C55x Digital Signal Processors Software Overview

Agenda

- C55x Chip Support Library (CSL)
  - Introduction
  - Benefits
  - Structure
  - Example
- C55x DSP Library (DSPLIB)
  - Introduction
  - Structure
  - Programmer Reference Guide
  - Benchmarks

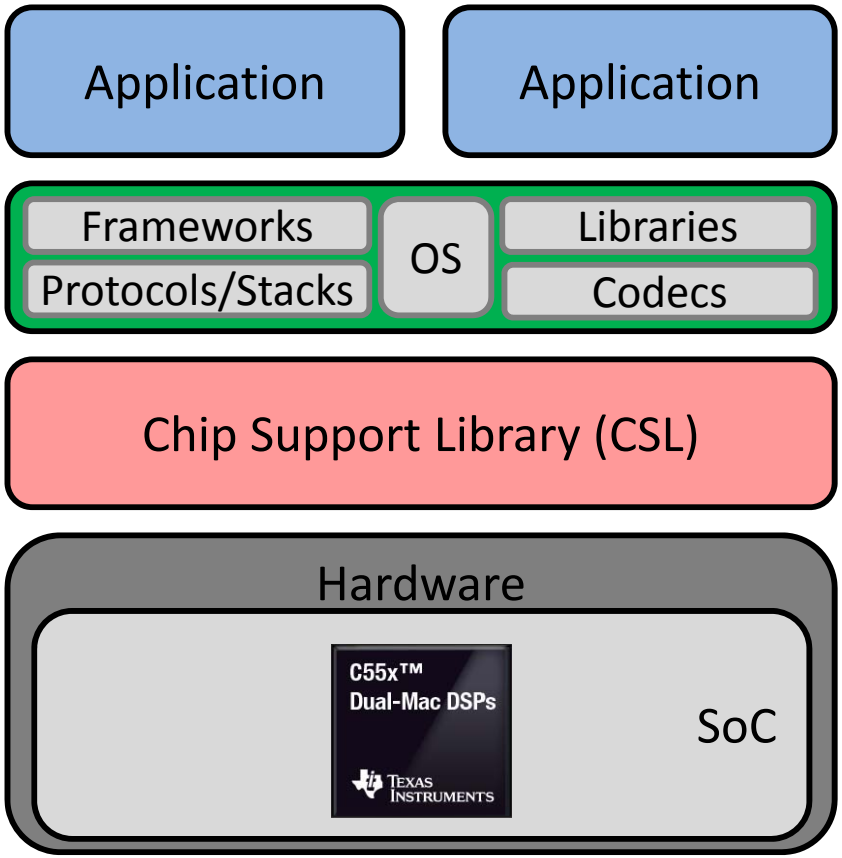
Chip Support Library (CSL)

C55x DSP Software Overview

Chip Support Library (CSL): Introduction

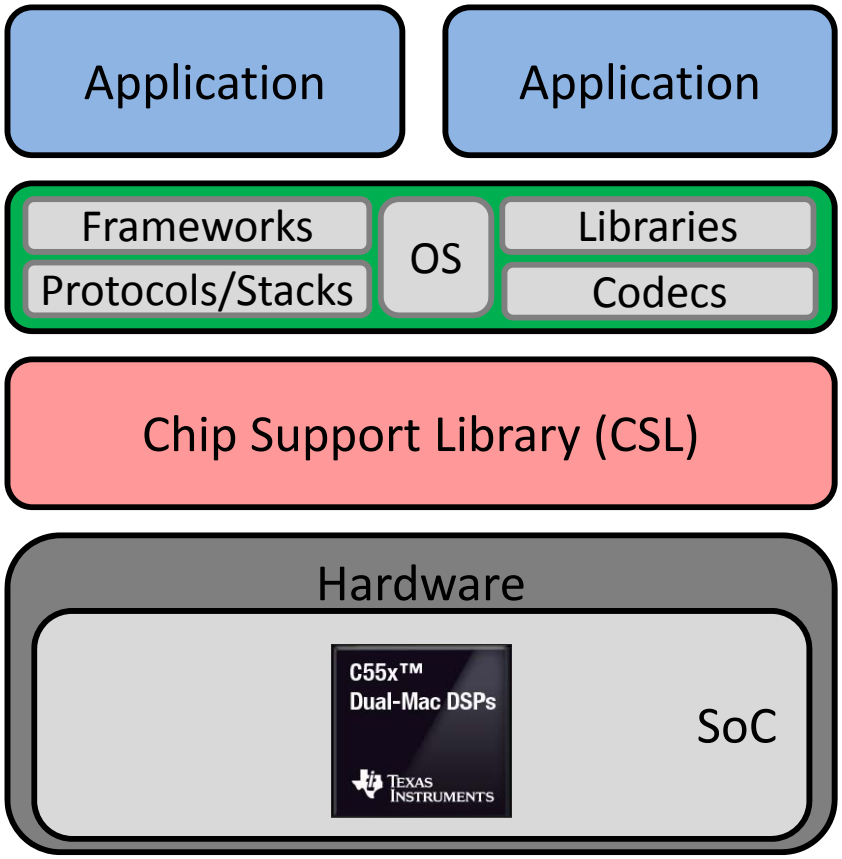
Chip Support Library (CSL):

- Facilitates software development on the following devices: C5504, C5505, C5514, C5515, C5517, C5535, and C5545
- Provides a collection of functions, macros, and symbols used to configure and control on-chip peripherals
- Is fully scalable and does not require the use of DSP/BIOS components to operate
- Contains examples that exercise the various peripherals on the C55x DSP and provides a foundation to build applications



CSL Benefits

- Benefits of CSL:
  - Peripherals ease of use
  - Shortened development time
  - Portability
  - Hardware abstraction
  - Level of standardization and compatibility among devices
- Reference for customer driver development:
  - All source code for CSL is open to customers.
  - Most of CSL is written in C.
- CCS Compatible: Examples are provided for CCSv6



CSL Structure: Package Contents

- Release notes and guides.
- Peripheral drivers with source code:

DAT	GPIO	I2S	McBSP	PLL	SAR	UHPI
DMA	GPT	INTC	McSPI	PM	SPI	USB
EMIF	I2C	LCD	MMC/SD	RTC	UART	WDT

- C55x CSL pre-built library and header files
- API documentation
- CCSv6 example code for the peripherals
- Programming Utility to burn user-specified binaries to memory:

NAND Flash	SPI EEPROM	MMC	SPI Flash
NOR Flash	IIC EEPROM	SD	McSPI Flash

CSL Directory Structure: Root

These are the top-level directories in the C55XCSL package:

TMS320C55XXCSL-LOWPWR-XX.XX.XX.XX

build

ccs_v6.x_examples

doc

drivers

inc

src

webgen

Installation_Guide.pdf

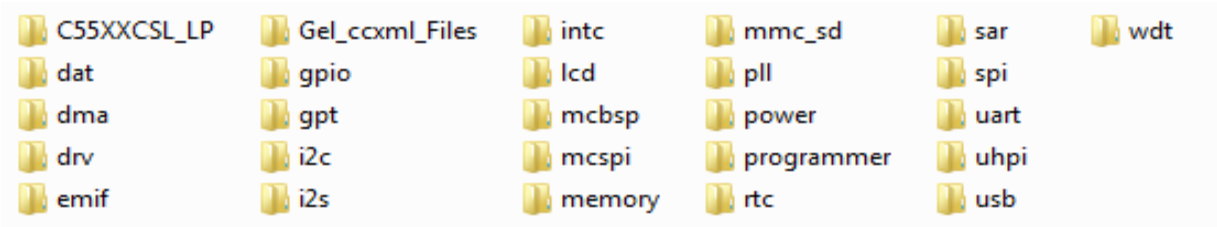
Release_Notes.txt

SWManifest.pdf

The Installation Guide provides details on how to load/build/execute example code in CCS.

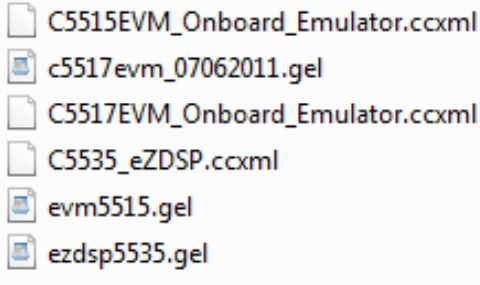
CSL Directory Structure: CCS, Build

ccs_v6.x_examples



These examples can be imported into the CCS workspace, built, and executed on the C55x EVMs. The example folder also contains the programmer tool.

build



The build directory contains device-specific configuration files for CCS projects.

CSL Directory Structure: Inc, Src

inc

The inc directory contains the CSL include files.

ata.h	csl_gpio.h	csl_mcbsp_tgt.h	csl_rtc.h	csl_uhpiAux.h	cslr_i2s.h	cslr_sysctrl.h	mcspi.h
csl_audioClass.h	csl_gpt.h	csl_mcspi.h	csl_sar.h	csl_usb.h	cslr_idle.h	cslr_tim.h	soc.h
csl_audioClassAux.h	csl_hid.h	csl_mem.h	csl_sdio.h	csl_usbAux.h	cslr_lcdc.h	cslr_uart.h	tistdtypes.h
csl_cdc.h	csl_hidAux.h	csl_mmcsd.h	csl_spi.h	csl_wdt.h	cslr_ldo.h	cslr_uhpi.h	
csl_cdcAux.h	csl_i2c.h	csl_mmcsd_ataIf.h	csl_sysctrl.h	cslr.h	cslr_mcbsp.h	cslr_usb.h	
csl_dat.h	csl_i2s.h	csl_msc.h	csl_types.h	cslr_cpu.h	cslr_mcspi.h	cslr_wdt.h	
csl_dma.h	csl_intc.h	csl_mscAux.h	csl_uart.h	cslr_dma.h	cslr_mmcsd.h	evm5515.h	
csl_emif.h	csl_lcdc.h	csl_nand.h	csl_uartAux.h	cslr_emif.h	cslr_rtc.h	evm5515_emif.h	
csl_error.h	csl_lcdcAux.h	csl_pll.h	csl_uhpi.h	cslr_gpio.h	cslr_sar.h	evm5515_gpio.h	
csl_general.h	csl_mcbsp.h	csl_pllAux.h	csl_uhpiAccess.h	cslr_i2c.h	cslr_spi.h	evm5515_i2c.h	

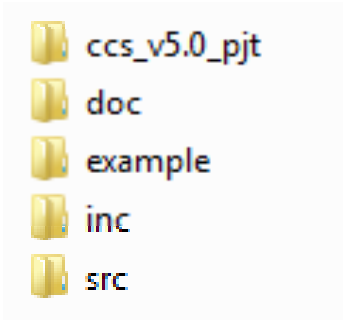
src

The src directory contains the CSL source code.

csl_audioClass.c	csl_i2c.c	csl_mem.c	csl_sdio.c	evm5515_gpio.c
csl_cdc.c	csl_i2c_ioExpander.c	csl_mmcsd.c	csl_spi.c	evm5515_i2c.c
csl_dat.c	csl_i2s.c	csl_mmcsd_ataIf.c	csl_sysctrl.c	vectors.asm
csl_dma.c	csl_intc.c	csl_msc.c	csl_uart.c	
csl_emif.c	csl_irqplug.asm	csl_nand.c	csl_uhpi.c	
csl_gpio.c	csl_lcdc.c	csl_pll.c	csl_usb.c	
csl_gpt.c	csl_mcbsp.c	csl_rtc.c	csl_wdt.c	
csl_hid.c	csl_mcspi.c	csl_sar.c	evm5515.c	

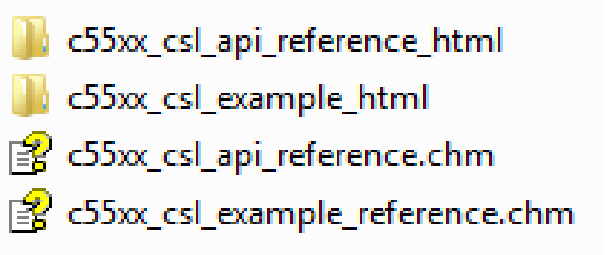
CSL Directory Structure: Drivers, Doc

drivers



The drivers directory contains an SD card driver and example.

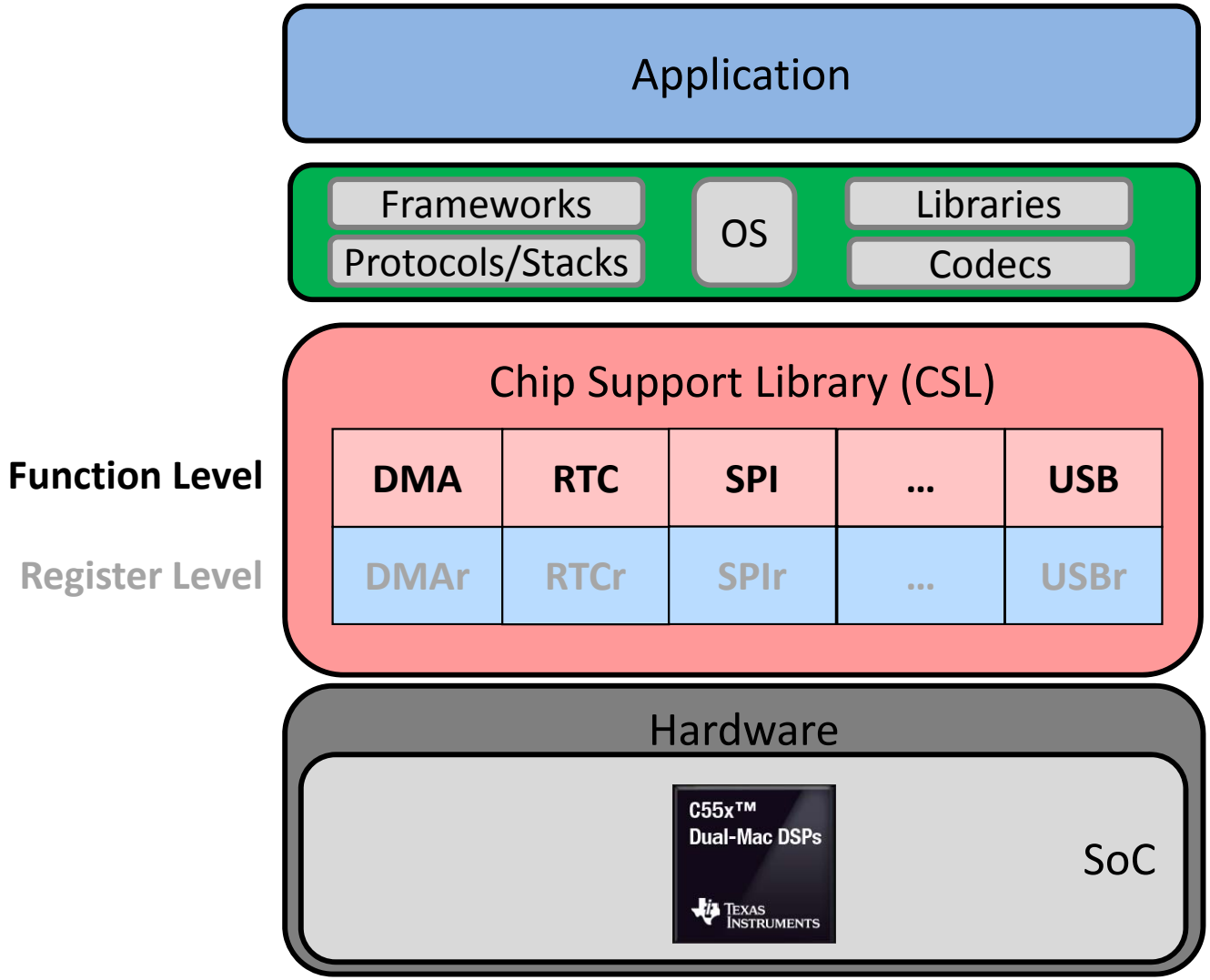
doc



The doc directory contains HTML-based documentation describing the various API calls and their arguments.

CSL Modules

- CSL consists of modules that are built and archived into a library file.
- Each peripheral is covered by a single module.
- Applications can be built utilizing either the function level or register level.



CSL Naming Conventions

The following conventions are used when naming CSL functions, macros, and data types.

Object	Naming Convention
Function	PER_funcName()
Variable	PER_varName
Macro	PER_MACRO_NAME
Typedef	PER_Typename
Function Argument	funcArg
Structure Member	memberName

- All functions, macros, and data types start with PER_ (where PER is the peripheral module) in uppercase letters. At times CSL precedes the PER syntax.
- Function names use all lowercase letters. Uppercase letters are used only if the function name consists of two separate words. For example, PER_getConfig().
- Macro names use all uppercase letters; for example, DMA_INTERRUPT_DISABLE.

CSL Example Code

- In order to better understand the usage of the CSL. Lets unpack the GPIO_ouput example contained in the package.
- **Step 1:** Include the header files of the module/peripheral, use <csl_gpio.h>.

```
//required include files
#include "csl_gpio.h"
#include "csl_intc.h"
#include <stdio.h>
#include <csl_general.h>
#include "csl_sysctrl.h"

//parameters assoc with the test
#define CSL_TEST_FAILED  (-1)
#define CSL_TEST_PASSED  (0)

//structures to be used in the example
CSL_GpioObj    gpioObj;
CSL_GpioObj    *hGpio;
```

- **Step 2:** Define the GPIO function prototypes.

```
int  gpio_output_pin_test(void);
int gpio_pin_config_test(void);
```

CSL Example Code

- **Step 3:** Start defining the test in main(). The returned result from gpio_output_pin_test determines the test outcome.

```
void main(void)
{
    printf("GPIO Output Pin Test!\n");

    result = gpio_output_pin_test();
    if(CSL_TEST_PASSED == result)
    {
        printf("GPIO Output Pin Test Passed!!\n");
    }
    else
    {
        printf("GPIO Output Pin Test Failed!!\n");

        PaSs_StAtE = 0x0000;
    }

}
```

Gpio_output_pin_test()
Unpacked in the next slide

Check the returned value from gpio_output_pin_test() to determine the test result.

CSL Example Code

gpio_output_pin_test() unpacked

```
int  gpio_output_pin_test(void)
{
    status = SYS_setEBSR(CSL_EBSR_FIELD_SP0MODE,
                         CSL_EBSR_SP0MODE_2);
    if(CSL_SOK != status)
    {
        printf("SYS_setEBSR failed\n");
        return(CSL_TEST_FAILED);
    }

    hGpio = GPIO_open(&gpioObj, &status);
    if((NULL == hGpio) || (CSL_SOK != status))
    {
        printf("GPIO_open failed\n");
        return(CSL_TEST_FAILED);
    }
    else
    {
        printf("GPIO_open Successful\n");
    }

    GPIO_reset(hGpio);

    config.pinNum     = CSL_GPIO_PIN0;
    config.direction  = CSL_GPIO_DIR_OUTPUT;
    config.trigger    = CSL_GPIO_TRIG_CLEAR_EDGE;

    status = GPIO_configBit(hGpio, &config);
    if(CSL_SOK != status)
    {
        printf("GPIO_configBit failed\n");
        return(CSL_TEST_FAILED);
    }
```

Pin Muxing for GPIO Pins. Make Port 0 serial and enable an 8-pin parallel port. Set pins A15 to A20 as GPIO.

Open and reset the GPIO module.

Configure GPIO Pin 0 as an output pin.

CSL Example Code

gpio_output_pin_test() unpacked.

```
status = GPIO_write(hGpio, CSL_GPIO_PIN0, writeVal);
if(CSL_SOK != status)
{
    printf("GPIO_write Failed\n");
    return(CSL_TEST_FAILED);
}
else
{
    printf("GPIO_write Successful\n");
}
```

Write a 1 to the output Pin 0.

```
status = GPIO_read(hGpio, CSL_GPIO_PIN0, &readVal);
if(CSL_SOK != status)
{
    printf("GPIO_read failed\n");
    return(CSL_TEST_FAILED);
}
else
{
    printf("GPIO_read Successful\n");
}

if(writeVal == readVal)
{
    printf("Data read is same as data written\n");
}
else
{
    printf("Data read is not same as data written \n");
    return(CSL_TEST_FAILED);
}
```

Read Pin 0 and compare the read and written data to ensure that they are the same.

```
status = GPIO_close(hGpio);
if(CSL_SOK != status)
{
    printf("GPIO_close failed\n");
    return(CSL_TEST_FAILED);
}
return(CSL_TEST_PASSED);
```

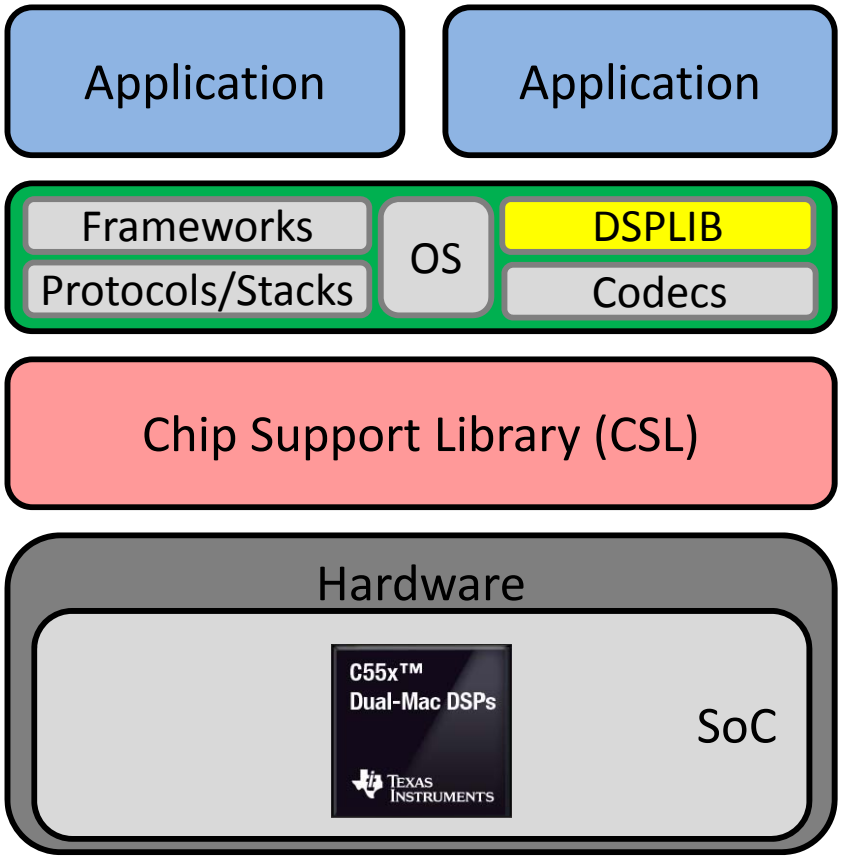
Close the GPIO module.

DSP Library (DSPLIB)

C55x DSP Software Overview

C55x DSPLIB: Introduction

- The TMS320C55x DSP Library (DSPLIB) from Texas Instruments is an optimized DSP Functions Library designed for use by C programmers on TMS320C55x devices.
- DSPLIB includes over 50 C-callable, assembly-optimized, general-purpose signal processing functions and mathematical functions.
  - Signal processing functions are typically used in computationally intensive real-time applications.
  - C55x is a fixed point processor. Most of the DSPLIB functions process 16-bit or 32-bit, fixed-point data. A few of the functions process floating-point data (Conversion from FP to Q15).
- Benchmark code and results are provided in the package.
- The package is tested against Matlab scripts.



C55x DSPLIB: DSP Routines

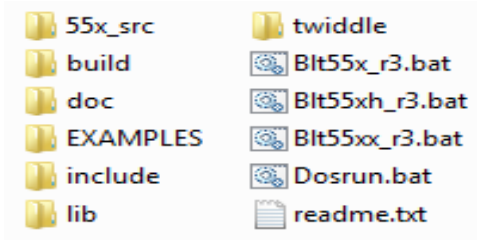
Below are some commonly used DSP routines contained within DSPLIB. These routines can be modified as needed to match the end application:

- Signal Processing: FFT, Filtering, FIR and IIR (complex and real), Convolution and Correlation, Adaptive Filters
- Standard Math: Log, Vector Add, subtract, Minimum, Maximum Reciprocal
- Trigonometry Functions: Sine, Arc tangent
- Linear Algebra: Matrix manipulation and transpose
- Utilities: Random number generation, Q15 to floating-point conversion (and back)

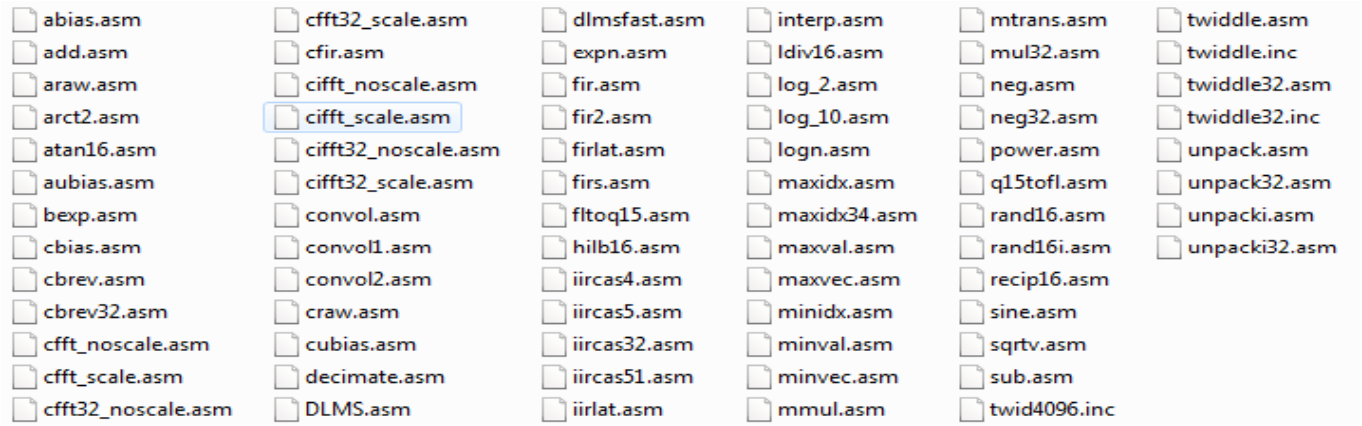
DSPLIB Directory Structure: Root, Src

These are the top-level directories in the DSPLIB package:

c55_dsplib_*XX.XX.XX.XX*



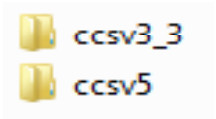
55x_src



The 55x_src directory has the source code for all of the library functions. This source is written in assembly and optimized for the C55X architecture.

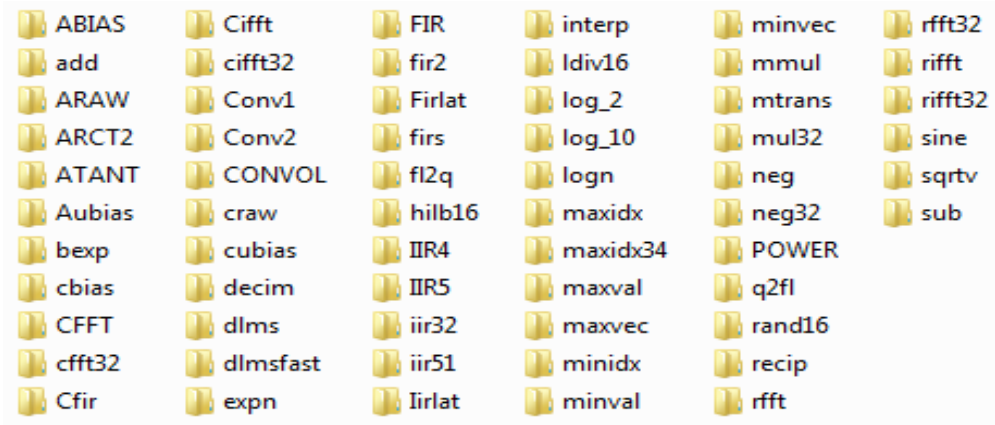
DSPLIB Directory Structure: Build, Examples

build



The build directory has the CCSv3 and CCSv5 main DSPLIB project that pulls in the files from the c55x_src. CCSv6 will be added in the future.

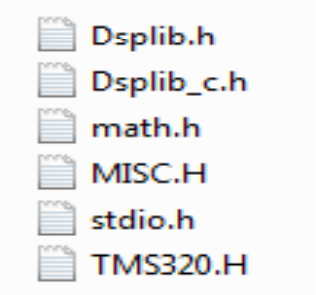
EXAMPLES



The examples directory has unit tests for ALL functions in the library. The examples provide a great platform to build applications utilizing DSPLIB.

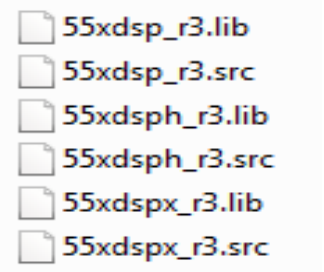
DSPLIB Directory Structure: Include, Lib

include



The include directory has include files that are needed to call DSPLIB functions.

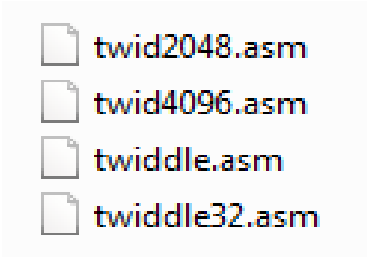
lib



The lib directory contains pre-built libraries as well as a collection of all source code.

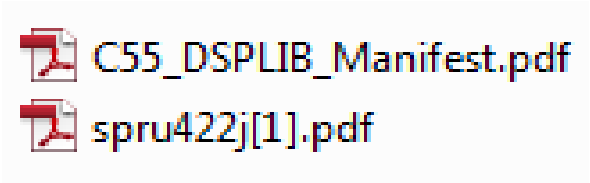
DSPLIB Directory Structure: Twiddle, Doc

twiddle



The twiddle directory contains routines with pre-calculated values used by FFT functions.

doc



The doc directory has the programmer reference document and the software manifest.

DSPLIB Programmer's Reference

The TMS320C55x DSP Library Programmer's Reference contains descriptions and detailed usage for each of the DSPLIB functions. Below is an example on the cfft function:

cfft *Forward Complex FFT*

Function

void cfft (DATA *x, ushort nx, type);
(defined in cfft.asm)

Arguments

x [2*nx]	Pointer to input vector containing nx complex elements (2*nx real elements) in normal order. On output, vector contains the nx complex elements of the FFT(x) in bit-reversed order. Complex numbers are stored in interleaved Re-Im format.
nx	Number of complex elements in vector x. Must be between 8 and 1024.
type	FFT type selector. Types supported: ❑ If type = SCALE, scaled version selected ❑ If type = NOSCALE, non-scaled version selected

Description

Computes a complex nx-point FFT on vector x, which is in normal order. The original content of vector x is destroyed in the process. The nx complex elements of the result are stored in vector x in bit-reversed order. The twiddle table is in bit-reversed order.

Benchmarks and Performance Estimation

The TMS320C55x DSP Library Programmer's Reference provides performance numbers (in cycles) for each FFT function. Below is an example of the benchmarks for the Forward Complex FFT (cfft) function:

FFT Size	Cycles†	Code Size (in bytes)
8	208	493
16	358	493
32	624	493
64	1210	493
128	2516	493
256	5422	493
512	11848	493
1024	25954	493

† Assumes all data is in on-chip dual-access RAM and that there is no bus conflict due to twiddle table reads and instruction fetches (provided linker command file reflects those conditions).

An application note describing the process of implementing C55x benchmarks can be found here: *(placeholder for Ran's DSPLIB benchmark app note)*.

For More Information

- C55x Product Folder: http://www.ti.com/c55x
- C55x CSL: http://www.ti.com/tool/sprc133
- C55x DSP Library (DSPLIB): http://www.ti.com/tool/sprc100
- C55x DSP Library (DSPLIB) Programmer's Reference: http://www.ti.com/lit/spru422j
- DSP/BIOS Real-Time Operating System (RTOS): http://www.ti.com/tool/dspbios
- C55x Tools w/ programmer, boot image creator, board support package, gel file, schematics, BOM, etc:
  - C5515 EVM Support Page: http://support.spectrumdigital.com/boards/evm5515
  - C5517 EVM Support Page: http://support.spectrumdigital.com/boards/evm5517
  - C5535 eZdsp Support Page: http://support.spectrumdigital.com/boards/ezdsp5535
- Code Composer Studio (CCS) http://www.ti.com/tool/ccstudio
- Code Composer Studio Training http://processors.wiki.ti.com/index.php/Category:CCS_Training
- For questions regarding topics covered in this training, visit the support forums at the TI E2E Community website: http://e2e.ti.com