure the SAR module

Arguments

```
opmode     - SAR Operation Mode
chanNum    - Channel Number to configure
conversion - Conversion mode used to configure the SAR
```

Return Value

- CSL_SOK - SAR Config successful
- CSL_ESYS_INVPARAMS - Invalid parameters

3.14.2.3 int SAR_Class::end ( )

==========

end()

Description
API to Close and De-initialize the SAR Module

Arguments

Return Value

- CSL_SOK - SAR close successful
- CSL_ESYS_INVPARAMS - Invalid parameters

3.14.2.4 Bool SAR_Class::getStatus ( )

==========

getStatus()

Description
API to read the status of the ADC conversion

Arguments

Return Value

- TRUE - If Conversion is Complete
- FALSE - If Conversion is Pending

3.14.2.5 unsigned int SAR_Class::readData ( )

==========

readData()

Description
API to read the digtal data converted by the SAR module

Arguments

Return Value

- The converted Digital data

3.14.2.6 int SAR_Class::startConversion ( )

==========

startConversion()

Description
API to Start the ADC conversion

Arguments

Return Value

- CSL_SOK - SAR Conversion Start successful
- CSL_ESYS_INVPARAMS - Invalid parameters

3.14.2.7 int SAR_Class::stopConversion ( )

==========

stopConversion()

Description
API to Stop the ADC conversion

Arguments

Return Value

- CSL_SOK - SAR Conversion Stop successful
- CSL_ESYS_INVPARAMS - Invalid parameters

The documentation for this class was generated from the following files:

- C:/DSPShieldDoc/SAR/SAR.h
- C:/DSPShieldDoc/SAR/SAR.cpp

3.15 SD_Class Class Reference

SD Card Class.

`#include <SD.h>`

Public Member Functions

- SD_Class ()
  SD_Class sdClass.
- ~SD_Class ()
  delete (sdClass)
- Bool begin ()
  begin()
- Bool begin (int opmode)
  begin()
- Bool exists (char *fileName)
  exists(filePath)
- Bool mkdir (char *fileName)
  mkdir(directoryPath)
- File open (char *filePath)
  open(filePath)
- File open (char *filePath, FILE_MODE mode)

open(filePath, mode)

- Bool remove (char *fileName)

  remove(filePath)

- Bool rmdir (char *fileName)

  rmdir(directoryPath)

3.15.1 Detailed Description

SD Card Class.

The SD class provides functions for accessing the SD card and manipulating its files and directories

3.15.2 Constructor & Destructor Documentation

3.15.2.1 SD_Class::SD_Class ( )

SD_Class sdClass.

SD Class Constructor.

Returns

NONE

3.15.2.2 SD_Class::~SD_Class ( )

delete (sdClass)

SD Class Destructor.

Returns

NONE

De-allocate all the memory that was previous allocated for the File objects of all the opened files

3.15.3 Member Function Documentation

3.15.3.1 Bool SD_Class::begin (void)

begin()

API to Initialize and Configure the SD Card

Returns

TRUE - Successful in initializing and configuring the SD Card FALSE - Unsuccessful in initializing and configuring the SD Card

3.15.3.2 Bool SD_Class::begin (int *opmode*)

begin()

API to Initialize and Configure the SD Card

Returns

TRUE - Successful in initializing and configuring the SD Card FALSE - Unsuccessful in initializing and configuring the SD Card

Get the clock divider value for the current CPU frequency

Initialize MMCSD CSL module

Select the Pin Muxing mode, to enable MMC/SD Module

Open MMCSD CSL module

Configure the DMA in case of operating mode is set to DMA

Initialize Dma

Open Dma channel for MMCSD write

Open Dma channel for MMCSD read

Set the DMA handle for MMC read

Reset the SD card

Check for the card

Verify whether the SD card is detected or not

Check if the card is high capacity card

Set the init clock

Send the card identification Data

Set the Relative Card Address

Read the SD Card Specific Data

Set clock for read-write access

Set Endian mode for read and write operations

Set block length for the memory card For high capacity cards setting the block length will have no effect

Call init function initialize ATA state structure

Set the temp write buffer

Find the first file available

3.15.3.3 Bool SD_Class::exists (char * *filePath*)

exists(filePath)

API to check whether a particular file exists in the ATA File System or not

Parameters

filePath	[IN] Path of the File to be checked

Returns

TRUE - If the file exists FALSE - If the file doesn't exists

Navigate to the Directory path and check whether the intermediate directories (if any) in the path are present

Check whether the file exists on the card or not

3.15.3.4 Bool SD_Class::mkdir (char * *directoryPath*)

mkdir(directoryPath)

API to create a new directory

Parameters

directoryPath	[IN] Path of the Directory that is to be created

Returns

TRUE - Successful in creating the requested directory FALSE - Unsuccessful in creating the requested directory

The length of the directory path is restricted to 255 charcaters

Extracts file name along with its extension

Navigate to the directory in the path, if it doesn't exists create all the intermediary directories

Directory doesn't exists, create it

3.15.3.5 File SD_Class::open (char * *filePath*)

open(filePath)

API to open an existing file Length of file name accepted by SD library is restricted to 8 characters

Parameters

filePath	[IN] Path of the File that is to be opened

Returns

Object of the 'File' class of the opened file

Request to open Root Directory

Adding the File Node to the Linked list of File Nodes

Initialize the new File Node

Add this fileNode to the List of file Nodes

Extracts file name along with its extension

Extract the file name of the requested file, which is to be opened Extracts file name along with its extension

Check whether the file is present on the card

Navigate to the Directory and check whether the intermediate directories (if any) in the path are present

Intermediate directory doesn't exists

Finished navigating to the directory

Check whether the requested file to be opened is a directory or a file

Directory Not Found on the SD Card or the directory is already in use

File Not Found on the SD Card or the file is already in use

Add the File Node to the Linked list of Opened File Nodes

Initialize the new File Node

Extracts file name along with its extension

Add this fileNode to the List of file Nodes

Its a directory, so find the find file under the directory. Note that "." and ".." are also considered as files

3.15.3.6 File SD_Class::open (char * *filePath*, FILE_MODE *mode*)

open(filePath, mode)

API to open/create a new file Length of file name accepted by SD library is restricted to 8 characters

Parameters

filePath	[IN] Path of the File that is to be created
mode	[IN] Mode of the File (Read/Write/RW)

Returns

Object of the 'File' class of the opened file

Navigate to the Directory path and check whether the intermediate directories (if any) in the path are present

Intermediate directory doesn't exists

Extracts file name along with its extension

File already exists on the card

Seek to the end of the file

Store the last byte

File doesn't exists on the card, create it and return the file handle

Extracts file name along with its extension

Add this fileNode to the List of file Nodes

Error in memory allocation, delete the created file

If the mode is 'Read' or the file already exists

3.15.3.7 Bool SD_Class::remove (char * *filePath*)

remove(filePath)

API to delete a file

Parameters

filePath	[IN] Path of the File to be deleted

Returns

TRUE - Successful in removing the file FALSE - Unsuccessful in removing the file

Extracts file name along with its extension

Navigate to the Directory path and check whether the intermediate directories (if any) in the path are present

Intermediate directory doesn't exists

Extracts file name along with its extension

Checking whether File is already opened and is in use or not

We have navigated to the desired directory path, now delete the directory

3.15.3.8 Bool SD_Class::rmdir (char * *directoryPath*)

rmdir(directoryPath)

API to delete a directory

Parameters

directoryPath	[IN] Path of the Directory to be deleted

Returns

TRUE - Successful in removing the directory FALSE - Unsuccessful in removing the directory

Extracts file name along with its extension

Navigate to the Directory path and check whether the intermediate directories (if any) in the path are present

We have navigated to the desired directory path, now delete the directory

Intermediate directory doesn't exists

The documentation for this class was generated from the following files:

- C:/DSPShieldDoc/SD/SD.h
- C:/DSPShieldDoc/SD/SD.cpp

3.16 SPI_Class Class Reference

SPI Class.

`#include <SPI.h>`

Public Member Functions

- void begin ()
- void end ()
- void setClockDivider (int divider)
- void setDataMode (int mode)
- void setLoopBackMode (int value)
- void setBitOrder (int order)
- int transfer (int value)
- int write (unsigned int buffer[], int length)
- int read (unsigned int buffer[], int length)

3.16.1 Detailed Description

SPI Class.

Contains prototypes for functions in SPI DSP API library

3.16.2 Member Function Documentation

3.16.2.1 void SPI_Class::begin (void)

==========

begin()

Description
API to Initialize the SPI Module

Arguments

Return Value
None

Set the SPI hardware configuration

3.16.2.2 void SPI_Class::end ( )

==========

end()

Description
API to disable the SPI Module

Arguments

Return Value
None

3.16.2.3 int SPI_Class::read (unsigned int *buffer[],* int *length*)

==========

read()

Description
API to read an array of data from the SPI

Arguments

```
buffer [IN] Buffer that will holds the read data
length [IN] Length of the Buffer to be read
```

Return Value

- CSL_SOK - SPI read successful
- CSL_ESYS_INVPARAMS - Invalid parameters

swapWords (buffer, length);

if (LSBFIRST == order) {

If the User requests to receive LSB first, then swap all the 32 bits of the User data

swap32Bits (buffer, length); }

3.16.2.4 void SPI_Class::setBitOrder (int *newOrder*)

==========

setBitOrder()

Description
API to set the order of the bits shifted out of and into the SPI bus, either Least-significant bit(LSB) first or Most-significant bit(MSB) first.

Arguments

```
newOrder - Value (LSBFIRST or MSBFIRST) that is to be set for the order
           of data transfer
```

Return Value
None

3.16.2.5 void SPI_Class::setClockDivider (int *divider*)

==========

setClockDivider()

Description
API to set the SPI clock divider relative to the system clock

Arguments

```
divider - The clock divider value
```

Return Value
None

3.16.2.6 void SPI_Class::setDataMode (int *mode*)

==========

setDataMode()

Description
API to set the clock, polarity and phase

Arguments

```
mode - Mode to be set
```

Return Value
None

The modes are for the following Clock Polarity(CPOL) and Clock

Phase(CPHA) Selections

MODE CPOL CPHA

SPI_MODE0 0 0 SPI_MODE1 0 1 SPI_MODE2 1 0

SPI_MODE3 1 1

Set Clock Polarity as 0 and Clock Phase as 0

Set Clock Polarity as 0 and Clock Phase as 1

Set Clock Polarity as 1 and Clock Phase as 0

Set Clock Polarity as 1 and Clock Phase as 1

3.16.2.7 void SPI_Class::setLoopBackMode (int *value*)

==========

setLoopBackMode()

Description
API to set the Loop Back Mode of the SPI

Arguments

```
value - Value (0 or 1) that is to be set for the Loop Back field
```

Return Value
None

3.16.2.8 int SPI_Class::transfer (int *value*)

==========

transfer()

Description
API to transfer one byte over the SPI bus, both sending and receiving

Arguments

```
value - the byte to send out over the bus
```

Return Value

- On Success - The byte read from the bus
- On Failure - CSL_ESYS_INVPARAMS

Send the User data

Receive the data, which is to be sent back to the User

3.16.2.9 int SPI_Class::write (unsigned int *buffer[]*, int *length*)

==========

write()

Description
API to write an array of data to the SPI

Arguments

```
buffer [IN] Buffer that holds the data to be written
length [IN] Length of the Buffer to be written
```

Return Value

- CSL_SOK - SPI write successful
- CSL_ESYS_INVPARAMS - Invalid parameters

if (LSBFIRST == order) {

If the User requests to send LSB first, then swap all the 32 bits of the User data

swap32Bits (buffer, length); }

The documentation for this class was generated from the following files:

- C:/DSPShieldDoc/SPI/SPI.h
- C:/DSPShieldDoc/SPI/SPI.cpp

3.17 TimerClass Class Reference

Timers Class.

`#include <Timers.h>`

Public Member Functions

- int selectTimer (unsigned short gptInstance)
- int configTimer (GPT_Config gptConfig)
- int configTimer (GPT_Config gptConfig, unsigned short gptInstance)
- void initialize (long period)
- void initialize (long period, unsigned short gptInstance)
- void setPeriod (long period)
- void setPeriod (long period, unsigned short gptInstance)
- unsigned long read ()
- unsigned long read (unsigned short gptInstance)
- void start ()
- void start (unsigned short gptInstance)
- void stop ()
- void stop (unsigned short gptInstance)
- int close ()
- int close (unsigned short gptInstance)
- int setWdt (unsigned short wdtInstance, WDT_Config wdtConfig)
- int serviceWdt ()
- int startWdt ()
- int stopWdt ()
- int closeWdt ()
- void clearInterrupt ()

3.17.1 Detailed Description

Timers Class.

Contains prototypes for functions in Timers DSP API library

3.17.2 Member Function Documentation

3.17.2.1 void TimerClass::clearInterrupt ( )

closeWdt()

```
Closes the specified handle of watchdog timer
```

3.17.2.2 int TimerClass::close (void)

close()

```
Closes the specified handle of GPT
```

3.17.2.3 int TimerClass::close (unsigned short *gptInstance*)

close(unsigned short gptInstance)

```
Closes the specified handle of GPT

unsigned short timerInstance   <- Timers instance number (0 to 2)
```

3.17.2.4 int TimerClass::closeWdt ( )

closeWdt()

```
Closes the specified handle of watchdog timer
```

3.17.2.5 int TimerClass::configTimer (GPT_Config *gptConfig*)

configTimer(Timers_config)

```
Configure the Timers parameters for the given timer.

CSL_Config gptConfig  <- Timer configurations
```

Configure GPT for a given timer

3.17.2.6 int TimerClass::configTimer (GPT_Config *gptConfig,* unsigned short *gptInstance*)

configTimer(GPT_Config gptConfig, unsigned short gptInstance)

```
Configure the Timers parameters for the given timer.

CSL_Config gptConfig  <- Timer configurations
unsigned short timerInstance   <- Timers instance number (0 to 2)
```

3.17.2.7 void TimerClass::initialize (long *period*)

initialize(period)

```
Set the given period for the selected timer.

long int period  <- Timer period
```

Configure GPT for a given timer

3.17.2.8 void TimerClass::initialize (long *period,* unsigned short *gptInstance*)

initialize(long period, unsigned short gptInstance)

```
Set the given period for the selected timer.

long int period  <- Timer period
unsigned short timerInstance   <- Timers instance number (0 to 2)
```

3.17.2.9 unsigned long TimerClass::read (void)

read()

```
Returns timer count value read from the timer.
```

3.17.2.10 unsigned long TimerClass::read (unsigned short *gptInstance*)

read(unsigned short gptInstance)

```
Returns timer count value read from the timer.

unsigned short timerInstance   <- Timers instance number (0 to 2)
```

3.17.2.11 int TimerClass::selectTimer (unsigned short *gptInstance*)

selectTimer(GPT_Instance gptInstance)

```
Initialize Timers Module for the given timer instance.

unsigned short timerInstance   <- Timers instance number (0 to 2)
```

3.17.2.12 int TimerClass::serviceWdt ( )

serviceWdt()

```
Services watch dog timer
```

3.17.2.13 void TimerClass::setPeriod (long *period*)

setPeriod(period)

```
Set the given period for the selected timer.

long period  <- Timer period
```

Configure GPT for a given timer

3.17.2.14 void TimerClass::setPeriod (long *period,* unsigned short *gptInstance*)

setPeriod(long period, unsigned short gptInstance)

```
Set the given period for the selected timer.

long period  <- Timer period
unsigned short timerInstance   <- Timers instance number (0 to 2)
```

3.17.2.15 int TimerClass::setWdt (unsigned short *wdtInstance,* WDT_Config *wdtConfig*)

setWdt(WDTIM_Instance wdtInstance, WDT_Config config)

```
Configure the Watchdog Timer's parameters for the given instance.

WDTIM_Instance wdtInstance  <- Watchdog timer instance
WDT_Config config           <- Watchdog timer configurations
```

3.17.2.16 void TimerClass::start ( )

start()

```
Starts to load value from period register to
count down register.
```

3.17.2.17 void TimerClass::start (unsigned short *gptInstance*)

start(unsigned short gptInstance)

```
Starts to load value from period register to
count down register.

unsigned short timerInstance   <- Timers instance number (0 to 2)
```

3.17.2.18 int TimerClass::startWdt ( )

startWdt()

```
Starts to loading value from counter register to kick register
```

3.17.2.19 void TimerClass::stop ( )

stop()

```
Stops loading from period register to
count down register.
```

3.17.2.20 void TimerClass::stop (unsigned short *gptInstance*)

stop(unsigned short gptInstance)

```
Stops loading from period register to
count down register.

unsigned short timerInstance   <- Timers instance number (0 to 2)
```

3.17.2.21 int TimerClass::stopWdt ( )

stopWdt()

```
Stops loading value from counter register to kick register
```

The documentation for this class was generated from the following files:

- C:/DSPShieldDoc/Timers/Timers.h
- C:/DSPShieldDoc/Timers/Timers.cpp

3.18 USB_Config Class Reference

USB confuguration Class.

`#include <USB.h>`

Public Attributes

- USB_opMode opMode
- USB_APP_CALLBACK appSuspendCallBack
- USB_APP_INT_CALLBACK rxIntCallback
- USB_APP_INT_CALLBACK txIntCallback
- USB_APP_INT_CALLBACK rxCompleteCallback
- unsigned short * deviceDescPtr
- unsigned short * deviceQualDescPtr
- unsigned short * cfgDescPtr
- unsigned short * cfgDescFSPtr
- unsigned short * strDescPtr [USB_STRDESC_COUNT]

3.18.1 Detailed Description

USB confuguration Class.

3.18.2 Member Data Documentation

3.18.2.1 USB_APP_CALLBACK USB_Config::appSuspendCallBack

Callback function for suspend operation

3.18.2.2 unsigned short* USB_Config::cfgDescFSPtr

Full speed config descriptor pointer

3.18.2.3 unsigned short* USB_Config::cfgDescPtr

Config descriptor pointer

3.18.2.4 unsigned short* USB_Config::deviceDescPtr

Device descriptor pointer

3.18.2.5 unsigned short* USB_Config::deviceQualDescPtr

Device qualifier descriptor pointer

3.18.2.6 USB_opMode USB_Config::opMode

USB operation mode

3.18.2.7 USB_APP_INT_CALLBACK USB_Config::rxCompleteCallback

Callback function for RX complete interrupt

3.18.2.8 USB_APP_INT_CALLBACK USB_Config::rxIntCallback

Callback function for RX interrupt

3.18.2.9 unsigned short* USB_Config::strDescPtr[USB_STRDESC_COUNT]

String descriptor pointer

3.18.2.10 USB_APP_INT_CALLBACK USB_Config::txIntCallback

Callback function for TX interrupt

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

- C:/DSPShieldDoc/USB/USB.h

3.19 USBClass Class Reference

USB Class.

`#include <USB.h>`

Public Member Functions

- USBClass ()
- int init ()
- int config (USB_Config)
- int connect ()
- int disconnect ()
- int suspend ()
- void * requestPipe (unsigned short, unsigned short)
- int closePipe (void *)
- int configPipe (void *, Pipe_Config)
- int stallPipe (void *)
- int clearPipe (void *)
- int writePipe (void *, unsigned short, unsigned short *)
- int readPipe (void *, unsigned short, unsigned short *)
- int handleInterrupts ()
- int setParams (void *)
- int dmaRxStop (void *)
- int dmaTxStop (void *)

Public Attributes

- unsigned short * usbBuffTxRxPtr1
- unsigned short * usbBuffTxRxPtr2
- unsigned short * usbBuffTxRxPtr3
- unsigned short * usbBuffTxRxPtr4

3.19.1 Detailed Description

USB Class.

Contains prototypes for functions in USB library

3.19.2 Constructor & Destructor Documentation

3.19.2.1 USBClass::USBClass ( )

USB Class Constructor

3.19.3 Member Function Documentation

3.19.3.1 int USBClass::clearPipe (void * *hEp*)

==========

clearPipe()

Description
This function clears an end point stall

Arguments

```
*       *hEp    <- End point handle
*
```

Return Value int

- 0:Success
- Error status

3.19.3.2 int USBClass::closePipe (void * *hEp*)

==========

closePipe()

Description
Function to release an endpoint

Arguments

```
*       *hEp    <- End point handle
*
```

Return Value int

- 0:Success
- Error status

3.19.3.3 int USBClass::config (USB_Config *usbConfig*)

==========

config()

Description
Configures the USB module

Arguments

```
*       usbConfig    <- Configuration structure
*
*
```

Return Value int

- 0:Success
- Error status

3.19.3.4 int USBClass::configPipe (void * *hEp*, Pipe_Config *pipeConfig*)

==========

configPipe()

Description
Function to configure endpoint

Arguments

```
*       *hEp       <- End point handle
*       pipeConfig <- End point config structure
*
```

Return Value int

- 0:Success
- Error status

3.19.3.5 int USBClass::connect ( )

==========

connect()

Description
This function connects the USB module to upstream port

Arguments

*

Return Value int

- 0:Success
- Error status

3.19.3.6 int USBClass::disconnect ( )

==========

disconnect()

Description
This function disconnects the USB module to upstream port

Arguments

*

Return Value int

- 0:Success
- Error status

3.19.3.7 int USBClass::dmaRxStop (void * *hEp*)

==========

dmaRxStop()

Description
Stops DMA data receive operation

Arguments

```
*       *hEp    <- End point handle
*
```

Return Value int

- 0:Success
- Error status

3.19.3.8 int USBClass::dmaTxStop (void * *hEp*)

==========

dmaTxStop()

Description
Stops DMA data transmit operation
Arguments

```
*       *hEp    <- End point handle
*
```

Return Value int

- 0:Success
- Error status

3.19.3.9 int USBClass::handleInterrupts (void)

==========

handleInterrupts()

Description
Function to handle the USB interrupts
Arguments

```
*
```

Return Value int

- 0:Success
- Error status

Read the masked interrupt status register

Read queue pending register1

Clear the interrupts

Reset interrupt

Resume interrupt

Check End point0 interrupts

Service the RXPKTRDY after reading the FIFO

zero data

Return 0x0000 (reserved for future use)

Check Data Out Ready

Check Data In Ready

Connect interrupt

Disconnect interrupt

Suspend interrupt

3.19.3.10 int USBClass::init (void)

==========

init()

Description
Function initializes USB module.

Arguments

*

Return Value int

- 0:Success
- Error status

3.19.3.11 int USBClass::readPipe (void * *hEp*, unsigned short *bytes*, unsigned short * *data*)

==========

readPipe()

Description
Function to read data from USB pipe

Arguments

```
*       *hEp  <- End point handle
*       bytes <- Number of bytes to be read
*       *data <- Pointer to data buffer
*
```

Return Value int

- 0:Success
- Error status

3.19.3.12 void * USBClass::requestPipe (unsigned short *pipeNum*, unsigned short *direction*)

==========

requestPipe()

Description
Function to request the endpoint for data communication

Arguments

```
*       pipeNum   <- Pipe number
*       direction <- Transfer direction, USB_IN or USB_OUT
*
```

Return Value void *

- Valid pipe handle if success
- NULL if error

3.19.3.13 int USBClass::setParams (void * *hEp*)

==========

setParams()

Description
Function to initialize the endpoint structures

Arguments

```
*       *hEp  <- End point handle
*
```

Return Value int

- 0:Success
- Error status

3.19.3.14 int USBClass::stallPipe (void * *hEp*)

==========

stallPipe()

Description
This function stalls an end point

Arguments

```
*       *hEp    <- End point handle
*
```

Return Value int

- 0:Success
- Error status

3.19.3.15 int USBClass::suspend ( )

==========

suspend()

Description
This function suspends the USB device. This function informs the application about the device suspend through a call back function.

Arguments

```
*
```

Return Value int

- 0:Success
- Error status

3.19.3.16 int USBClass::writePipe (void * *hEp*, unsigned short *bytes*, unsigned short * *data*)

==========

writePipe()

Description
Function to write data to USB pipe

Arguments

```
*       *hEp  <- End point handle
*       bytes <- Number of bytes to be written
*       *data <- Pointer to data buffer
*
```

Return Value int

- 0:Success
- Error status

3.19.4 Member Data Documentation

3.19.4.1 unsigned short* USBClass::usbBuffTxRxPtr1

USB data buffer 1 pointer

3.19.4.2 unsigned short* USBClass::usbBuffTxRxPtr2

USB data buffer 2 pointer

3.19.4.3 unsigned short* USBClass::usbBuffTxRxPtr3

USB data buffer 3 pointer

3.19.4.4 unsigned short* USBClass::usbBuffTxRxPtr4

USB data buffer 4 pointer

The documentation for this class was generated from the following files:

- C:/DSPShieldDoc/USB/USB.h
- C:/DSPShieldDoc/USB/USB.cpp

3.20 WDT_Config Struct Reference

Configuration structure.

`#include <Timers.h>`

Public Attributes

- unsigned short counter
- unsigned short prescale

3.20.1 Detailed Description

Configuration structure.

Hardware register configuration structure for WDT

3.20.2 Member Data Documentation

3.20.2.1 unsigned short WDT_Config::counter

Counter value for the WDT

3.20.2.2 unsigned short WDT_Config::prescale

Prescale value for the WDT

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

- C:/DSPShieldDoc/Timers/Timers.h

Chapter 4

File Documentation

4.1 C:/DSPShieldDoc/AnalogReadWrite/examples/AnalogRead/AnalogRead.cpp File Reference

Read analog pin data.

Functions

- void setup ()

  Set up Serial Display.
- void loop ()

  Read analog pin and send the value to serial monitor.

4.1.1 Detailed Description

Read analog pin data.

Reads analog pin data from A0 pin contnuosly and displays it on the serial console.

4.2 C:/DSPShieldDoc/Audio/Audio.cpp File Reference

Audio implementation.

```
#include "Audio.h"
```

Variables

- AudioClass AudioC
- Uint16 i2sDmaLeftBuff1 [2][I2S_DMA_BUF_LEN]
- Uint16 i2sDmaRightBuff1 [2][I2S_DMA_BUF_LEN]
- Uint16 i2sDmaLeftBuff2 [2][I2S_DMA_BUF_LEN]
- Uint16 i2sDmaRightBuff2 [2][I2S_DMA_BUF_LEN]

4.2.1 Detailed Description

Audio implementation.

4.2.2 Variable Documentation

4.2.2.1 AudioClass AudioC

Class identifier declaration

4.2.2.2 Uint16 i2sDmaLeftBuff1[2][I2S_DMA_BUF_LEN]

DMA left channel buffer

4.2.2.3 Uint16 i2sDmaLeftBuff2[2][I2S_DMA_BUF_LEN]

DMA left channel buffer 2

4.2.2.4 Uint16 i2sDmaRightBuff1[2][I2S_DMA_BUF_LEN]

DMA right channel buffer

4.2.2.5 Uint16 i2sDmaRightBuff2[2][I2S_DMA_BUF_LEN]

DMA right channel buffer 2

4.3 C:/DSPShieldDoc/Audio/Audio.h File Reference

Audio library header file.

```
#include "core.h"
```

Classes

- class AudioClass
  Audio Class.

Macros

- #define I2S_DMA_BUF_LEN (512)
- #define DMA_CHAN_ReadL (CSL_DMA_CHAN4)
- #define DMA_CHAN_ReadR (CSL_DMA_CHAN5)
- #define DMA_CHAN_WriteL (CSL_DMA_CHAN6)
- #define DMA_CHAN_WriteR (CSL_DMA_CHAN7)
- #define SAMPLING_RATE_8_KHZ (8000u)
- #define SAMPLING_RATE_11_KHZ (11025u)
- #define SAMPLING_RATE_12_KHZ (12000u)
- #define SAMPLING_RATE_16_KHZ (16000u)
- #define SAMPLING_RATE_22_KHZ (22050u)
- #define SAMPLING_RATE_24_KHZ (24000u)
- #define SAMPLING_RATE_32_KHZ (32000u)
- #define SAMPLING_RATE_44_KHZ (44100u)
- #define SAMPLING_RATE_48_KHZ (48000u)
- #define CHANNEL_MONO (1)
- #define CHANNEL_STEREO (2)

Variables

- AudioClass AudioC

4.3.1 Detailed Description

Audio library header file.

4.3.2 Macro Definition Documentation

4.3.2.1 #define CHANNEL_MONO (1)

Macro to indicate the Channel type as Mono

4.3.2.2 #define CHANNEL_STEREO (2)

Macro to indicate the Channel type as Stereo

4.3.2.3 #define DMA_CHAN_ReadL (CSL_DMA_CHAN4)

DMA left read channel

4.3.2.4 #define DMA_CHAN_ReadR (CSL_DMA_CHAN5)

DMA right read channel

4.3.2.5 #define DMA_CHAN_WriteL (CSL_DMA_CHAN6)

DMA left write channel

4.3.2.6 #define DMA_CHAN_WriteR (CSL_DMA_CHAN7)

DMA right write channel

4.3.2.7 #define I2S_DMA_BUF_LEN (512)

Buffer length

4.3.2.8 #define SAMPLING_RATE_11_KHZ (11025u)

Sampling Rate as 11 kHz

4.3.2.9 #define SAMPLING_RATE_12_KHZ (12000u)

Sampling Rate as 12 kHz

4.3.2.10 #define SAMPLING_RATE_16_KHZ (16000u)

Sampling Rate as 16 kHz

4.3.2.11 #define SAMPLING_RATE_22_KHZ (22050u)

Sampling Rate as 22 kHz

4.3.2.12 #define SAMPLING_RATE_24_KHZ (24000u)

Sampling Rate as 24 kHz

4.3.2.13 #define SAMPLING_RATE_32_KHZ (32000u)

Sampling Rate as 32 kHz

4.3.2.14 #define SAMPLING_RATE_44_KHZ (44100u)

Sampling Rate as 44 kHz

4.3.2.15 #define SAMPLING_RATE_48_KHZ (48000u)

Sampling Rate as 48 kHz

4.3.2.16 #define SAMPLING_RATE_8_KHZ (8000u)

Sampling Rate as 8 kHz

4.3.3 Variable Documentation

4.3.3.1 AudioClass AudioC

Audio class instance extern which can used by application programs to access Audio DSP APIs

Class identifier declaration

4.4 C:/DSPShieldDoc/Audio/examples/AudioLoopback/AudioLoopback.cpp File Reference

Audio Loopback demo plays back audio input. Audio library reads data from codec audio IN and sends the data to codec audio OUT which can be listened on headphone.

```
#include "Audio.h"
#include "OLED.h"
```

Functions

- interrupt void dmaIsr (void)
- void setup ()

  Setup code that runs once.
- void **loop** ()

4.4.1 Detailed Description

Audio Loopback demo plays back audio input. Audio library reads data from codec audio IN and sends the data to codec audio OUT which can be listened on headphone.

For any data processing on audio data, use the pointers AudioC.audioInLeft and AudioC.audioInRight. Audio data will be available in these buffers when dmaIsr is triggered. Buffer having the data read from codec will be indicated by the index 'activeInBuf'

4.4.2 Function Documentation

4.4.2.1 interrupt void dmaIsr (void)

Copies audio samples received from audio In to the audio Out of the Codec

4.4.2.2 void setup (void)

Setup code that runs once.

Initialize OLED module for status display

Initialize audio modules

Here, Audio library is configured for loopback mode. Calling AudioC.Audio() with out any argument sets the Loopback mode. It gives control to the Audio library to directly copy any audio samples coming through Audio IN back to the Audio Out of the Codec

Enable interrupts and start audio data transfer

4.5 C:/DSPShieldDoc/Audio/examples/Filter/FIR/FIR.cpp File Reference

FIR Filter demo low-pass filters audio input.

```
#include "Audio.h"
#include "filter.h"
#include "OLED.h"
```

Macros

- #define FILTER_LENGTH (202)

  Length of the Coefficient Vector.

Functions

- interrupt void dmaIsr (void)

  DMA Interrupt Service Routine - Copies data to/from output/input.
- void setup ()

  Initializes OLED and Audio modules.
- void **loop** ()

Variables

- int filterIn1 [I2S_DMA_BUF_LEN]

Buffer to hold the Input Samples of left channel, for the FIR filter.

- int filterIn2 [I2S_DMA_BUF_LEN]

  Buffer to hold the Input Samples of right channel, for the FIR filter.

- int filterOut1 [I2S_DMA_BUF_LEN]

  Buffer to hold the Output Samples of left channel, from the FIR filter.

- int filterOut2 [I2S_DMA_BUF_LEN]

  Buffer to hold the Output Samples of right channel, from the FIR filter.

- unsigned short readyForFilter = 0
- unsigned short filterBufAvailable = 0
- unsigned short writeBufIndex = 0

  Variable to switch between the data buffers of the Audio library.

- int coeffs [FILTER_LENGTH]

  Filter coefficients with 200 taps - LPF with cut-off frequency as 2kHz.

- int delayBufferL [FILTER_LENGTH+2]

  Delay buffer used by the FIR filtering algorithm for Left Channel.

- int delayBufferR [FILTER_LENGTH+2]

  Delay buffer used by the FIR filtering algorithm for Right Channel.

4.5.1 Detailed Description

FIR Filter demo low-pass filters audio input.

Reads data from codec audio IN, sends the data to FIR filter and finally sends the filtered audio to the codec OUT which can be listened on headphone.

Filter is configured as a Low Pass Filter(LPF) with cut-off frequency as 2kHz.

4.5.2 Function Documentation

4.5.2.1 interrupt void dmaIsr (void)

DMA Interrupt Service Routine - Copies data to/from output/input.

Data read from codec is copied to filter input buffers for both channels. Filtering is done after configuring DMA for next block of transfer ensuring no data loss

Filtered buffers need to be copied to audio out buffers manually as audio library is configured for non-loopback mode

Calling AudioC.isrDma() will copy the buffers to Audio Out of the Codec, initiates next DMA transfers of Audio samples to and from the Codec

Check if filter buffers are ready. No filtering required for write interrupt

Filter Left Audio Channel

Filter Right Audio Channel

4.5.2.2 void setup (void)

Initializes OLED and Audio modules.

Initialize OLED module for status display

Clear all the data buffers

Clear the delay buffers, which will be used by the FIR filtering algorithm, These buffers need to be initialized to all zeroes in the beginning of the FIR filtering algorithm

Audio library is configured for non-loopback mode. Gives enough time for FIR filter processing in ISR, then buffers manually copied to output

Enable interrupts and start filtering

4.5.3 Variable Documentation

4.5.3.1 int coeffs[FILTER_LENGTH]

Initial value:

```
= {
        7,      6,      4,      2,      0,     -2,     -5,     -7,     -9,
      -11,    -12,    -12,    -12,    -10,     -8,     -4,      0,      5,
       10,     15,     19,     23,     25,     26,     24,     21,     16,
        9,      0,    -10,    -20,    -30,    -39,    -46,    -50,    -51,
      -49,    -42,    -31,    -17,      0,     19,     38,     57,     73,
       86,     93,     94,     89,     76,     56,     31,      0,    -34,
      -68,   -100,   -129,   -150,   -163,   -164,   -154,   -132,    -97,
      -53,      0,     58,    117,    173,    222,    260,    282,    286,
      269,    231,    172,     94,      0,   -104,   -212,   -318,   -412,
     -488,   -538,   -555,   -533,   -467,   -356,   -200,      0,    239,
      510,    805,   1114,   1424,   1725,   2005,   2252,   2455,   2607,
     2701,   2733,   2701,   2607,   2455,   2252,   2005,   1725,   1424,
     1114,    805,    510,    239,      0,   -200,   -356,   -467,   -533,
     -555,   -538,   -488,   -412,   -318,   -212,   -104,      0,     94,
      172,    231,    269,    286,    282,    260,    222,    173,    117,
       58,      0,    -53,    -97,   -132,   -154,   -164,   -163,   -150,
     -129,   -100,    -68,    -34,      0,     31,     56,     76,     89,
       94,     93,     86,     73,     57,     38,     19,      0,    -17,
      -31,    -42,    -49,    -51,    -50,    -46,    -39,    -30,    -20,
      -10,      0,      9,     16,     21,     24,     26,     25,     23,
       19,     15,     10,      5,      0,     -4,     -8,    -10,    -12,
      -12,    -12,    -11,     -9,     -7,     -5,     -2,      0,      2,
        4,      6,      7,      0
}
```

Filter coefficients with 200 taps - LPF with cut-off frequency as 2kHz.

4.5.3.2 unsigned short filterBufAvailable = 0

Variable to indicate that the FIR filtering is completed and the filtered samples are available in the "filterOut" buffer

4.5.3.3 unsigned short readyForFilter = 0

Variable to indicate to the FIR Filtering section that the Input samples are ready to be filtered

4.6 C:/DSPShieldDoc/Audio/examples/Filter/IIR/IIR.cpp File Reference

IIR Filter demo.

```
#include "Audio.h"
#include "filter.h"
#include "OLED.h"
```

Macros

- #define **IIR_ORDER_MAX** (16)
- #define **DELAY_COUNT** (4)
- #define **DELAY_BUF_SIZE** ((IIR_ORDER_MAX/2)∗(DELAY_COUNT))
- #define FILTER_LENGTH (24)

  Length of the Coefficient Vector, used by the IIR filter.

Functions

- interrupt void dmaIsr (void)

  DMA Interrupt Service Routine – copy data to/from output/input.
- void setup ()

  Initializes OLED and Audio modules.
- void **loop** ()

Variables

- unsigned short readyForFilter = 0
- unsigned short filterBufAvailable = 0
- unsigned short writeBufIndex = 0

  Variable to switch between the data buffers of the Audio library.
- int filterInLeft [I2S_DMA_BUF_LEN]

  Buffer to hold the Input Samples of left channel, for the IIR filter.
- int filterInRight [I2S_DMA_BUF_LEN]

  Buffer to hold the Input Samples of right channel, for the IIR filter.
- int filterOutLeft [I2S_DMA_BUF_LEN]

  Buffer to hold the Output Samples of left channel, from the IIR filter.
- int filterOutRight [I2S_DMA_BUF_LEN]

  Buffer to hold the Output Samples of right channel, from the IIR filter.
- int coeffs [FILTER_LENGTH]

  Filter coefficients - 8th order, 5k cutoff LPF, assumes 48k sample frequency.
- int delayBufferL [DELAY_BUF_SIZE]

  Delay buffer used by the IIR filtering algorithm for Left Channel.
- int delayBufferR [DELAY_BUF_SIZE]

  Delay buffer used by the IIR filtering algorithm for Right Channel.

4.6.1 Detailed Description

IIR Filter demo.

This demo reads data from codec audio IN, sends the data to IIR filter and finally sends the filtered audio to the codec OUT which can be listened on headphone.

This program acts as a 8th order Low Pass Filter(LPF) with cut-off frequency as 5kHz.

4.6.2 Function Documentation

4.6.2.1 interrupt void dmaIsr (void)

DMA Interrupt Service Routine – copy data to/from output/input.

Data read from codec is copied to filter input buffers. Filtering is done after configuring DMA for next block of transfer ensuring no data loss

Filtered buffers need to be copied to audio out buffers as audio library is configured for non-loopback mode

Calling AudioC.isrDma() will copy the buffers to Audio Out of the Codec, initiates next DMA transfers of Audio samples to and from the Codec

Filter Left Audio Channel - configuring filter for 8th order

Filter Right Audio Channel - configuring filter for 8th order

4.6.2.2 void setup (void)

Initializes OLED and Audio modules.

Initialize OLED module for status display

Clear all the data buffers

Clear the delay buffers, which will be used by the IIR filtering algorithm, These buffers need to be initialized to all zeroes in the beginning of the IIR filtering algorithm

Audio library is configured for non-loopback mode. Gives enough time for processing in ISR

4.6.3 Variable Documentation

4.6.3.1 int coeffs[FILTER_LENGTH]

Initial value:

```
= {
    15, -8139, 2067, 8192, 16389, 8194,
    15, -8630, 2686, 8192, 16589, 8400,
    15, -9713, 4051, 8192, 16379, 8190,
    15, -11618, 6453, 8192, 16179, 7990
}
```

Filter coefficients - 8th order, 5k cutoff LPF, assumes 48k sample frequency.

4.6.3.2 unsigned short filterBufAvailable = 0

Variable to indicate that the IIR filtering is completed and the filtered samples are available in the "filterOut" buffer

4.6.3.3 unsigned short readyForFilter = 0

Variable to indicate to the IIR Filtering section that the Input samples are ready to be filtered

4.7 C:/DSPShieldDoc/Audio/examples/Recorder/Recorder.cpp File Reference

Wave recorder demo: records wave file to SD card Stores the audio samples received from audio module to a file on SD card in wave format.

```
#include "SD.h"
#include "Audio.h"
#include "OLED.h"
```

Macros

- #define RECORD_DURATION (60)
- #define SINGLE_BUFFER_SIZE (4096)
- #define NO_OF_WRITE_BUFFERS (5)

  Circular Buffer Size.

Functions

- void updateWavFileHeader (long samplingRate, int noOfChannels)

Forms wave file header.

- interrupt void dmaIsr (void)
- void setup ()

  Set up wave file on SD Card, initialize Audio.
- void swapBytes (int *buffer, int length)
- void loop ()

  Record audio input, write to file for specified duration.

Variables

- long recFileSize = 0

  #define ENABLE_STEREO_RECORD
- long recordCounter = 0
- File **fileHandle**
- unsigned short audioLibWriteBufIndex = 0

  Variable to switch between the data buffers of the Audio library.
- volatile bool stopRecording = false
- int writeToFileL [I2S_DMA_BUF_LEN]

  Data buffers to transfer the audio data to file write buffers.
- int **writeToFileR** [I2S_DMA_BUF_LEN]
- int codecIndex = 0

  Variables to store different buffer indexes used by the program.
- int **writeBufIndex** = 0
- int **writeToFileIndex** = 0
- volatile long **fileCounter** = 0
- char waveFileHeader [44] = {0}

  Stores format for wave file header.
- long samplingRate = SAMPLING_RATE_48_KHZ
- int **channelType** = CHANNEL_MONO
- int writeBuf [NO_OF_WRITE_BUFFERS][SINGLE_BUFFER_SIZE] = {0}

  Buffers to store the data to be written to file.
- volatile bool **buffAvailable** [NO_OF_WRITE_BUFFERS] = {false}

4.7.1 Detailed Description

Wave recorder demo: records wave file to SD card Stores the audio samples received from audio module to a file on SD card in wave format.

Default configurations: Sampling rate - 48KHz Channel Type - Mono Duration - 60 secs

Note: Stereo mode configurations can be enabled by defining macro 'ENABLE_STEREO_RECORD'

Changing the default configurations: Sampling rate - Change the value of 'samplingRate' which can be one of the macros SAMPLING_RATE_XX_KHZ defined in Audio.h Channel Type - Change the value of 'channelType' to C↩ HANNEL_MONO for mono recording Duration - Change the value of macro 'RECORD_DURATION' which defines the duration of record in secs

CAUTION: DO NOT power off the DSP shield when record is in progress. Recorded file will not be saved unless program ends smoothly.

SD Card performance Requirements for proper record

SD card used for record should be of class 4 or higher for proper recording in stereo mode. Recording in stereo mode can be done without data loss up to 32KHz sampling frequency with class 4 SD card. It is not recommended to use sampling frequency more than 32KHz for stereo mode which causes data loss during record.

4.7.2 Macro Definition Documentation

4.7.2.1 #define RECORD_DURATION (60)

Record duration in seconds. Modify this macro to change the duration of record

4.7.2.2 #define SINGLE_BUFFER_SIZE (4096)

Size of the Block of data. File.write() writes 'SINGLE_BUFFER_SIZE' number of words at a time to the recorded file

4.7.3 Function Documentation

4.7.3.1 interrupt void dmaIsr (void)

Copies the audio Rx data into Tx buffers and copies the audio data to file write buffers If recording audio (reading), copy audio input to buffer

Data read from codec is copied to file input buffer

If still recording and the intermediate buffer is ready to read

Recording in Stereo mode

Copy to left and right channels

Audio data of the Wave file should be in little endian form, hence swap the bytes before writing it to the file

Recording in Mono mode

Only need to copy one channel... arbitrarily pick left

Audio data of the Wave file should be in little endian form, hence swap the bytes before writing it to the file

Once we've written one buffer of data

Wrap writeBufIndex around

indicate that the current buffer is available (since we've already written the data)

Increment codec index, wrap around if at end

4.7.3.2 void loop (void)

Record audio input, write to file for specified duration.

Data is ready to be written to SD card

increment writeToFileIndex, wrap around if necessary

If the recording has hit the set duration, stop recording

4.7.3.3 void setup (void)

Set up wave file on SD Card, initialize Audio.

Initialize OLED module for file name display

'recFileSize' will hold the file size in terms of bytes

File Size in bytes

Double the file size in case of stereo channel

File.write() will write (2∗SINGLE_BUFFER_SIZE) bytes of data at once, hence the value (recordCounter ∗ (2∗SI↩ NGLE_BUFFER_SIZE)) will give the recorded file size.

recordCounter value might contain fractional part, which will not be taken into account because of conversion from float to long. In such case, increase the recordCounter value by 1 so that recorded file duration will not be less than RECORD_DURATION

Update and write wave file header

Initialize Audio

4.7.3.4 void swapBytes (int * *buffer,* int *length*)

function to swap bytes (eg to get from big endian to little endian)

4.7.3.5 void updateWavFileHeader (long *samplingRate,* int *noOfChannels*)

Forms wave file header.

When the 'recordCounter' value is calculated, it might be in fractions, hence set the file size in terms of 'record↩ Counter' value

Size of PCM samples

Size of WAV header

Update Header with the Chunk Id: "RIFF", in Big Endian Mode

Update Header with the Chunk Size, this size is equal to total file size minus 8 bytes. This is the size of the rest of the chunk following this number. It should be written in Little Endian mode

Update Header with the File format: "WAVE", in Big Endian Mode

Update Header with the Sub Chunk Id: "fmt ", in Big Endian Mode

Update Header with the Sub Chunk1 Size, this size is 16 for PCM data. It should be written in Little Endian mode

Update Header with the Audio Format. For PCM format, its value should be 1. It should be written in Little Endian mode

Update Header with the Number of channels present in the Audio file. Mono = 1, Stereo = 2, it should be written in Little Endian mode

Update Header with the Sampling rate of the Audio file. It should be written in Little Endian mode

Update Header with the Byte Rate of the Audio File. Its value is equal to, ByteRate = SampleRate * NumChannels * BitsPerSample / 8. It should be written in Little Endian mode

Update Header with the Block Align. Its value is, BlockAlign = NumChannels * BitsPerSample / 8. In Little Endian Mode

Update Header with the Bits Per Sample, 8 bits = 8, 16 bits = 16. In Little Endian Mode

Update Header with Sub Chunk2 ID: "data". Its in Big Endian Mode

Update Header with the Sub Chunk2 Size, this size is equal to total file size minus 24 bytes. Its in Little Endian Mode

LSB of Sub Chunk Size

MSB of Sub Chunk Size

4.7.4 Variable Documentation

4.7.4.1 long recFileSize = 0

#define ENABLE_STEREO_RECORD

Variable to hold Record file size in terms of KiloBytes

4.7.4.2 long recordCounter = 0

Counter to indicate the number of times to call File.write(), to write the audio data to the recorded file. File.write() will write (2∗SINGLE_BUFFER_SIZE) bytes of data at once, hence the value (recordCounter ∗ (2∗SINGLE_BUFFER_SIZE)) will give the recorded file size.

4.7.4.3 long samplingRate = SAMPLING_RATE_48_KHZ

Record channel type.

4.7.4.4 volatile bool stopRecording = false

Variable to indicate when to stop recording (if there's an error or if the recording has reached its full duration

4.8 C:/DSPShieldDoc/DigitalReadWrite/examples/DigitalWrite/DigitalWrite.cpp File Reference

Digital Read Write: Toggles LEDs.

Functions

- void setup ()
- void loop ()

  toggles LEDs

Variables

- int ledValue = 0

  Variable represents which of 3 LEDs to turn on.

4.8.1 Detailed Description

Digital Read Write: Toggles LEDs.

Toggles the states of LED0, LED1 and LED2 continuously in a loop.

4.8.2 Function Documentation

4.8.2.1 void loop (void)

toggles LEDs

Write a value to the LED0

On, if ledValue == 0

Write a value to the LED1

On, if ledValue == 1

Write a value to the LED2

On, if ledValue == 2

Giving a delay of 1sec between toggling

Toggling LED2 to Off, for clear toggling from LED2 to LED0

increment ledValue to turn next led on

4.8.2.2 void setup (void)

set LED pin states as outputs

4.9 C:/DSPShieldDoc/DigitalReadWrite/examples/GPIO/GPIO.cpp File Reference

Functions

- void setup ()

  Set modes for GPIO12 & GPIO13, and set to input and output pins.
- void loop ()

4.9.1 Detailed Description

@ brief GPIO Read Write

Configures GPIO12 as input pin and GPIO13 as output pin. Both the GPIO pins (pin16 and pin18 on P2)need to be shorted to check the functionality of the module. Set/Reset the output pin(GPIO13) P2 pin16 and read the value of input pin (GPIO12) P2 pin18 and compare both values. If both the values match, then the test passes.

4.9.2 Function Documentation

4.9.2.1 void loop (void)

This API continuosly sets the pin value of DSP_GPIO13 pin high and low one after another. Since DSP_GPIO12 is configured as input pin and DSP_GPIO13 is configured as output pin. Value written to DSP_GPIO13 pin should be sent to DSP_GPIO12 pin, if DSP_GPIO13 and DSP_GPIO12 pins are shorted externally DSP_GPIO12 is configured as input pin, whereas DSP_GPIO13 is configured as output pin. Value written to DSP_GPIO13 pin will be sent to DSP_GPIO12 pin, since DSP_GPIO13 and DSP_GPIO12 pins are shorted

GPIO set test - Sets GPIO pin value to high

set the output pin value (GPIO13)

Read the input pin value (GPIO12). gpioRead() will return the pin value of the requested pin, if its in 'high' state returns 1, else returns 0 for low state

wait for a second

GPIO reset test - Sets GPIO pin value to low

set the output pin value (GPIO13)

Read the input pin value (GPIO12)

wait for a second

4.9.2.2 void setup (void)

Set modes for GPIO12 & GPIO13, and set to input and output pins.

Set mode for GPIO12 and GPIO13 - This configures the Pin Muxing to enable the GPIO pins. The system consists of pins that will be multiplexed between various peripherals, hence configure the system to multiplex those pins and enable GPIO pins.

Configure GPIO12 as input pin

Returns 0 on success

Configure GPIO13 as output pin

4.10 C:/DSPShieldDoc/DigitalReadWrite/examples/readDIPSwitch/readDIPSwitch.cpp File Reference

DIP switch read state demo.

Functions

- void setup ()

  Read DIP switch states.
- void **loop** ()

Variables

- unsigned int **pinVal**

4.10.1 Detailed Description

DIP switch read state demo.

This program reads the DIP switch states on Switches 1-4 and displays on serial monitor

Note: This demo is specific for C5535 DSP shield. May not work/compile for other platforms

4.10.2 Function Documentation

4.10.2.1 void setup (void)

Read DIP switch states.

Read the state of Switch 1

Read the state of Switch 2

Read the state of Switch 3

Read the state of Switch 4

4.11 C:/DSPShieldDoc/FFT/examples/FFT/FFT_ex.cpp File Reference

Example sketch to demonstrate functionality of FFT DSP API library.

```
#include "FFT.h"
#include "Audio.h"
```

Macros

- #define CIRCULAR_BUFFER_SIZE (4u)

- #define SINGLE_BUFFER_SIZE (I2S_DMA_BUF_LEN)
- #define OVERLAP_SIZE (1024)

Functions

- void copyBuf16 (short *input, short *output, int dataLength)
- interrupt void dmaIsr (void)
- void setup ()
- void **loop** ()

Variables

- long coeff32 [I2S_DMA_BUF_LEN]
- short fftBufL [CIRCULAR_BUFFER_SIZE][SINGLE_BUFFER_SIZE]
- short fftBufR [CIRCULAR_BUFFER_SIZE][SINGLE_BUFFER_SIZE]
- short overlapL [OVERLAP_SIZE]
- short **overlapR** [OVERLAP_SIZE]
- int fftCodecIndex = 0
- int fftProcessIndex = 0
- volatile int readyForFFT [CIRCULAR_BUFFER_SIZE] = {0}
- volatile int fftbufAvailable [CIRCULAR_BUFFER_SIZE] = {0}
- int writeBufIndex

4.11.1 Detailed Description

Example sketch to demonstrate functionality of FFT DSP API library.

FFT transform is computed for audio received from Audio input port. Audio Input is multiplied by low-pass filter in the frequency domain IFFT is computed for FFT output in order to get the audio signal into time domain. The resulting signal is sent to the Audio output.

4.11.2 Macro Definition Documentation

4.11.2.1 #define CIRCULAR_BUFFER_SIZE (4u)

Circular Buffer Size

4.11.2.2 #define OVERLAP_SIZE (1024)

Size of the Overlap buffer, used by FFT-filtering algorithm

4.11.2.3 #define SINGLE_BUFFER_SIZE (I2S_DMA_BUF_LEN)

Size of the Block of data. Number of samples that will be processed by FFT-filtering

4.11.3 Function Documentation

4.11.3.1 void copyBuf16 (short * *input*, short * *output*, int *dataLength*)

API to copy 16 bit integer elements from one buffer to an another buffer

4.11.3.2 interrupt void dmaIsr (void)

Interrupt Service Routine, which will be called when the audio samples have been completely transferred from/to the Audio Codec Clear the DMA interrupt

If DMA interrupt indicates Read, copy left & right audio buffers, and indicate data is ready for FFT

If DMA interrupt indicates Write, copy output to left & right audio buffers

Process FFT, filtering and IFFT of the input samples

Filter Left Channel

Filter Right Channel

4.11.3.3 void setup (void)

Initializes Audio and FFT library, along with the buffers used in this demo Initialize buffer that indicate when reads/writes are ready for processing

Initialize overlap buffers

Initialize FFT

Initialize Audio

4.11.4 Variable Documentation

4.11.4.1 long coeff32[I2S_DMA_BUF_LEN]

Buffer to hold the co-efficients used for FFT-filtering algorithm

4.11.4.2 volatile int fftbufAvailable[CIRCULAR_BUFFER_SIZE] = {0}

Variable to indicate that a particular buffer (input audio samples) has been processed by FFT-filtering

4.11.4.3 short fftBufL[CIRCULAR_BUFFER_SIZE][SINGLE_BUFFER_SIZE]

Buffer containing the left channel audio samples, that will be sent as input to FFT-filtering algorithm

4.11.4.4 short fftBufR[CIRCULAR_BUFFER_SIZE][SINGLE_BUFFER_SIZE]

Buffer containing the right channel audio samples, that will be sent as input to FFT-filtering algorithm

4.11.4.5 int fftCodecIndex = 0

Variable to hold the index of the circular buffer, where the next input audio samples must to be copied

4.11.4.6 int fftProcessIndex = 0

Variable to hold the index of the circular buffer, where the next input audio samples must sent for FFT-filtering algorithm

4.11.4.7 short overlapL[OVERLAP_SIZE]

Buffers to hold the Frequency domain overlap buffers for left and right channels, used by the FFT-filtering algorithm

4.11.4.8 volatile int readyForFFT[CIRCULAR_BUFFER_SIZE] = {0}

Variable to indicate whether a particular buffer (input audio samples) is ready for FFT-filtering

4.11.4.9 int writeBufIndex

Variable to switch between the data buffers of the Audio library

4.12 C:/DSPShieldDoc/FFT/FFT.cpp File Reference

FFT implementation.

```
#include <stdio.h>
#include "FFT.h"
#include <stdlib.h>
#include <math.h>
#include <dsplib.h>
#include "coeffdef_fft.h"
```

Macros

- #define ENABLE_FILTER

Functions

- void IR2FR (short *ir, short *fr)
- long CPLX_Mul (short *op1, short *op2)
- void downScaleData (short *data, int length, short scale)
- void upScaleData (short *data, int length, short scale)

Variables

- FFTClass FFTransform
- short buf [FFTLEN *2]

  #pragma DATA_ALIGN (32)

4.12.1 Detailed Description

FFT implementation.

4.12.2 Macro Definition Documentation

4.12.2.1 #define ENABLE_FILTER

Macro to enable/disable Filtering

4.12.3 Function Documentation

4.12.3.1 long CPLX_Mul (short * *op1,* short * *op2*)

==========

CPLX_Mul()

Description
API to perform multiplication between two complex numbers

Arguments

```
op1 - First Input to be used in Complex multiplication
op2 - Second Input to be used in Complex multiplication
```

Return Value

- The product of the complex numbers, passed as inputs to this API

Extract the real and imag parts from the input data = (real:imag)

Generate the real and imag part of the product

4.12.3.2 void downScaleData (short * *data,* int *length,* short *scale*)

==========

downScaleData()

Description
API to down scale data by an amount specified by 'scale' parameter

Arguments

```
data   - Data buffer
length - Length of data
scale  - Scaling factor
```

Return Value none

4.12.3.3 void IR2FR (short * *ir,* short * *fr*)

==========

IR2FR()

Description
API to convert samples from Real Impulse response to complex Frequency response. Real and imaginary parts are alternated

Arguments

```
ir - Input Buffer containing audio samples in Real form
fr - Output Buffer to hold the converted audio samples in complex form
```

Return Value
None

Arranging the data so that real and imaginary parts alternate. Zero padding the second half of the buffer

real

imag

zero padding

real

imag

No Scale FFT to preserve the dynamic range and resolution

4.12.3.4 void upScaleData (short * *data,* int *length,* short *scale*)

==========

upScaleData()

Description
API to up scale data by an amount specified by 'scale' parameter

Arguments

```
data   - Data buffer
length - Length of data
scale  - Scaling factor
```

Return Value none

4.12.4 Variable Documentation

4.12.4.1 short buf[FFTLEN *2]

#pragma DATA_ALIGN (32)

Buffer for FFT processing

4.12.4.2 FFTClass FFTransform

Class identifier declaration

4.13 C:/DSPShieldDoc/FFT/FFT.h File Reference

FFT library header file.

```
#include "tistdtypes.h"
#include "dsplib.h"
#include "core.h"
```

Classes

- class FFTClass
  FFT Class.

Macros

- #define FFT_REAL (0)
- #define FFT_COMPLEX (1)
- #define FFTQ15 (0)
- #define FFTQ31 (1)
- #define FFT_BAD_PARAMS (-1)
- #define FFT_SUCCESS (0)
- #define CONVERT_32_TO_16_MASK (0x0000FFFF)

- #define CONVERT_32_TO_16_SHIFT (16)
- #define Q15_MULT_SHIFT (15)
- #define ENABLE_SCALE (1)
- #define DISABLE_SCALE (0)

Variables

- FFTClass FFTransform

4.13.1 Detailed Description

FFT library header file.

4.13.2 Macro Definition Documentation

4.13.2.1 #define CONVERT_32_TO_16_MASK (0x0000FFFF)

Mask required to convert a 32 bit value to 16 bit

4.13.2.2 #define CONVERT_32_TO_16_SHIFT (16)

Shift required to convert a 32 bit value to 16 bit

4.13.2.3 #define DISABLE_SCALE (0)

Scaling is disabled

4.13.2.4 #define ENABLE_SCALE (1)

Scaling is enabled

4.13.2.5 #define FFT_BAD_PARAMS (-1)

Parameters are invalid

4.13.2.6 #define FFT_COMPLEX (1)

Fast Fourier Transform for complex inputs

4.13.2.7 #define FFT_REAL (0)

Fast Fourier Transform for real inputs

4.13.2.8 #define FFT_SUCCESS (0)

Fast Fourier Transform is successful

4.13.2.9 #define FFTQ15 (0)

Fast Fourier Transform for 16 bit inputs in Q15 format

4.13.2.10 #define FFTQ31 (1)

Fast Fourier Transform for 32 bit inputs in Q31 format

4.13.2.11 #define Q15_MULT_SHIFT (15)

Shift required to convert a 32 bit value to Q15 format

4.13.3 Variable Documentation

4.13.3.1 FFTClass FFTransform

FFT class instance extern which can used by application programs to access FFT DSP APIs

Class identifier declaration

4.14 C:/DSPShieldDoc/Interrupt/examples/dmaInterrupt/dmaInterrupt.cpp File Reference

Interrupt Example for DMA: Configures channels for data transfer.

Macros

- #define **DMA_BUFFER_SIZE** (512u)

Functions

- interrupt void dma_isr (void)
- void setup ()
- void **loop** ()

Variables

- volatile int interrupt_occurred = 0
- unsigned short * dmaSRCBuff = NULL
- unsigned short * **dmaDESTBuff** = NULL

4.14.1 Detailed Description

Interrupt Example for DMA: Configures channels for data transfer.

Configures all the DMA channels for data transfer, in interrupt mode. DMA is configured with proper source and destination address and data length. Data in the source buffer is copied into the destination buffer, using DMA. The read and write buffer contents should match with each other.

4.14.2 Function Documentation

4.14.2.1 interrupt void dma_isr (void)

Simple DMA ISR that reads the interrupt status and clears it, finally informs the setup() API that DMA transfer has completed for the current DMA channel

4.14.2.2 void setup (void)

DMA.dmaInit() creates DMA data buffers and initializes the config buffer The DMA.dmaInit() API will initialize the config structure passed to it with the default config values

Enable DMA interrupt

Enabling Interrupt

Initialize DMA

Test all the DMA channels by opening each of the channels, configuring them to transfer data from 'dmaSRCBuff' and 'dmaDESTBuff' buffers. Finally, once the DMA transfer completes compare the source and destination buffer contents

Set value to indicate that the DMA transfer has not yet completed for the current DMA channel

Initialize the DMA Source and Destination buffers

Opens the respective DMA channel

Configure the respective DMA channel

Start DMA transfer for the respective DMA channel

Wait for the respective DMA transfer to complete

The value of the variable 'interrupt_occurred' will be set in the DMA ISR, which will be triggered once the current DMA transfer completes

Close the respective DMA channel

When index will be equal to DMA_BUFFER_SIZE, then both the source and destination buffers would have matched

DMA transfer test completed for all channels

De-allocate the buffers, which were allocated by the DMA module by calling DMA.dmaInit() API

4.14.3 Variable Documentation

4.14.3.1 unsigned short* dmaSRCBuff = NULL

Variables to store Source and Destination buffer addresses used during DMA transfers

4.14.3.2 volatile int interrupt_occurred = 0

Variable to indicate that the DMA transfer has completed for the respective DMA channel

4.15 C:/DSPShieldDoc/Interrupt/examples/ioExpanderInterrupt/ioExpanderInterrupt.cpp File Reference

IO expander Interrupt Example waits for interrupt on IO expander pins.

Functions

- interrupt void INT1_isr (void)
- void setup ()
- void loop ()

Variables

- volatile int intr_flag = 0
- unsigned int **cmd** [2]
- unsigned int **readBuf** [2]
- int **status**
- unsigned int **bytesAvail**
- unsigned int **pinNum**
- unsigned int **pinVal**

4.15.1 Detailed Description

IO expander Interrupt Example waits for interrupt on IO expander pins.

Configures interrupt for IO expander and waits for the interrupt on IO expander pins. Interrupt on IO expander can be generated by changing the DIP switch states on SW1.

Note: This demo is specific for C5535 DSP shield. May not work/compile for other platforms

4.15.2 Function Documentation

4.15.2.1 interrupt void INT1_isr (void)

ISR that clears the interrupt status and disables it, finally informs the setup() API that DMA transfer has completed for the current DMA channel

4.15.2.2 void loop (void)

Read all the input pin values to clear the intterupts

4.15.2.3 void setup (void)

Configure the state of pins to output

Configure IO pins as input

Configure DIP switch pins as input

Init interrupt module

Plugin the ISR to vector table and enbale interrupts

4.15.3 Variable Documentation

4.15.3.1 volatile int intr_flag = 0

Variable to indicate that the interrupt has occurred

4.16 C:/DSPShieldDoc/OLED/bitmapDb.h File Reference

Bitmap database for OLED module.

```
#include "tistdtypes.h"
```

Classes

- struct FONT_CHAR_INFO

  This structure describes a single character's display information.
- struct FONT_INFO

  This structure describes a single font.

4.16.1 Detailed Description

Bitmap database for OLED module.

FILE NAME: bitmapDb.h FILE DESCRIPTION: Database of all bitmaps in the system

FILE CREATION DATE: 24-07-2009

Modification history:

01a,24jul09 erd written

4.17 C:/DSPShieldDoc/OLED/examples/OledBasic/OledBasic.cpp File Reference

Displays text on OLED screen & scrolls the display OLED Library.

```
#include <OLED.h>
```

Functions

- void setup ()
- void loop ()

4.17.1 Detailed Description

Displays text on OLED screen & scrolls the display OLED Library.

Displays Welcome and DSP SHIELD on OLED screen Starts scrolling the display after 5 secs Scroll direction will be changed between left and right for every 10secs

4.17.2 Function Documentation

4.17.2.1 void loop (void)

Scroll entire display screen to right

Give delay of 10 msecs between right and left scrolling

Scroll entire display screen to left

Give delay of 10 msecs between left and right scrolling

4.17.2.2 void setup (void)

Initialize the OLED module

Clears the entire display screen

Set the current display line to Line0

Set orientation of the LCD to horizontal

Displays string "Welcome" on Line0 of the display screen

Set the current display line to Line1

Displays string "DSP SHIELD" on Line1 of the display screen

4.18 C:/DSPShieldDoc/OLED/examples/OledPrint/OledPrint.cpp File Reference

Demonstrates print APIs for OLED Display OLED Library.

```
#include <OLED.h>
```

Functions

- void setup ()
- void **loop** ()

4.18.1 Detailed Description

Demonstrates print APIs for OLED Display OLED Library.

Program to demonstrate the Use of all print APIs of OLED library used for Displaying

4.18.2 Function Documentation

4.18.2.1 void setup (void)

Initialize the OLED module

Set orientation of the LCD to horizontal

Clears the entire display screen

Set the current display line to Line0

Print a string

Clears the entire display screen

Set the current display line to Line0

Print a char

Clears the entire display screen

Set the current display line to Line0

Print an int

Clears the entire display screen

Set the current display line to Line0

Print a hex number (arg2 is HEXADECIMAL)

Clears the entire display screen

Set the current display line to Line0

Print a octal number (arg2 is OCTAL)

Clears the entire display screen

Set the current display line to Line0

Print a binary number (arg2 is BINARY)

4.19 C:/DSPShieldDoc/OLED/examples/OledSerialControl/OledSerialControl.cpp File Reference

Allows User to input to Serial to change OLED display status OLED Library.

```
#include <OLED.h>
```

Functions

- void setup ()
- void **loop** ()

4.19.1 Detailed Description

Allows User to input to Serial to change OLED display status OLED Library.

Program which gives a menu list to the User to try out the different OLED APIs

4.19.2 Function Documentation

4.19.2.1 void setup (void)

limit of 13 characters on LED display

Initialize the OLED module

Clears the entire display screen

Set the current display line to Line0

Set orientation of the LCD to horizontal

Print initial instructions to Serial

Read User Option, which will be in the format 00,01,02,...,13

read 10s digit

read ones digit

Turns off automatic scrolling of the LCD

Clear the requested Line of the display screen

Read string from Serial monitor untill User enters '$' character or length of string has reached 12 characters

Set the current display line as requested by the User

Displays the string entered by User on Serial monitor

Scroll Line0 of display screen to left

Scroll Line0 of display screen to right

Scroll Line1 of display screen to left

Scroll Line1 of display screen to right

Scroll entire display screen to left

Scroll entire display screen to right

Turns off automatic scrolling of the LCD

Flips the screen vertically

Clears Line0 of the display screen

Clears Line1 of the display screen

Clears the entire display screen

Stop the Program

4.20 C:/DSPShieldDoc/OLED/examples/OledSerialMsg/OledSerialMsg.cpp File Reference

Reads Messages from the Serial port and Displays on the LCD.

```
#include <OLED.h>
```

Macros

- #define **MSG_LENGTH** (12)

Functions

- void setup ()
- void loop ()

4.20.1 Detailed Description

Reads Messages from the Serial port and Displays on the LCD.

OLED and Serial Library Example

Note: 1. Length of Messages is limited to 12 characters because of the LCD size

1. If the User wants to send a message lesser than 12 characters, enter '$' at the end of the message.

4.20.2 Function Documentation

4.20.2.1 void loop (void)

Loop to read string from Serial monitor until user enters '$' character or length of string has reached 12 characters

replace $ char with terminating char /0

Display to Serial

Display on LCD

Clears the entire display screen

Displays the string entered by User on Serial monitor

4.20.2.2 void setup (void)

Initialize the OLED module

Set the current display line to Line0

Set orientation of the LCD to horizontal

Clears the entire display screen

4.21 C:/DSPShieldDoc/OLED/OLED.cpp File Reference

OLED implementation.

```
#include "OLED.h"
```

Variables

- OLED disp

4.21.1 Detailed Description

OLED implementation.

4.21.2 Variable Documentation

4.21.2.1 OLED disp

Class identifier declaration

4.22 C:/DSPShieldDoc/OLED/OLED.h File Reference

OLED library header file.

```
#include "core.h"
#include "bitmapDb.h"
```

Classes

- class OLED
  OLED Class.

Typedefs

- typedef enum NUMBER_BASE NUMBER_BASE
  Enumeration for number format.

Enumerations

- enum NUMBER_BASE { **BINARY**, **DECIMAL**, **OCTAL**, **HEXADECIMAL** }

  Enumeration for number format.

Variables

- OLED disp

4.22.1 Detailed Description

OLED library header file.

4.22.2 Variable Documentation

4.22.2.1 OLED disp

OLED class instance extern which can used by application programs to access OLED DSP APIs

Class identifier declaration

4.23 C:/DSPShieldDoc/PLL/pll.cpp File Reference

PLL implementation.

```
#include "pll.h"
```

Variables

- PLL_Class PLL

4.23.1 Detailed Description

PLL implementation.

4.23.2 Variable Documentation

4.23.2.1 PLL_Class PLL

Defining a global PLL Class Object which can be used by the User Application

4.24 C:/DSPShieldDoc/PLL/pll.h File Reference

PLL library header file.

```
#include <stdio.h>
#include <stdlib.h>
#include "csl_pllAux.h"
#include "csl_sysctrl.h"
```

Classes

- class PLL_Class

  PLL Class.

Variables

- PLL_Class PLL

4.24.1 Detailed Description

PLL library header file.

4.24.2 Variable Documentation

4.24.2.1 PLL_Class PLL

PLL class instance extern which can used by application programs to access PLL DSP APIs

Defining a global PLL Class Object which can be used by the User Application

4.25 C:/DSPShieldDoc/RTC/examples/RTC/RTC.cpp File Reference

Configures RTC date/time & reads value.

```
#include "RTC_lib.h"
```

Functions

- void setup ()

  Configure RTC date/time.
- void loop ()

  Continuously read RTC date/time.

Variables

- RTCDate **rtcDateRead**
- RTCTime **rtcTimeRead**
- int **result**

4.25.1 Detailed Description

Configures RTC date/time & reads value.

RTC Demo

This demo configures the RTC date, time and reads the updated date and time for every 1 secs. Date and time value read from RTC library shall be displayed on serial monitor.

4.25.2 Function Documentation

4.25.2.1 void loop (void)

Continuously read RTC date/time.

if RTC date/time was successfully configured

Read RTC time

Give some delay for the time to get updated

4.25.2.2 void setup (void)

Configure RTC date/time.

Print the format of date which will be displayed on serial monitor

Resets and configures RTC time registers, enables RTC interrupts

Set the RTC time

Set the RTC date

RTC starts counting the time

Give some delay for the time to get updated

Read the date

4.26 C:/DSPShieldDoc/SAR/SAR.cpp File Reference

SAR implementation.

```
#include <core.h>
#include <SAR.h>
```

Functions

- void SAR_LOG_MSG_PRINT (char *printString)

  #define ENABLE_SERIAL_MSGS
- void SAR_LOG_MSG_PRINT (int integer)

Variables

- SAR_Class SAR

4.26.1 Detailed Description

SAR implementation.

4.26.2 Function Documentation

4.26.2.1 void SAR_LOG_MSG_PRINT (char * *printString*)

#define ENABLE_SERIAL_MSGS

Macro to enable the Print messages to be displayed on the Serial

SAR_LOG_MSG_PRINT()

Description
API to display debug messages

Arguments

```
printString - String to be displayed
```

Return Value
None

==========

SAR_LOG_MSG_PRINT()

Description
API to display debug messages

Arguments

```
printString - String to be displayed
```

Return Value
None

4.26.2.2 void SAR_LOG_MSG_PRINT (int *integer*)

==========

SAR_LOG_MSG_PRINT()

Description
API to display debug messages

Arguments

```
integer - Integer to be displayed
```

Return Value
None

4.26.3 Variable Documentation

4.26.3.1 SAR_Class SAR

Defining a global SAR Class Object which can be used by the User Application

4.27 C:/DSPShieldDoc/SAR/SAR.h File Reference

SAR library header file.

```
#include <stdio.h>
#include "csl_sar.h"
```

Classes

- class SAR_Class
  SAR Class.

Macros

- #define SAR_POLL_MODE (0)
- #define SAR_INTERRUPT_MODE (1)
- #define SAR_DMA_MODE (2)
- #define SAR_CHANNEL0 (0)
- #define SAR_CHANNEL1 (1)
- #define SAR_CHANNEL2 (2)
- #define SAR_CHANNEL3 (3)
- #define SAR_CHANNEL4 (4)
- #define SAR_CHANNEL5 (5)
- #define SAR_CONTINUOUS_CONVERSION (0)
- #define SAR_SINGLE_CONVERSION (1)

Variables

- SAR_Class SAR

4.27.1 Detailed Description

SAR library header file.

4.27.2 Macro Definition Documentation

4.27.2.1 #define SAR_CHANNEL0 (0)

Macro to indicate Channel No 0 for SAR

4.27.2.2 #define SAR_CHANNEL1 (1)

Macro to indicate Channel No 1 for SAR

4.27.2.3 #define SAR_CHANNEL2 (2)

Macro to indicate Channel No 2 for SAR

4.27.2.4 #define SAR_CHANNEL3 (3)

Macro to indicate Channel No 3 for SAR

4.27.2.5 #define SAR_CHANNEL4 (4)

Macro to indicate Channel No 4 for SAR

4.27.2.6 #define SAR_CHANNEL5 (5)

Macro to indicate Channel No 5 for SAR

4.27.2.7 #define SAR_CONTINUOUS_CONVERSION (0)

Macro to indicate SAR to perform Continous Conversion

4.27.2.8 #define SAR_DMA_MODE (2)

Macro to indicate the mode of SAR as DMA Mode

4.27.2.9 #define SAR_INTERRUPT_MODE (1)

Macro to indicate the mode of SAR as Interrupt Mode

4.27.2.10 #define SAR_POLL_MODE (0)

Macro to indicate the mode of SAR as Polled Mode

4.27.2.11 #define SAR_SINGLE_CONVERSION (1)

Macro to indicate SAR to perform Single Conversion

4.27.3 Variable Documentation

4.27.3.1 SAR_Class SAR

SAR class instance extern which can used by application programs to access SAR DSP APIs

Defining a global SAR Class Object which can be used by the User Application

4.28 C:/DSPShieldDoc/SD/chk_mmc.h File Reference

Header file for MMC checking routines.

```
#include "ata.h"
#include "csl_mmcsd_ataIf.h"
#include "ata_.h"
```

Classes

- struct PARTITION_TABLE
- struct MBR_struct
- struct BR_struct

Macros

- #define HD_BOOT_RECORD 0
- #define FD_BOOT_RECORD 1

Enumerations

- enum MMC_ERR_int16 {
  MMC_ERR_NONE, **MMC_ERR_BAD_CSD_FILE_FORMAT**, **MMC_ERR_CARD_NOT_READABLE**, **MM↩C_ERR_MBR_BAD_NUM_PART_SECTORS**,
  MMC_ERR_MBR_BAD_PARTITIONS_234, **MMC_ERR_MBR_DSKSIZE_MISMATCH**, MMC_ERR_BR_↩BAD_JMP_OPCODE, **MMC_ERR_BR_BAD_BYTES_PER_SECTOR**,
  MMC_ERR_BR_BAD_SECTORS_PER_CLUSTER, **MMC_ERR_BR_BAD_RESERVED_SECTORS**, **MM↩**

C_ERR_BR_BAD_NUM_OF_ROOT_ENTRIES, **MMC_ERR_BR_BAD_NUM_PART_SECTORS**, **MMC_ERR_BR_BAD_SECTORS_PER_FAT**, **MMC_ERR_BR_BAD_FILE_SYS_TYPE**, **MMC_ERR_BR_BAD_SIGNATURE** }

Functions

- MMC_ERR_int16 chk_mmc (AtaState *pAtaDrive, unsigned int *disk_type)
- MMC_ERR_int16 Check_boot_record (unsigned long boot_record_sector, unsigned long ref_num_of_sectors, unsigned int boot_record_type, AtaState *pAtaDrive, BR_struct *pBootRecord)
- AtaUint32 getMMCSize (AtaState *pAtaDrive)
- AtaError mmc_format (AtaState *pAtaDrive, AtaUint16 *MBRptr, AtaUint16 *BRptr)

4.28.1 Detailed Description

Header file for MMC checking routines.

Copyright 2004 by Texas Instruments Incorporated. All rights reserved. Property of Texas Instruments Incorporated. Restricted rights to use, duplicate or disclose this code are granted through contract.

This is proprietary information, not to be published – TI INTERNAL DATA Copyright (C) 2003, Texas Instruments, Inc. All Rights Reserved.

Title: Header file for the MMC sanity check routines

4.28.2 Macro Definition Documentation

4.28.2.1 #define FD_BOOT_RECORD 1

FD boot record

4.28.2.2 #define HD_BOOT_RECORD 0

<pramod : : 18-Dec-02>

HD boot record

4.28.3 Enumeration Type Documentation

4.28.3.1 enum MMC_ERR_int16

Return Values of chk_mmc.

Enumerator

MMC_ERR_BR_BAD_JMP_OPCODE <susmit : : 04-Mar-2003>

4.28.4 Function Documentation

4.28.4.1 MMC_ERR_int16 Check_boot_record (unsigned long *boot_record_sector,* unsigned long *ref_num_of_sectors,* unsigned int *boot_record_type,* AtaState * *pAtaDrive,* BR_struct * *pBootRecord*)

MMC_ERR_int16 Check_boot_record(unsigned long boot_record_sector, unsigned long ref_num_of_sectors, unsigned int boot_record_type, AtaState *pAtaDrive, BR_struct *pBootRecord

boot_record_sector -> The sector number to fetch the BR from. ref_num_of_sectors -> The reference number of sectors on disk to validate BR against. boot_record_type -> Indicates whether this is floppy type or hard disk type

boot record. pAtaDrive -> Pointer to initialised AtaState structure. pBootRecord -> 512 byte buffer to read boot record.

Function : MMC_ERR_int16 Check_boot_record(unsigned long boot_record_sector, unsigned long ref_num_of_↩ sectors, unsigned int boot_record_type, AtaState *pAtaDrive, BR_struct *pBootRecord

Parameters:

boot_record_sector -> The sector number to fetch the BR from. ref_num_of_sectors -> The reference number of sectors on disk to validate BR against. boot_record_type -> Indicates whether this is floppy type or hard disk type boot record. pAtaDrive -> Pointer to initialised AtaState structure. pBootRecord -> 512 byte buffer to read boot record.

Return Values:

MMC_ERR_CARD_NOT_READABLE -> Unable to read card using ATA_readSector() MMC_ERR_BR_BAD_J↩ MP_OPCODE -> The first byte of BR was not 0xEB OR the third byte was not 0x90 MMC_ERR_BR_BAD_BYT↩ ES_PER_SECTOR -> The BR bytes per sector was not 512. MMC_ERR_BR_BAD_SECTORS_PER_CLUSTER -> The sectors per cluster is not a power of 2. MMC_ERR_BR_BAD_RESERVED_SECTORS -> Boot sector + reserved sector is not pointing to FAT table. (0x fff8 ffff) MMC_ERR_BR_BAD_NUM_OF_FATS -> The number of FATs is not 2. MMC_ERR_BR_BAD_NUM_OF_ROOT_ENTRIES -> The number of root directory entries is not 512. MMC_ERR_BR_BAD_NUM_PART_SECTORS -> * For boot_record_type = HD_BOOT_RECORD The number of partition sectors does not match with MBR (ref_num_of_sectors)

- For boot_record_type = FD_BOOT_RECORD The number of partition sectors is greater than sectors in disk from CSD (ref_num_of_sectors). Note: If the function parameter ref_num_of_sectors > 0xFFFF the extended number of sectors in partition is used in place of number of partition sectors.

MMC_ERR_BR_BAD_MEDIA_DESCRIPTOR -> The media descriptor was not 0xF8. Win2k sets this field as 0xF8 for both hard disk type and floppy type formats. MMC_ERR_BR_BAD_SECTORS_PER_FAT -> The value was not between 1 and 256.

MMC_ERR_BR_BAD_EXTENDED_BOOT_SIGNATURE -> The extended boot signature is not 0x29.

MMC_ERR_BR_BAD_FILE_SYS_TYPE -> The filesystem type is not " FAT16"

MMC_ERR_BR_BAD_SIGNATURE -> The boot record signature was not 0x55AA Read Boot Record from sector number = boot_record_sector

ata_error should be ATA_ERROR_NONE unless the card is damaged and unreadable

Check first and third byte of boot record for valid opcodes

<pramod : Added check for second possible jump instruction 0xE9 : 24-Apr-2003>

Verify that bytes per sector to be 512

Verify that sectors per cluster is a power of 2

<susmit : Cluster size should not exceed 32Kbytes : 04-Mar-2003>

Verify that data at boot sector + reserved sector is 0xFFF8 FFFF

<susmit : Commented off this code for now but should be enabled : 04-Mar-2003>

MS FAT Specs say that this field should never be anything other than 1 for FAT12/16. But a particular customer's images puts some other values in here. Need to discuss these

if(l_reserved_sectors!=1) return MMC_ERR_BR_BAD_RESERVED_SECTORS;

<susmit : _AtaReadDoubleWord() should not be called directly, hence replaced the call with ATA_readSector() : 04-Mar-2003>

l_fat_signature = _AtaReadDoubleWord(boot_record_sector + l_reserved_sectors , pAtaDrive, 0);

The FAT signature must be 0xFFFFFFF8 . But if the volume is not dismounted correctly , windows sets the dirty bit to zero which makes the value appear as 0x7FFFFFF8

Verify that the number of partition sectors or extended number of partition sectors is valid

<susmit : Check whether the format type is FAT16 : 04-Mar-2003>

Verify that number of root directory entries is 512

<susmit : This might be different for some disks but for a FAT16 MMC this check should be okay : 04-Mar-2003>

<susmit : If l_num_partition_sectors is not equal to sectors_in_partition_from_MBR for HD format and l_num_↩ partition_sectors is not equal to disk_sectors_from_csd for FD format, return error : 04-Mar-2003>

Verify that sectors per fat is between 1 to 256

<susmit : We should verify that the number of clusters match the no. of sectors per fat calculations : 04-Mar-2003>

Verify that boot record signature is 0x55AA

4.28.4.2 MMC_ERR_int16 chk_mmc (AtaState * *pAtaDrive,* unsigned int * *disk_type*)

Function prototypes

<susmit : Added a new parameter unsigned int * disk_type : 04-Mar-2003>

<susmit : Added a new parameter, unsigned int *disk_type : 04-Mar-2003> <susmit : Variables to hold the error value and boot record type : 04-Mar-2003>

For the cards with csd ver2.0 it is not possible to determine the file system format

Get total number of sectors on MMC from CSD register

Try to read Master Boot Record from sector

ata_error should be ATA_ERROR_NONE unless the card is damaged and unreadable

Check for all partition entries of partitions 2,3,4 to be zero

return MMC_ERR_MBR_BAD_PARTITIONS_234;

<susmit : Set the error : 04-Mar-2003>

Verify number of sectors in partition as reported by MBR is less than the total number of sectors reported by CSD register

<susmit : Set the error : 04-Mar-2003>

Calculate the sector number of the boot record

<susmit : The number of sectors in partition plus the number of reserved sectors must be equal to the number of sectors present in the disk as indicated by the CSD register : 04-Mar-2003>

<susmit : If no errors upto this, then this might be a disk with both MBR & BR. Otherwise set boot sector as zero & check : 04-Mar-2003>

floppy-like file format (without partition table)

Now check Boot Record

<susmit : Can be HD or FD boot record : 04-Mar-2003>

4.28.4.3 AtaUint32 getMMCSize (AtaState * *pAtaDrive*)

getMMCSize(AtaState *pAtaDrive)

This function will returns total number of sectors in the disk, Not the actual size of the disk

512 byte buffer for checking boot record

NOTE: This function will returns total number of sectors in the disk, Not the actual size of the disk unsigned int i = 0;

Uint16 sectorSize = 0;

Read the Card Specific Data(CSD)

Get the CSD card structure version; Size calculataion will be different for different versions of CSD structures

CSD structure version is 1.x

mwei total capacity computation

CSD structure version is 2.x

sectorSize = 1 << read_bl_len;

Calculate the size of the disk

For CSD ver2.0 size will be in KBytes, multiply with 1024 to convert to bytes

mwei CSD ver2.0 size is in KB not byte

1KB has two sectors (512 Byte)

Return the valus of Number of sectors

4.28.4.4 AtaError mmc_format (AtaState * *pAtaDrive,* AtaUint16 * *MBRptr,* AtaUint16 * *BRptr*)

mmc_format(AtaState *pAtaDrive, AtaUint16 *MBRptr, AtaUint16 *BRptr)

This function will format the disk

4.29 C:/DSPShieldDoc/SD/examples/DisplayContents/DisplayContents.cpp File Reference

Directory Browse Demo: query directory & subdirectories.

```
#include "SD.h"
```

Functions

- void display (void *Handle)

  recursively query contents of directory, calling display on subdirectories
- void setup ()

  Open SD card, get file handle to Root directory.
- void **loop** ()

4.29.1 Detailed Description

Directory Browse Demo: query directory & subdirectories.

In this demo a file handle to the root directory is obtained. The contents of the root directory and its subsequent sub-directories are queried recursively and are displayed on the Serial.

Note: SD Library requires SD card to be formatted in a specific format for proper operation. It is recommended to use SD formatter from SD card org or HP format tool for formatting the SD card

4.29.2 Function Documentation

4.29.2.1 void display (void * *Handle*)

recursively query contents of directory, calling display on subdirectories

Open next file in current directory using File.openNextFile() API

If the child is a directory, set up to query the files it contains

Since its a directory display the contents under the respective directory

If the child is a file, print the name and size

print the total number of contained files and directories at the end

4.29.2.2 void setup (void)

Open SD card, get file handle to Root directory.

Get a File handle to the Root directory

query subdirectories

4.30 C:/DSPShieldDoc/SD/examples/Exists/Exists.cpp File Reference

Checks whether a file is on the SD card.

```
#include "SD.h"
```

Functions

- void setup ()
- void **loop** ()

4.30.1 Detailed Description

Checks whether a file is on the SD card.

SD.exists() Demo

This demo checks whether the file "DEMO.TXT" exists under the root directory on the Card, using SD.exists() API

Note: SD Library requires SD card to be formatted in a specific format for proper operation. It is recommended to use SD formatter from SD card org or HP format tool for formatting the SD card

4.30.2 Function Documentation

4.30.2.1 void setup (void)

Initialize SD Module

Check whether the file "DEMO.TXT" exists or not under root directory

4.31 C:/DSPShieldDoc/SD/examples/FileFolderCreation/FileFolderCreation.cpp File Reference

Demo creates Files and Folders on SD card.

```
#include "SD.h"
```

Functions

- void setup ()

Create Files & Folders on SD card.

- void createFiles ()
- void **loop** ()

4.31.1 Detailed Description

Demo creates Files and Folders on SD card.

This demo creates the directory path "DIR1/DIR2/DIR3/DIR4/DIR5" and creates 4 files under each directory in the path created before.

In the first part of this demo, a directories chain with path as 'DIR1/DIR2/DIR3/DIR4/DIR5' is created. In the second part, 4 files are created under each directory in the directory path 'DIR1/DIR2/DIR3/DIR4/DIR5' and some data is written in these files.

Note: SD Library requires SD card to be formatted in a specific format for proper operation. It is recommended to use SD formatter from SD card org or HP format tool for formatting the SD card

4.31.2 Function Documentation

4.31.2.1 void createFiles ( )

Create 4 files under each directory in the path 'DIR1/DIR2/DIR3/DIR4/DIR5'

Create 4 files – dir1, dir2, dir3, dir4

create new file

write short string to new file

if it fails to write

4.31.2.2 void setup (void)

Create Files & Folders on SD card.

Create the directory path using mkdir(char ∗dirPath)

if successful

4.32 C:/DSPShieldDoc/SD/examples/Filesize/Filesize.cpp File Reference

Find the size of a file on the SD card.

```
#include "SD.h"
```

Functions

- void setup ()

  find the filesize of the each file
- void **loop** ()

4.32.1 Detailed Description

Find the size of a file on the SD card.

File Size Demo

Demo to find the size of a file using File.size() API. This demo finds the file size of each file under root.

Note: SD Library requires SD card to be formatted in a specific format for proper operation. It is recommended to use SD formatter from SD card org or HP format tool for formatting the SD card

4.32.2 Function Documentation

4.32.2.1 void setup (void)

find the filesize of the each file

Initialize SD Module

open the first file

Get the File name of the file

Get the size of the file using File.size()

Open next file in current directory using openNextFile()

4.33 C:/DSPShieldDoc/SD/examples/Peek/Peek.cpp File Reference

Demo keep reading the same character from a file using File.peek() API.

```
#include "SD.h"
```

Functions

- void setup ()
- void **loop** ()

4.33.1 Detailed Description

Demo keep reading the same character from a file using File.peek() API.

File.peek() demo

Note: SD Library requires SD card to be formatted in a specific format for proper operation. It is recommended to use SD formatter from SD card org or HP format tool for formatting the SD card

4.33.2 Function Documentation

4.33.2.1 void setup (void)

write and open file

Read the character using of File.peek() API

Any number of calls to File.peek() will return the same character

4.34 C:/DSPShieldDoc/SD/examples/Position/Position.cpp File Reference

Print position of file cursor.

```
#include "SD.h"
```

Functions

- void setup ()

  Opens SD, prints position of file cursor.
- void **loop** ()

4.34.1 Detailed Description

Print position of file cursor.

File Cursor Position demo

Demo to print the current position of the file cursor using File.seek() and File.position() APIs

Note: SD Library requires SD card to be formatted in a specific format for proper operation. It is recommended to use SD formatter from SD card org or HP format tool for formatting the SD card

4.34.2 Function Documentation

4.34.2.1 void setup (void)

Opens SD, prints position of file cursor.

Seek to the beginning of the file and display the current position

Seek to the end of the file and display the current position

4.35 C:/DSPShieldDoc/SD/examples/ReadWrite/ReadWrite.cpp File Reference

Reads data from Serial and writes to SD card.

```
#include "SD.h"
```

Functions

- void setup ()

  Read from Serial and write to SD card.
- void **loop** ()

4.35.1 Detailed Description

Reads data from Serial and writes to SD card.

File Read Write Demo

This demo continuously reads data from serial monitor and writes it on to a file "transfer.txt" on the SD card. The demo stops reading data from serial monitor when the user types "end" string.

Note: SD Library requires SD card to be formatted in a specific format for proper operation. It is recommended to use SD formatter from SD card org or HP format tool for formatting the SD card

4.35.2 Function Documentation

4.35.2.1 void setup (void)

Read from Serial and write to SD card.

Create and open the file to write to

Reading data continuously from serial till user enters "end"

Write the data entered by the User on Serial using File.write(char *printString)

Write the data entered by the User on Serial using File.write(char *printString)

Display size of file

Displaying File contents before exiting the demo

Read the data entered written to the file using File.read(char *readString, length)

If the file size is greater than 512 bytes, can read up to 512 bytes at a time

If the file is less than 512 bytes, read the number of bytes it has

If no bytes were read, there was an error, otherwise print the string

4.36 C:/DSPShieldDoc/SD/SD.cpp File Reference

SD/File implementation.

```
#include <SD.h>
#include <core.h>
```

Classes

- class fileNodesList

  Node of the Linked List to hold the details of all the Opened files.

Macros

- #define CSL_SD_CLOCK_MAX_KHZ (20000u)
- #define CSL_MMCSD_ATA_BUF_SIZE (512u)

  #define ENABLE_SERIAL_MSGS
- #define WRITE_CACHE_SIZE (512)

Functions

- void LOG_MSG_print (char *printString)

  API to print Debug messages.

Variables

- fileNodesList * fileListHeadNode
- fileNodesList * fileListLastNode
- SD_Class SD
- AtaUint16 AtaWrBuf [CSL_MMCSD_ATA_BUF_SIZE]
- char writeCacheBuffer [WRITE_CACHE_SIZE+1]

4.36.1 Detailed Description

SD/File implementation.

4.36.2 Macro Definition Documentation

4.36.2.1 #define CSL_MMCSD_ATA_BUF_SIZE (512u)

#define ENABLE_SERIAL_MSGS

Macro to enable the Print messages to be displayed on the Serial

Macro to indicate the Size of Data buffer used by ATA File System for file Read and Write Operations

4.36.2.2 #define CSL_SD_CLOCK_MAX_KHZ (20000u)

Macro to indicate the maximum clock value for the SD/MMC Card

4.36.2.3 #define WRITE_CACHE_SIZE (512)

Macro to indicate write cache size

4.36.3 Function Documentation

4.36.3.1 void LOG_MSG_print (char * *printString*)

API to print Debug messages.

This API will print the messages to either the Serial or the Output Console

4.36.4 Variable Documentation

4.36.4.1 AtaUint16 AtaWrBuf[CSL_MMCSD_ATA_BUF_SIZE]

Data Buffer used for Read and Write Operations by the ATA File System

4.36.4.2 fileNodesList* fileListHeadNode

Pointer to the first node in the Linked List of File Nodes

4.36.4.3 fileNodesList* fileListLastNode

Pointer to the last node in the Linked List of File Nodes

4.36.4.4 SD_Class SD

Defining a global SD Class Object which can be used by the User Application

4.36.4.5 char writeCacheBuffer[WRITE_CACHE_SIZE+1]

Data Buffer used for DSP API write operations

4.37 C:/DSPShieldDoc/SD/SD.h File Reference

SD library header file.

```
#include <string.h>
#include <stdio.h>
#include <ctype.h>
#include "ata.h"
#include "csl_mmcsd.h"
#include "csl_sysctrl.h"
#include "csl_pll.h"
#include "chk_mmc.h"
```

Classes

- class File

  File Class.
- class SD_Class

  SD Card Class.

Typedefs

- typedef enum FILE_MODE FILE_MODE

  Enum to indicate the Mode of the file for opening.
- typedef enum NUMBER_FORMAT_BASE NUMBER_FORMAT_BASE

  Enum to indicate base format for an integer.

Enumerations

- enum FILE_MODE { **FILE_READ**, **FILE_WRITE**, **FILE_APPEND** }

  Enum to indicate the Mode of the file for opening.
- enum NUMBER_FORMAT_BASE { **FILE_BIN**, **FILE_OCT**, **FILE_DEC**, **FILE_HEX** }

  Enum to indicate base format for an integer.

Variables

- SD_Class SD

4.37.1 Detailed Description

SD library header file.

4.37.2 Typedef Documentation

4.37.2.1 typedef enum FILE_MODE FILE_MODE

Enum to indicate the Mode of the file for opening.

Note: SD Library requires SD card to be formatted in a specific format for proper operation. It is recommended to use SD formatter from SD card org or HP format tool for formatting the SD card

4.37.3 Enumeration Type Documentation

4.37.3.1 enum FILE_MODE

Enum to indicate the Mode of the file for opening.

Note: SD Library requires SD card to be formatted in a specific format for proper operation. It is recommended to use SD formatter from SD card org or HP format tool for formatting the SD card

4.37.4 Variable Documentation

4.37.4.1 SD_Class SD

SD class instance extern which can used by application programs to access SD DSP APIs

Defining a global SD Class Object which can be used by the User Application

4.38 C:/DSPShieldDoc/Serial/examples/Find/Find.cpp File Reference

Finds a user-generated target string in a second user-generated string (from Serial)

Functions

- void setup ()
- void **loop** ()

4.38.1 Detailed Description

Finds a user-generated target string in a second user-generated string (from Serial)

Serial.find() Demo

This demo first reads a string of length 5 and then keeps reading data from serial, until the User types the first string or the serial.read() API times out waiting for the User input

4.38.2 Function Documentation

4.38.2.1 void setup (void)

Ask for the target string from the User

Add terminating char to string

find the target string in the serial string

4.39 C:/DSPShieldDoc/Serial/examples/Finduntil/Finduntil.cpp File Reference

Reads user input to Serial until string is found or user terminates.

Functions

- void setup ()

  Read string from Serial until comparison string is found.
- void **loop** ()

4.39.1 Detailed Description

Reads user input to Serial until string is found or user terminates.

Serial.findUntil() Demo

This demo first reads a string of length 5 and then keeps reading data from serial, until the User types the first string or the terminating string ("@") or the serial.read() API times out waiting for the User input

4.39.2 Function Documentation

4.39.2.1 void setup (void)

Read string from Serial until comparison string is found.

Ask for the target string from the User

read 5 char string from Serial

add string termination char

Ask for the longer string in which to find the target string

Find target string in Serial string

4.40 C:/DSPShieldDoc/Serial/examples/ParseInt/ParseInt.cpp File Reference

Reads data from serial to form a int value.

Functions

- void setup ()
- void **loop** ()

4.40.1 Detailed Description

Reads data from serial to form a int value.

Serial.parseInt() Demo

This demo keeps reading data from serial, until the User enters a int value followed by a non-numeric character or the serial.read() API times out waiting for the User input

4.40.2 Function Documentation

4.40.2.1 void setup (void)

Ask for the input (integer value) from the User (from serial)

Print the integer value to serial

4.41 C:/DSPShieldDoc/Serial/examples/PrintFormat/PrintFormat.cpp File Reference

Prints and int in various base representations Serial Print Format Demo.

Functions

- void setup ()

  Print numbers to serial.
- void **loop** ()

4.41.1 Detailed Description

Prints and int in various base representations Serial Print Format Demo.

This demo prints an integer in various numeric format such as: binary, decimal, octal and hexadecimal using Serial.println(value, format) API

4.41.2 Function Documentation

4.41.2.1 void setup (void)

Print numbers to serial.

Binary representation uses BIN as format

Decimal representation uses DEC as format

Octal representation uses OCT as format

Hexadecimal representation uses HEX as format

4.42 C:/DSPShieldDoc/Serial/examples/Read/Read.cpp File Reference

Reads & Writes char on Serial display.

Functions

- void setup ()
- void **loop** ()

4.42.1 Detailed Description

Reads & Writes char on Serial display.

Serial Read and Write Demo

This demo read a single character from the Serial and displays it back on the Serial

4.42.2 Function Documentation

4.42.2.1 void setup (void)

Setup Serial to read user input & print values Setup Serial to prompt user for input

Read Serial char value

Print Value to Serial from user input

4.43 C:/DSPShieldDoc/Serial/examples/ReadBytes/ReadBytes.cpp File Reference

Reads a string of length 5 from the Serial and displays it back on the Serial.

Functions

- void setup ()
- void **loop** ()

4.43.1 Detailed Description

Reads a string of length 5 from the Serial and displays it back on the Serial.

Serial.readBytes() Demo

4.43.2 Function Documentation

4.43.2.1 void setup (void)

Setup Serial to read user input & print values Initialize buffer to hold user input

Setup Serial to prompt user for input string

Read string from serial

Print string to serial

4.44 C:/DSPShieldDoc/Serial/examples/ReadBytesUntil/ReadBytesUntil.cpp File Reference

Reads string from Serial until terminating char Serial.readBytesUntil() Demo.

Functions

- void setup ()
- void **loop** ()

4.44.1 Detailed Description

Reads string from Serial until terminating char Serial.readBytesUntil() Demo.

This demo reads a string from serial, until the User types a string of length 50 or types the terminating character ('@')

4.44.2 Function Documentation

4.44.2.1 void setup (void)

Setup Serial to read user input & print values Initialize Buffer to hold String

Setup Serial to prompt user for input string

readBytesUntil API reads string until terminating char or buffer length

Print output to Serial

4.45 C:/DSPShieldDoc/SPI/examples/Arduino_SPI/Arduino_SPI.cpp File Reference

SPI and Arduino Communication demo.

```
#include <SPI.h>
```

Functions

- void setup (void)
- void loop (void)

4.45.1 Detailed Description

SPI and Arduino Communication demo.

This code should be flashed to the Arduino Uno board

Sends data (value 0-2) to the DSP Shield to blink that particular LED on the DSP Shield

If value sent is 0, LED0 will be ON on DSP Shield If value sent is 1, LED1 will be ON on DSP Shield If value sent is 2, LED2 will be ON on DSP Shield

4.45.2 Function Documentation

4.45.2.1 void loop (void)

Main loop - Send the value to DSP Shield to blink the LED Send the value

Wait for the DSP Shield to read the data that was sent

4.45.2.2 void setup (void)

Configures Arduino as Slave for SPI communication with DSP Shield Value should be sent on master in slave out (MISO) line

Configure SPI for Slave mode

4.46 C:/DSPShieldDoc/SPI/SPI.cpp File Reference

SPI implementation.

```
#include <core.h>
#include <SPI.h>
```

Functions

- void SPI_LOG_MSG_PRINT (char *printString)

  #define ENABLE_SERIAL_MSGS

Variables

- SPI_Class SPI

4.46.1 Detailed Description

SPI implementation.

4.46.2 Function Documentation

4.46.2.1 void SPI_LOG_MSG_PRINT (char * *printString*)

#define ENABLE_SERIAL_MSGS

Macro to enable the Print messages to be displayed on the Serial

SPI_LOG_MSG_PRINT()

Description
API to display debug messages
Arguments

```
printString - String to be displayed
```

Return Value
None

==========

SPI_LOG_MSG_PRINT()

Description
API to display debug messages
Arguments

```
printString - String to be displayed
```

Return Value
None

4.46.3 Variable Documentation

4.46.3.1 SPI_Class SPI

Defining a global SPI Class Object which can be used by the User Application

4.47 C:/DSPShieldDoc/SPI/SPI.h File Reference

SPI library header file.

```
#include <stdio.h>
#include "csl_spi.h"
```

Classes

- class SPI_Class

  SPI Class.

Macros

- #define SPI_CLOCK_DIV2 (2)
- #define SPI_CLOCK_DIV4 (4)
- #define SPI_CLOCK_DIV8 (8)
- #define SPI_CLOCK_DIV16 (16)
- #define SPI_CLOCK_DIV32 (32)
- #define SPI_CLOCK_DIV64 (64)
- #define SPI_CLOCK_DIV128 (128)
- #define SPI_CLK_DIV (25)
- #define SPI_FRAME_LENGTH (1)
- #define SPI_MODE0 (0)
- #define SPI_MODE1 (1)
- #define SPI_MODE2 (2)
- #define SPI_MODE3 (3)
- #define LSBFIRST (0)
- #define MSBFIRST (1)

Variables

- SPI_Class SPI

4.47.1 Detailed Description

SPI library header file.

4.47.2 Macro Definition Documentation

4.47.2.1 #define LSBFIRST (0)

Macro to indicate the Order of Data transfer as LSB first

4.47.2.2 #define MSBFIRST (1)

Macro to indicate the Order of Data transfer as MSB first

4.47.2.3 #define SPI_CLK_DIV (25)

SPI Clock Divisor

4.47.2.4 #define SPI_CLOCK_DIV128 (128)

Macro for SPI clock divider value 128

4.47.2.5 #define SPI_CLOCK_DIV16 (16)

Macro for SPI clock divider value 16

4.47.2.6 #define SPI_CLOCK_DIV2 (2)

Macro for SPI clock divider value 2

4.47.2.7 #define SPI_CLOCK_DIV32 (32)

Macro for SPI clock divider value 32

4.47.2.8 #define SPI_CLOCK_DIV4 (4)

Macro for SPI clock divider value 4

4.47.2.9 #define SPI_CLOCK_DIV64 (64)

Macro for SPI clock divider value 64

4.47.2.10 #define SPI_CLOCK_DIV8 (8)

Macro for SPI clock divider value 8

4.47.2.11 #define SPI_FRAME_LENGTH (1)

SPI Frame length

4.47.2.12 #define SPI_MODE0 (0)

Macro to indicate SPI mode 0

4.47.2.13 #define SPI_MODE1 (1)

Macro to indicate SPI mode 1

4.47.2.14 #define SPI_MODE2 (2)

Macro to indicate SPI mode 2

4.47.2.15 #define SPI_MODE3 (3)

Macro to indicate SPI mode 3 The above modes are for the following Clock Polarity(CPOL) and Clock

Phase(CPHA) Selections

MODE CPOL CPHA

SPI_MODE0 0 0 SPI_MODE1 0 1 SPI_MODE2 1 0

SPI_MODE3 1 1

4.47.3 Variable Documentation

4.47.3.1 SPI_Class SPI

SPI class instance extern which can be used by application programs to access SPI DSP APIs

Defining a global SPI Class Object which can be used by the User Application

4.48 C:/DSPShieldDoc/Timers/examples/Timer1/Timer1.cpp File Reference

Timer instance 1 demo: verifies if GPT1 decrements counter.

```
#include <Timers.h>
```

Functions

- void setup ()

  Set up timer, start it, verify that the count decreases.
- void **loop** ()

4.48.1 Detailed Description

Timer instance 1 demo: verifies if GPT1 decrements counter.

Time instance 1 demo: Verifies whether the GPT1 is decrementing the counter or not. GPT is configured with a count value of 0xFFFF and pre-scaler divider value of 256. GPT1 is started and counter value is read. After few cycles of delay GPT1 counter value is read again. Both the counter values are compared to verify whether the second count value is less than the first counter value or not and the test passes if it is true.

This sketch dmonstrates usage of function Timer.configTimer

4.48.2 Function Documentation

4.48.2.1 void setup (void)

Set up timer, start it, verify that the count decreases.

Open the GPT module for GPT 0 instance

Configure GPT module

Start the Timer

Read the Timer Count

Give Some Delay

Read the Timer Count Again

Compare the timer count to verify whether the timer is counting or not

Stop The Timer

Close the GPT module

4.49 C:/DSPShieldDoc/Timers/examples/Timer2/Timer2.cpp File Reference

Timer instance 2 demo: verifies if GPT2 decrements counter.

```
#include <Timers.h>
```

Functions

- void setup ()

  Set up timer, start it, verify that the count decreases.

- void **loop** ()

4.49.1 Detailed Description

Timer instance 2 demo: verifies if GPT2 decrements counter.

This test verifies whether the GPT2 is decrementing the counter or not. GPT2 is configured with a count value of 0x20000 and pre-scaler divider value of 2. GPT2 is started and counter value is read. After few cycles of delay GPT2 counter value is read again. Both the counter values are compared to verify whether the second count value is less than the first counter value or not and the test passes if it is true.

This sketch dmonstrates usage of function Timer.initialize

4.49.2 Function Documentation

4.49.2.1 void setup (void)

Set up timer, start it, verify that the count decreases.

Open the GPT module for GPT 2 instance

Configure GPT period

Start the Timer

Read the Timer Count

Give Some Delay

Read the Timer Count Again

Compare the timer count to verify whether the timer is counting or not

Stop The Timer

Close the GPT module

4.50 C:/DSPShieldDoc/Timers/examples/WatchdogTimer/WatchdogTimer.cpp File Reference

Watchdog timer demo:

```
#include <Timers.h>
```

Functions

- void setup ()

  Configures Timer and services it, then resets.
- void **loop** ()

4.50.1 Detailed Description

Watchdog timer demo:

Configures watchdog timer, services it for some cycles, then stops servicing the watchdog which should reset the DSP shield board

4.50.2 Function Documentation

4.50.2.1 void setup (void)

Configures Timer and services it, then resets.

Open the WDTIM module

Start the watch dog timer

Stop the watch dog timer

Start the watch dog timer

Servicing the watchdog timer for 256 cycles program should be running properly as long as watchdog is serviced

Watchdog servicing is stopped. DSP shield shall be reset soon and message display on serial monitor shall be stopped

4.51 C:/DSPShieldDoc/Timers/Timers.cpp File Reference

Timers implementation.

```
#include "Timers.h"
```

Variables

- CSL_GptObj gptObj [TIMER_INSTANCE_COUNT]
- CSL_Handle hGpt
- CSL_WdtHandle hWdt
- CSL_WdtObj wdtObj
- TimerClass Timer

4.51.1 Detailed Description

Timers implementation.

4.51.2 Variable Documentation

4.51.2.1 CSL_GptObj gptObj[TIMER_INSTANCE_COUNT]

GPT object

4.51.2.2 CSL_Handle hGpt

GPT handle

4.51.2.3 CSL_WdtHandle hWdt

WDT handle

4.51.2.4 TimerClass Timer

Class identifier declaration

4.51.2.5 CSL_WdtObj wdtObj

WDT object

4.52 C:/DSPShieldDoc/Timers/Timers.h File Reference

Timers library header file.

```
#include "core.h"
#include "csl_gpt.h"
#include "csl_wdt.h"
```

Classes

- struct GPT_Config

  Configuration structure.
- struct WDT_Config

  Configuration structure.
- class TimerClass

  Timers Class.

Macros

- #define GPT0 (0)
- #define GPT1 (1)
- #define GPT2 (2)
- #define TIMER_INSTANCE_COUNT (3)
- #define GPT_PRE_SC_DIV_0 (0)
- #define GPT_PRE_SC_DIV_1 (1)
- #define GPT_PRE_SC_DIV_2 (2)
- #define GPT_PRE_SC_DIV_3 (3)
- #define GPT_PRE_SC_DIV_4 (4)
- #define GPT_PRE_SC_DIV_5 (5)
- #define GPT_PRE_SC_DIV_6 (6)
- #define GPT_PRE_SC_DIV_7 (7)
- #define GPT_PRE_SC_DIV_8 (8)
- #define GPT_PRE_SC_DIV_9 (9)
- #define GPT_PRE_SC_DIV_10 (10)
- #define GPT_PRE_SC_DIV_11 (11)
- #define GPT_PRE_SC_DIV_12 (12)
- #define HIGH_WORD_MASK (0xFFFF0000)
- #define LOW_WORD_MASK (0x0000FFFF)
- #define WORD_LENGTH (16)

Variables

- TimerClass Timer

4.52.1 Detailed Description

Timers library header file.

4.52.2 Macro Definition Documentation

4.52.2.1 #define GPT0 (0)

**********\ Timer global macro declarations \

Timer instace 0

4.52.2.2 #define GPT1 (1)

Timer instace 1

4.52.2.3 #define GPT2 (2)

Timer instace 2

4.52.2.4 #define GPT_PRE_SC_DIV_0 (0)

Pre scale Divide input clock by 2

4.52.2.5 #define GPT_PRE_SC_DIV_1 (1)

Pre scale Divide input clock by 4

4.52.2.6 #define GPT_PRE_SC_DIV_10 (10)

Pre scale Divide input clock by 2048

4.52.2.7 #define GPT_PRE_SC_DIV_11 (11)

Pre scale Divide input clock by 4096

4.52.2.8 #define GPT_PRE_SC_DIV_12 (12)

Pre scale Divide input clock by 8192

4.52.2.9 #define GPT_PRE_SC_DIV_2 (2)

Pre scale Divide input clock by 8

4.52.2.10 #define GPT_PRE_SC_DIV_3 (3)

Pre scale Divide input clock by 16

4.52.2.11 #define GPT_PRE_SC_DIV_4 (4)

Pre scale Divide input clock by 32

4.52.2.12 #define GPT_PRE_SC_DIV_5 (5)

Pre scale Divide input clock by 64

4.52.2.13 #define GPT_PRE_SC_DIV_6 (6)

Pre scale Divide input clock by 128

4.52.2.14 #define GPT_PRE_SC_DIV_7 (7)

Pre scale Divide input clock by 256

4.52.2.15 #define GPT_PRE_SC_DIV_8 (8)

Pre scale Divide input clock by 512

4.52.2.16 #define GPT_PRE_SC_DIV_9 (9)

Pre scale Divide input clock by 1024

4.52.2.17 #define HIGH_WORD_MASK (0xFFFF0000)

Mask for upper word of a 32bit value

4.52.2.18 #define LOW_WORD_MASK (0x0000FFFF)

Mask for lower word of a 32bit value

4.52.2.19 #define TIMER_INSTANCE_COUNT (3)

Timer instace count

4.52.2.20 #define WORD_LENGTH (16)

Length of a word

4.52.3 Variable Documentation

4.52.3.1 TimerClass Timer

Timers class instance extern which can be used by application programs to access Timers DSP APIs

Class identifier declaration

4.53 C:/DSPShieldDoc/USB/examples/interrupt/interrupt.cpp File Reference

Example demo to verify operation of USB module in Interrupt Mode.

```
#include "USB.h"
```

Functions

- interrupt void **usb_isr** (void)
- void suspendCallBack (int status)
- void **rxIntCallback** (void *hPipe, unsigned short pipeNum, unsigned short bytes)
- void rxCompleteCallback (void *hPipe, unsigned short pipeNum, unsigned short bytes)
- void **txIntCallback** (void *hPipe, unsigned short epNum, unsigned short bytes)
- void setup ()
- void **loop** ()

Variables

- USB_pipeHandle **hPipe** [USB_PIPE_COUNT]
- unsigned short **deviceDesc** [9]
- unsigned short **deviceQualDesc** [5] = {0x060A, 0x0200, 0x0000, 0x4000, 0x0002}
- unsigned short **cfgDesc** [40]
- unsigned short **OtherSpeedcfgDesc** [40]
- unsigned short **strDesc** [4][20]

4.53.1 Detailed Description

Example demo to verify operation of USB module in Interrupt Mode.

USB - Interrupt mode example

This example is to verify the operation of the USB module. This example runs in interrupt mode. USB interrupts are configured and ISR is registered using Interrupt module. After initializing and configuring the USB module, when there is any request from the USB host application USB ISR is triggered and the requested operation is performed inside the ISR.

This USB test can be verified by a host USB tool(c55xx_usb_ep_diag.exe which has to be obtained from TI) which can send or receive 64 bytes of data to the USB device. This tool should be installed on the host PC. This tool requires Jungo USB driver which can be downloaded from the link www.jungo.com. inf file should be installed for C5517 USB device using driver wizard of the windriver. This installation is required only when running this example for the first time.

Test procedure

1. Connect Arduino to host PC using USB cable.
2. Verify and Upload the example binary to DSP shield.
3. Open the Windriver driver wizard and install the inf file for the USB device (Only when running for first time)
4. Run the c55xx_usb_ep_diag.exe application. It displays following message

   DeviceAttach: received and accepted attach for vendor id 0x451, product id 0x901 0, interface 0, device handle 0x00392AD8

Main Menu (active Dev/Prod/Interface/Alt. Setting: 0x451/0x9010/0/0)

1. Display device configurations
2. Change interface alternate setting
3. Reset Pipe

4. Read/Write from pipes
5. Selective Suspend
6. Refresh
7. Exit Enter option:

For read and write operations command number should be selected depending on the end points configured for IN and OUT USB transfers. As per the current implementation, all end points are configured for both IN and OUT. Data should be read back from the same end point to which data was written. For example, write data to EP1 and read back the data from EP1 itself.

1. The first command should be read followed by write-read... for the proper synchronization of host and target USB device.
2. For verifying the USB operations send write command from host and then read command. check whether the data sent by the host is transmitted back by the target or not.
3. During read/write operations data will be displayed by the host USB tool

4.53.2 Function Documentation

4.53.2.1 void rxCompleteCallback (void * *hPipe,* unsigned short *pipeNum,* unsigned short *bytes*)

Do nothing for Ep0

4.53.2.2 void setup (void)

Connect the USB device

4.53.2.3 void suspendCallBack (int *status*)

For future use

4.53.3 Variable Documentation

4.53.3.1 unsigned short cfgDesc[40]

Initial value:

```
= {0x0209, 0x004A, 0x0101, 0xC001, 0x0928,
                                0x0004, 0x0800, 0x0000, 0x0000,
                                0x0507, 0x0281, 0x0200, 0x0700,
                                0x0105, 0x0002, 0x0002,
                                0x0507, 0x0282, 0x0200, 0x0700,
                                0x0205, 0x0002, 0x0002,
                                0x0507, 0x0283, 0x0200, 0x0700,
                                0x0305, 0x0002, 0x0002,
                                0x0507, 0x0284, 0x0200, 0x0700,
                                0x0405, 0x0002, 0x0002
                               }
```

4.53.3.2 unsigned short deviceDesc[9]

Initial value:

```
= {0x0112, 0x0200, 0x0000, 0x4000, 0x0451,
                                0x9010, 0x0100, 0x0201, 0x0103}
```

4.53.3.3 unsigned short OtherSpeedcfgDesc[40]

Initial value:

```
= {0x0709, 0x004A, 0x0201, 0xC001, 0x0928,
                                        0x0004, 0x0800, 0x0000, 0x0000,
                                        0x0507, 0x0281, 0x0040, 0x0700,
                                        0x0105, 0x4002, 0x0000,
                                        0x0507, 0x0282, 0x0040, 0x0700,
                                        0x0205, 0x4002, 0x0000,
                                        0x0507, 0x0283, 0x0040, 0x0700,
                                        0x0305, 0x4002, 0x0000,
                                        0x0507, 0x0284, 0x0040, 0x0700,
                                        0x0405, 0x4002, 0x0000
                                       }
```

4.53.3.4 unsigned short strDesc[4][20]

Initial value:

```
= {

                                    {0x0304, 0x0409},

                                    {0x0324, 0x0045, 0x0054, 0x0041, 0x0058, 0x0020,
                                     0x0053, 0x004E, 0x0049, 0x0054, 0x0053, 0x0055,
                                     0x0052, 0x0045, 0x004D, 0x0054, 0x004E, 0x0053},

                                    {0x030C, 0x0043, 0x0035, 0x0035, 0x0031, 0x0037},

                                    {0x030C, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031}
                                   }
```

4.54 C:/DSPShieldDoc/USB/USB.cpp File Reference

USB implementation.

```
#include "USB.h"
```

Functions

- int startTransferCallback (void *vpContext, void *vpeps)

  Start transfer call back function.
- int completeTransferCallback (void *vpContext, void *vpeps)

  Complete transfer call back function.

Variables

- CSL_UsbHostPktDescr hpdtx
- CSL_UsbHostPktDescr hpdrx
- unsigned long linking_ram0 [USB_LRAM_SIZE]
- unsigned short usbDataBufferTxRx1 [USB_DATA_SIZE]
- unsigned short usbDataBufferTxRx2 [USB_DATA_SIZE]
- unsigned short usbDataBufferTxRx3 [USB_DATA_SIZE]
- unsigned short usbDataBufferTxRx4 [USB_DATA_SIZE]
- USBClass USB
- CSL_UsbDevHandle hUsbDev = NULL
- USB_Config gUsbConfig
- bool sentLongEp0Pkt

4.54.1 Detailed Description

USB implementation.

4.54.2 Function Documentation

4.54.2.1 int completeTransferCallback (void * *vpContext*, void * *vpeps*)

Complete transfer call back function.

Parameters

vpContext	- USB context structure
vpeps	- End point status structure pointer

Returns

CSL_Status

4.54.2.2 int startTransferCallback (void * *vpContext*, void * *vpeps*)

Start transfer call back function.

Parameters

vpContext	- USB context structure
vpeps	- End point status structure pointer

Returns

CSL_Status

The endpoint should be initialized

4.54.3 Variable Documentation

4.54.3.1 USB_Config gUsbConfig

USB configuration structure

4.54.3.2 CSL_UsbHostPktDescr hpdrx

USB host packet descriptor for reading

4.54.3.3 CSL_UsbHostPktDescr hpdtx

USB host packet descriptor for writing

4.54.3.4 CSL_UsbDevHandle hUsbDev = NULL

USB handle

4.54.3.5 unsigned long linking_ram0[USB_LRAM_SIZE]

Linking RAM buffer

4.54.3.6 bool sentLongEp0Pkt

Flag to indicate if a long data packet is being sent

4.54.3.7 USBClass USB

USB class object

4.54.3.8 unsigned short usbDataBufferTxRx1[USB_DATA_SIZE]

USB data buffer 1

4.54.3.9 unsigned short usbDataBufferTxRx2[USB_DATA_SIZE]

USB data buffer 2

4.54.3.10 unsigned short usbDataBufferTxRx3[USB_DATA_SIZE]

USB data buffer 3

4.54.3.11 unsigned short usbDataBufferTxRx4[USB_DATA_SIZE]

USB data buffer 4

4.55 C:/DSPShieldDoc/USB/USB.h File Reference

USB library header file.

```
#include <stdio.h>
#include <string.h>
#include "tistdtypes.h"
#include "csl_usb.h"
#include "csl_usbAux.h"
#include "csl_pll.h"
#include "core.h"
```

Classes

- class USB_Config

  USB confuguration Class.
- class Pipe_Config

  Pipe confuguration Class.
- class USBClass

  USB Class.

Macros

- #define USB_HS_MAX_PACKET_SIZE (64)
- #define USB_DATA_SIZE (USB_HS_MAX_PACKET_SIZE / 2)
- #define USB_LRAM_SIZE (256)
- #define USB_MAX_CURRENT (50)
- #define USB_WAKEUP_DELAY (10)
- #define USB_OUT (1)
- #define USB_IN (2)
- #define USB_PIPE_COUNT (CSL_USB_ENDPOINT_COUNT + 2)
- #define USB_STRDESC_COUNT (4)
- #define USB_EP0 (CSL_USB_EP0)
- #define USB_EP1 (CSL_USB_EP1)
- #define USB_EP2 (CSL_USB_EP2)
- #define USB_EP3 (CSL_USB_EP3)
- #define USB_EP4 (CSL_USB_EP4)
- #define USB_OUT_EP0 (CSL_USB_OUT_EP0)
- #define USB_OUT_EP1 (CSL_USB_OUT_EP1)
- #define USB_OUT_EP2 (CSL_USB_OUT_EP2)
- #define USB_OUT_EP3 (CSL_USB_OUT_EP3)
- #define USB_OUT_EP4 (CSL_USB_OUT_EP4)
- #define USB_IN_EP0 (CSL_USB_IN_EP0)
- #define USB_IN_EP1 (CSL_USB_IN_EP1)
- #define USB_IN_EP2 (CSL_USB_IN_EP2)
- #define USB_IN_EP3 (CSL_USB_IN_EP3)
- #define USB_IN_EP4 (CSL_USB_IN_EP4)

Typedefs

- typedef void(* USB_APP_CALLBACK)(Uint16 flag)
- typedef int(* USB_APP_INT_CALLBACK)(void *, unsigned short, unsigned short)
- typedef void * USB_pipeHandle
- typedef enum USB_opMode USB_opMode

  Enum to indicate operation mode.
- typedef enum USB_xferType USB_xferType

  Enum to indicate the transfer type.

Enumerations

- enum USB_opMode { **USB_OPMODE_POLLED** = 0, **USB_OPMODE_DMA** }

  Enum to indicate operation mode.
- enum USB_xferType { **USB_CTRL** = 0, **USB_BULK** }

  Enum to indicate the transfer type.

Variables

- USBClass USB

4.55.1 Detailed Description

USB library header file.

4.55.2 Macro Definition Documentation

4.55.2.1 #define USB_DATA_SIZE (USB_HS_MAX_PACKET_SIZE / 2)

Maximum USB data size

4.55.2.2 #define USB_EP0 (CSL_USB_EP0)

Macro to indicate Endpoint 0

4.55.2.3 #define USB_EP1 (CSL_USB_EP1)

Macro to indicate Endpoint 1

4.55.2.4 #define USB_EP2 (CSL_USB_EP2)

Macro to indicate Endpoint 2

4.55.2.5 #define USB_EP3 (CSL_USB_EP3)

Macro to indicate Endpoint 3

4.55.2.6 #define USB_EP4 (CSL_USB_EP4)

Macro to indicate Endpoint 4

4.55.2.7 #define USB_HS_MAX_PACKET_SIZE (64)

USB highspeed maximum packet size

4.55.2.8 #define USB_IN (2)

USB IN endpoint number

4.55.2.9 #define USB_IN_EP0 (CSL_USB_IN_EP0)

Macro to indicate IN Endpoint 0

4.55.2.10 #define USB_IN_EP1 (CSL_USB_IN_EP1)

Macro to indicate IN Endpoint 1

4.55.2.11 #define USB_IN_EP2 (CSL_USB_IN_EP2)

Macro to indicate IN Endpoint 2

4.55.2.12 #define USB_IN_EP3 (CSL_USB_IN_EP3)

Macro to indicate IN Endpoint 3

4.55.2.13 #define USB_IN_EP4 (CSL_USB_IN_EP4)

Macro to indicate IN Endpoint 4

4.55.2.14 #define USB_LRAM_SIZE (256)

Linking RAM size

4.55.2.15 #define USB_MAX_CURRENT (50)

USB maximum current

4.55.2.16 #define USB_OUT (1)

USB OUT endpoint number

4.55.2.17 #define USB_OUT_EP0 (CSL_USB_OUT_EP0)

Macro to indicate OUT Endpoint 0

4.55.2.18 #define USB_OUT_EP1 (CSL_USB_OUT_EP1)

Macro to indicate OUT Endpoint 1

4.55.2.19 #define USB_OUT_EP2 (CSL_USB_OUT_EP2)

Macro to indicate OUT Endpoint 2

4.55.2.20 #define USB_OUT_EP3 (CSL_USB_OUT_EP3)

Macro to indicate OUT Endpoint 3

4.55.2.21 #define USB_OUT_EP4 (CSL_USB_OUT_EP4)

Macro to indicate OUT Endpoint 4

4.55.2.22 #define USB_PIPE_COUNT (CSL_USB_ENDPOINT_COUNT + 2)

Number of pipes supported

4.55.2.23 #define USB_STRDESC_COUNT (4)

String descriptor count

4.55.2.24 #define USB_WAKEUP_DELAY (10)

USB wakeup delay

4.55.3 Typedef Documentation

4.55.3.1 typedef void(* USB_APP_CALLBACK)(Uint16 flag)

USB application call back pointer

4.55.3.2 typedef int(* USB_APP_INT_CALLBACK)(void *, unsigned short, unsigned short)

USB application call back pointer for tx and rx interrupts

4.55.3.3 typedef void* USB_pipeHandle

USB pipe handle

4.55.4 Variable Documentation

4.55.4.1 USBClass USB

USB class instance extern which can used by application programs to access USB DSP APIs

USB class object

4.56 C:/DSPShieldDoc/Wire/examples/Wire/Wire.cpp File Reference

Toggles LEDs connected to IO expander in sequence Wire Demo.

Functions

- void setup ()
- void loop ()

  Toggle which of the 3 LEDs will be on.

Variables

- unsigned int **cmd** [2]
- int **ledValue** = 0
- int **ledState** = 0

4.56.1 Detailed Description

Toggles LEDs connected to IO expander in sequence Wire Demo.

4.56.2 Function Documentation

4.56.2.1 void loop (void)

Toggle which of the 3 LEDs will be on.

Below logic ensures 3 LEDs connected to IO expander are toggled in sequence LED0 (pin 0) - Will be ON when 'ledValue' is '0' LED1 (pin 1) - Will be ON when 'ledValue' is '1' LED2 (pin 2) - Will be ON when 'ledValue' is '2'

Delay of 1sec

Value 'ledValue' never goes beyond 3

4.56.2.2 void setup (void)

Command for IO expander configuration

0x21 is the address of IO expander for which LED are connected

Configures IO expander port 0 pins as output

Delay of 1sec

Index