**Trademarks**
TMS320 is the trademark of Texas Instruments Incorporated.
All other trademark mentioned herein are property of their respective companies

**Acronyms**
xDAIS : eXpress DSP Algorithm Interface Standard
IALG  : Algorithm interface defines a framework independent interface for the creation of
       algorithm instance objects
STB   : Software Test Bench
QMATH: Fixed Point Mathematical computation
CcA   : C-Callable Assembly
FIR   : Finite Impulse Response Filter
IIR   : Infinite Impulse Response Filter
FFT   : Fast Fourier Transform
# Contents

1. Software Test Bench (STB) overview ................................................................. 1

2. Math module directory structure................................................................. 5

3. C-framework configuration and warning messages................................. 7

4. ASM framework configuration................................................................. 9

5. Hardware setup.......................................................................................... 11
   5.1. F2407 Evaluation module .................................................................. 11
   5.2. F243 Evaluation module .................................................................. 11

6. Loading and building the CC project for C/ASM framework.................. 12

7. Software Test Bench (STB).......................................................................... 14
   7.1. Software Test Bench (STB) for QSIN/QSINLT ................................ 14
   7.2. Software Test Bench (STB) for QCOS/QCOSLT ............................. 16
   7.3. Software Test Bench (STB) for QATAN ........................................... 18
   7.4. Software Test Bench (STB) for QSQRT .......................................... 20
   7.5. Software Test Bench (STB) for QLOG10 .......................................... 22
   7.6. Software Test Bench (STB) for QLOGN ......................................... 24
   7.7. Software Test Bench (STB) for QINV1 ............................................. 26
   7.8. Software Test Bench (STB) for QINV2 ............................................. 28
1. Software Test Bench (STB) overview

This document describes the Software Test Bench (STB) to demonstrate the fixed-point math functions implemented using C2000 ISA. The STB is available in ASM/C framework to demonstrate “ASM Only” and “CcA” math modules respectively and it is designed to run on x243, x2407 EVM and x2407eZdsp kit.

The idea behind the STB strategy to demonstrate the math function is indeed simple. Input to the math function is allowed to ramp up at incremental steps to sweep over a range of interest in a cyclic manner and the input/output of the math function are logged, sent through EVMDAC and PWMDAC for observation by the user.

PWMDAC output is available only in the case of TMS320X240x DSP, which has two Event managers. This is due to the fact that the PWMDAC uses Timer T3 available in Event manager B(EVB) to generate 20Khz PWM outputs.

The STB for math functions in C/ASM framework has the following properties

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Program memory usage</th>
<th>Data memory usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q SIN</td>
<td>727 words</td>
<td>100 words</td>
</tr>
<tr>
<td>Q SIN LT</td>
<td>968 words</td>
<td>101 words</td>
</tr>
<tr>
<td>Q COS</td>
<td>741 words</td>
<td>101 words</td>
</tr>
<tr>
<td>Q COS LT</td>
<td>972 words</td>
<td>101 words</td>
</tr>
<tr>
<td>Q ATAN</td>
<td>776 words</td>
<td>103 words</td>
</tr>
<tr>
<td>Q SQRT</td>
<td>807 words</td>
<td>103 words</td>
</tr>
<tr>
<td>Q LOG10</td>
<td>758 words</td>
<td>103 words</td>
</tr>
<tr>
<td>Q LOGN</td>
<td>758 words</td>
<td>103 words</td>
</tr>
<tr>
<td>QINV1</td>
<td>730 words</td>
<td>103 words</td>
</tr>
<tr>
<td>QINV2</td>
<td>722 words</td>
<td>100 words</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Program memory usage</th>
<th>Data memory usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q SIN</td>
<td>1118 words</td>
<td>103 words</td>
</tr>
<tr>
<td>Q SIN LT</td>
<td>1362 words</td>
<td>103 words</td>
</tr>
<tr>
<td>Q COS</td>
<td>1133 words</td>
<td>103 words</td>
</tr>
<tr>
<td>Q COS LT</td>
<td>1366 words</td>
<td>103 words</td>
</tr>
<tr>
<td>Q ATAN</td>
<td>1162 words</td>
<td>104 words</td>
</tr>
<tr>
<td>Q SQRT</td>
<td>1211 words</td>
<td>104 words</td>
</tr>
<tr>
<td>Q LOG10</td>
<td>1160 words</td>
<td>104 words</td>
</tr>
<tr>
<td>Q LOGN</td>
<td>1160 words</td>
<td>104 words</td>
</tr>
<tr>
<td>QINV1</td>
<td>1121 words</td>
<td>104 words</td>
</tr>
<tr>
<td>QINV2</td>
<td>1113 words</td>
<td>103 words</td>
</tr>
</tbody>
</table>

1 Excluding the Stack Size
## Software Test Bench (STB) overview

<table>
<thead>
<tr>
<th>Development/Emulation</th>
<th>Code Composer 4.2 (or above) with Real Time debugging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target Controller</td>
<td>Spectrum Digital – x243 EVM, x2407 EVM &amp; x2407eZdsp kit.</td>
</tr>
<tr>
<td>Emulator</td>
<td>XDS510PP Plus</td>
</tr>
<tr>
<td>Board Name (in CC Setup)</td>
<td>sdgo2xx (Spectrum Digital)</td>
</tr>
<tr>
<td>Desktop Area</td>
<td>1024×768 pixels</td>
</tr>
<tr>
<td>Interrupts</td>
<td>1 (Timer T2 underflow – Implements 20Khz ISR execution rate)</td>
</tr>
<tr>
<td>Peripheral Used</td>
<td>Timer T2/T3, EVMDAC, PWM7/8/9/10</td>
</tr>
<tr>
<td>PWM Frequency (PWMDAC)</td>
<td>20Khz (Timer 3 of Event manager-B)</td>
</tr>
<tr>
<td>PWM Mode</td>
<td>Symmetrical with no dead band</td>
</tr>
</tbody>
</table>
Figure 1. Software Test Bench (STB)
Software Test Bench (STB) overview

Figure 2. Software Flowchart

1. Initialize Software Modules
2. Initialize Real Time Monitor and Timer 2
3. Enable Timer 2 underflow Interrupt and Core Interrupt INT3
4. Background Loop
5. INT3 Interrupt
6. T2UF ISR
7. Save contexts and clear interrupt flag
8. Execute the Ramp generation module
9. Format Adjustment (Scaling and offsetting)
10. Execute the math function under test (MFUT)
11. Format Adjustment (Scaling and offsetting)
12. Update the DATALOG, EVMDAC and PWMDAC modules
13. Return
14. Restore contexts
2. MATH module directory structure

The math modules are available in “ASM only” and “CcA” interface, they are located under `alib` and `clib` respectively. All the “C” callable assembly modules are assembled and grouped together as an object library using archiver. Executing the batch file available in the “build” directory creates the object library. To use the CcA module in the project, add the object library to the project and include the header file (for math modules) that defines the module interface.
The Software Test Bench (STB) in C/ASM framework to demonstrate “ASM Only” and “CcA” math modules are available under astb and cstb directory respectively. The STB of each module contains three sub-directory viz., build, include & src.

- **built** directory has the work space files, make files for STB and also contains the map file and output file generated by the compiler.
- **include** directory contains the header files for STB.
- **src** directory contains the main system file (*.asm or *.C) and the compiler generated object file.

```
astb  ➔ <STB Name>  ➔ build  ➔ *.mak
       ➔          ➔ *.wks
       ➔          ➔ *.map
       ➔          ➔ *.out
       ➔          ➔ include ➔ *.h
       ➔          ➔ src     ➔ *.asm
                       ➔ *.obj

cstb  ➔ <STB Name>  ➔ build  ➔ *.mak
       ➔          ➔ *.wks
       ➔          ➔ *.map
       ➔          ➔ *.out
       ➔          ➔ include ➔ *.h
       ➔          ➔ src     ➔ *.asm
                       ➔ *.obj
```
3. “C” Framework configuration and warning messages

- **Selecting the Target**
  The ‘C/ASM’ framework assumes F2407 as default target. In order to execute the program using the F243 target, the user needs to edit the `target.h` header file located in the `include` directory of C-Software Test Bench. Here are the few lines of code relevant to this context extracted from the `target.h` header file.

```
/*----------------------------------------------------------
Following is the selection list of the target choices.
Note that the F241 is represented by the F243 and the
LF2407 represents the LF2406 and the LF2402.
---------------------------------------------------------------*
#define F240   1
#define F243   2
#define F2407  4
#define UNKNOWN 8

/*----------------------------------------------------------
This line sets the target to one of the available choices.
---------------------------------------------------------------*/
#define TARGET F2407
```

Modify the statement `#define TARGET F2407` to `#define TARGET F243` if you need to execute on the F243 target.

- **Selecting the Clock Frequency**
  The PLL used in the ‘24x device is hardwired to multiply-by-four (x4) mode, whereas the PLL used in the ‘240x device supports the programmability of multiplication factor from 0.4 to 4 by writing into the SCSR1 Register.

  By Default, the C/ASM framework assumes 7.5Mhz Input clock frequency for F2407 target and initializes the multiplication factor to 4 by writing into the SCSR1 register in the RstInit() routine. RstInit() routine is available in the main source file.

  The `period_max` parameter of the PWMDAC module is initialized to generate 20Khz PWM output by assuming 30MHz CPU Clock for the F2407 target.

  Time base of the Timer T2 is initialized to generate 20Khz interrupt by assuming the 30Mhz CPU clock for F2407 devices and 20Mhz CPU clock for F243 devices.

  In summary, followings are the three potential problems that will arise if you overlook the clock frequency information:
  1. F2407 will fail to work if the input clock frequency is more then 7.5Mhz as the SCSR1 is initialized to multiplication factor of 4, which means you are trying to clock the device at higher frequency then the rated operating frequency of 30Mhz for the F2407 device.
  2. PWMDAC module (Only for F2407) will generate different PWM frequency output instead of the intended 20Khz PWM output.
  3. Both the F2407 and F243 device will certainly execute the ISR at different frequency instead of 20Khz, this will have the repercussion on the generated ramp frequency.
Configuring the C Framework

Here are the few lines of code relevant to this context extracted from the `cdemo.h` header file.

```c
#warn Assumed CLKin \( \rightarrow \) 7.5 MHz for TMS320F240x, 5MHz for TMS320F24x
#warn Modify if not \( \rightarrow \) SYSTEM_INI_PERIOD,PWM_DAC_PERIOD,SCSR1_REG_VALUE
#if (TARGET==F243)
   /* ISR_EXE_PERIOD * CPU_CLK_FREQ=(1/20kz)*20Mhz=1000 */
   #define SYSTEM_INT_PERIOD  1000
#endif /* TARGET==F243 */
#if (TARGET==F2407)
   /* ISR_EXE_PERIOD * CPU_CLK_FREQ=(1/20kz)*30Mhz=1500 */
   #define SYSTEM_INT_PERIOD  1500
   /* (1/2)*PWM_PERIOD*CPU_CLK_FREQ=(1/2)*(1/20k)*30MHz=750 */
   #define PWM_DAC_PERIOD    750
   /* PLL MF=4, Enable EVA, Clear illegal address detect bit */
   #define SCSR1_REG_VALUE     005
#endif /* TARGET==F2407 */
#warn
Set the REAL_TIME symbolic constant to 1, for real time debugging
#define  REAL_TIME 1
```

To rectify the problems mentioned earlier, modify the value of the following symbolic constants to commensurate with the CPU clock frequency

1. SCSR1_REG_VALUE
2. SYSTEM_INT_PERIOD
3. PWM_DAC_PERIOD

Two warning messages are deliberately placed in the `cdemo.h` header file to alert the user to modify the symbolic constants

- **Enable/Disable the Real Time Mode**

  The C/ASM framework allows the user to either enable/disable the real time mode by setting the REAL_TIME symbolic constant. The effect of disabling the Real time mode is that the graph window will no longer be refreshed continuously. That means there is no exchange of data between the PC and the Target when the program runs.

  Here are the relevant lines of code extracted from the main source file (*.c) of the C framework, located in the `src` directory of C demo. Warning message is deliberately introduced to alert the user in advance

```c
#warn Set the REAL_TIME symbolic constant to 1, for real time debugging
#define  REAL_TIME 1
```

Three warning messages will be generated when you compile the C framework.
Configuring the ASM framework

4. ASM Framework Configuration

ASM framework needs the same kind of configuration as explained for the C framework. Followings are the three settings that need to be modified based on the target.

1. Selecting the Target
2. Selecting the Clock frequency
3. Enable/Disable the Real Time Mode

The following section exemplifies the ASM framework configuration

• Selecting the Target

Select the target device by setting the corresponding constant to 1 as given below in the **x24x_app.h** header file, located in the **include** directory of ASM-Software Test Bench.

```assembly
; Select the target device by setting 1
;--------------------------------------------------------------
x240 .set 0 ; C/F240
x243 .set 0 ; C/F243
x2407 .set 1 ; F2407
''```

• Selecting the Clock frequency

Modify the value of the following constants in the **asmdemo.h** header file, located in the **include** directory of ASM-Software Test Bench to commensurate with the CPU clock frequency

1. SCSR1_REG_VALUE
2. SYSTEM_INT_PERIOD
3. PWM_DAC_PERIOD

```assembly
.if x243
; ISR_EXE_PERIOD * CPU_CLK_FREQ=(1/20kz)*20Mhz=1000
SYSTEM_INT_PERIOD .set 1000
.endif

.if x2407
; ISR_EXE_PERIOD * CPU_CLK_FREQ=(1/20kz)*30Mhz=1500
SYSTEM_INT_PERIOD .set 1500
; PLL MF=4, Enable EVA, Clear illegal address detect bit
SCSR1_REG_VALUE .set 005
.endif
```

Modify the constant **PWM_DAC_PERIOD** in **pwm_dac.asm** source file, located in the **src** directory of ASM-Software Test Bench in pro rata with the CPU clock frequency

```assembly
; PWM_DAC_PERIOD=(1/2)*PWM_PERIOD*CPU_CLK_FREQ=(1/2)*(1/20k)*30Mhz=750
PWM_DAC_PERIOD .set 750
```
• Enable/Disable the Real Time Mode

Modify the constant `REAL_TIME` in the main system source file (*.asm), located in the `src` directory of ASM-Software Test Bench.

```assembly
;*************************************************************** *******
;   SYSTEM OPTIONS
;***************************************************************
REAL_TIME .set 1 ; 1 for real time mode, otherwise set 0
;***************************************************************
```
5. Hardware Setup

5.1. F2407 Evaluation Module (EVM)

- EVMDAC outputs are unidirectional and varies from 0 to 3.3v

5.2. F243 Evaluation Module (EVM)

- EVMDAC outputs are unidirectional and varies from 0 to 5v
6. Loading and Building CC Project for C/ASM framework

The workspace file (*.wks) and make file (*.mak) for both ASM framework and C framework to demonstrate the Math functions are located in the build directory of the respective modules within the astb/cstb directory. The CC workspace file, contains the setup information for the whole project and the debugging environment such as the graph window properties, watch window parameters, break points and probe points etc. It facilitates the user to save and restore the same environment between debugging sessions instead of reconfiguring the working environment again and again for each debugging session.

- To quickly execute demo using the pre-configured work environment, load the workspace file from the build directory

Loading the workspace file will automatically open up the project file (*.mak) for the corresponding project and show all the files relevant to the project in the FILEVIEW tab.

- From the Option menu choose Program Load’ and enable the Load Program After Build’ option. This allows the automatic loading of the program to the target DSP once the program is compiled

- From the Project menu choose ‘Rebuild All’ or the ‘Rebuild All’ shortcut on the toolbar to compile the program and load it to the target.

Once this is done, the expanded project view as part of the CC environment will be as shown in figure 3, if you have loaded sin.wks for C demo. Graph window on the TOP is configured to plot the data available from location 8000h to 83FFh, similarly the Graph window on the BOTTOM is configured to plot the data available from location 8400h to 87FF. Upon power up these memory locations will have garbage data and hence the plot on the graph window.

- From the Debug menu choose ‘Real Time Mode’

On some occasions CC will fail to switch to Real Time mode. If that happens, then run the code, halt the processor and reset it. Once again choose the Real Time Mode

- After selecting Real Time Mode, run the software by choosing Run from the Debug menu or using the tool bar shortcut.

- Right Click on the TOP Graph Window and choose continuous refresh, graph window should be displaying a RAMP at this time. Similarly choose continuous refresh for the BOTTOM graph window, the graph window should be displaying the math function output. Figure 4 shows the project view when the C demo is executed for QSIN function
Figure 3. CC Project View of QSIN demo using C framework

Figure 4. QSIN demo using C-framework
7. Software Test Bench (STB)

7.1. Software Test Bench (STB) for QSIN & QSINLT

The RAMP_GEN module is configured to generate 50Hz asymmetric ramp output that swings between +1 and –1. The ramp output is directly connected to the SIN function as it is in Q15 format. The Q15 output of the QSIN function and the output of RAMP_GEN module are sent through the PWM_DAC_DRV, DAC_VIEW_DRV and also logged by the DATA_LOG module. The logged information is graphed on the console for observation, by the Real Time Monitor program running in the background. The DAC_VIEW_DRV output can be viewed with the scope and the PWM_DAC_DRV output should be filtered to view the signal.

Figure 6 & 7 shows the CC project view of QSIN function demo in C & ASM framework respectively.
Figure 6. CC Project view of QSIN demo in C-Framework

Figure 7. CC Project view of QSIN demo in ASM framework
7.2. Software Test Bench (STB) for QCOS/QCOSLT

The RAMP_GEN module is configured to generate 50Hz asymmetric ramp output that swings between +1 and –1. The ramp output is directly connected to the COS function as it is in Q15 format. The Q15 output of the QCOS function and the output of the RAMP_GEN module are sent through the PWM_DAC_DRV, DAC_VIEW_DRV and also logged by the DATA_LOG module. The logged information is graphed on the console for observation, by the Real Time Monitor program running in the background. The DAC_VIEW_DRV output can be viewed with the scope and the PWM_DAC_DRV output should be filtered to view the signal.

Figure 9 & 10 shows the CC project view of QCOS function demo in C & ASM framework respectively.
Figure 9. CC Project view of QCOS demo in C framework

Figure 10. CC Project view of QCOS demo in ASM framework
7.3. Software Test Bench (STB) for QATAN

The RAMP_GEN module is configured to generate 50Hz asymmetric ramp output that swings between +1 and –1. The Q15 output of the RAMP_GEN module is scaled to sweep the range from –10 to +10 and the scaled Q15 number is adjusted to Q16 format by shifting up 1 bit in order to input it to the QATAN function. User can change the range by modifying the symbolic constant SCALE_FACTOR (SF) in the main source file, which is set to 10 as the default parameter.

The Q15 output of the QATAN function and the output of RAMP_GEN module are sent through the PWM_DAC_DRV, DAC_VIEW_DRV and also logged by the DATA_LOG module. The logged information is graphed on the console for observation, by the Real Time Monitor program running in the background. The DAC_VIEW_DRV output can be viewed with the scope and the PWM_DAC_DRV output should be filtered to view the signal.

Figure 12 & 13 shows the CC project view of QATAN function demo in C & ASM framework respectively.
Figure 12. CC Project view of QATAN demo in C-framework

Figure 13. CC Project view of QATAN demo in ASM framework
7.4. Software Test Bench (STB) for QSQRT

The RAMP_GEN module is configured to generate 50Hz asymmetric ramp output that swings between +1 and –1. The Q15 output of the RAMP_GEN module is offset by +1 so that the ramp will be sweeping from 0 to 2 (Q15). This positive ramp is scaled by 65535 to cover the range from 0 to 65535 in unsigned 16.16 format in order to input it to the QSQRT function. User can change the range by modifying the symbolic constant SCALE_FACTOR (SF) in the main source file, which is set to 65535 as the default parameter.

The unsigned 8.8 format output of the QSQRT function is converted to signed 8.8 format by adding negative offset in order to output the result using the PWM_DAC_DRV and DAC_VIEW_DRV. The signed 8.8 output and the output of RAMP_GEN module are sent through PWM_DAC_DRV, DAC_VIEW_DRV and also logged by the DATA_LOG module. The logged information is graphed on the console for observation, by the Real Time Monitor program running in the background. The DAC_VIEW_DRV output can be viewed with the scope and the PWM_DAC_DRV output should be filtered to view the signal.

Figure 15 & 16 shows the CC project view of QSQRT function demo in C & ASM framework respectively.
Figure 15. CC Project view of SQRT demo in C-framework

Figure 16. CC Project view of SQRT demo in ASM framework
The RAMP_GEN module is configured to generate 50Hz asymmetric ramp output that swings between +1 and –1. The Q15 output of the RAMP_GEN module is offset by +1 so that the ramp will be sweeping from 0 to 2 (Q15). This positive ramp is scaled by 5 to cover the range from 0 to 5 in unsigned 16.16 format in order to input it to the QLOG10 function. User can change the range by modifying the symbolic constant SCALE_FACTOR (SF) in the main source file, which is set to 5 as the default parameter.

The signed 4.12 output of QLOG10 function and the output of RAMP_GEN module are sent through PWM_DAC_DRV, DAC_VIEW_DRV and also logged by the DATA_LOG module. The logged information is graphed on the console for observation, by the Real Time Monitor program running in the background. The DAC_VIEW_DRV output can be viewed with the scope and the PWM_DAC_DRV output should be filtered to view the signal.

Figure 18 & 19 shows the CC project view of QLOG10 function demo in C & ASM framework respectively.
Figure 18. CC Project view of QLOG10 demo in C-framework

Figure 19. CC Project view of QLOG10 in ASM framework
7.6. Software Test Bench (STB) for QLOGN

The RAMP_GEN module is configured to generate 50Hz asymmetric ramp output that swings between +1 and –1. The Q15 output of the RAMP_GEN module is offset by +1 so that the ramp will be sweeping from 0 to 2 (Q15). This positive ramp is scaled by 5 to cover the range from 0 to 5 in unsigned 16.16 format in order to input it to the QLOGN function. User can change the range by modifying the symbolic constant SCALE_FACTOR (SF) in the main source file, which is set to 5 as the default parameter.

The signed 5.11 output of QLOGN function and the output of RAMP_GEN module are sent through PWM_DAC_DRV, DAC_VIEW_DRV and also logged by the DATA_LOG module. The logged information is graphed on the console for observation, by the Real Time Monitor program running in the background. The DAC_VIEW_DRV output can be viewed with the scope and the PWM_DAC_DRV output should be filtered to view the signal.

Figure 21 & 22 shows the CC project view of QLOGN function demo in C & ASM framework respectively.
Figure 21. CC Project view of QLOGN demo in C-framework

Figure 22. CC Project view of QLOGN demo in ASM framework
The RAMP_GEN module is configured to generate 50Hz asymmetric ramp output that swings between +1 and –1. The Q15 output of the RAMP_GEN module is scaled by \( \frac{1}{2^8} \) to cover the range from \(-1/256\) to \(+1/256\) in Q15 format. User can change the range by modifying the symbolic constant SCALE_FACTOR (SF) in the main source file, which is set to 8 as the default parameter. The Acceptable value for the scale factor is from 0 to 15. For a given SCALE_FACTOR, the scaling value is \( \frac{1}{2^{SF}} \).

The signed 32-bit (16.16) output of the QINV1 is scaled down by 65536 to convert into signed 16 bit number (16.0) and sent through PWM_DAC_DRV, DAC_VIEW_DRV and also logged by the DATA_LOG module along with the RAMP_GEN output. The logged information is graphed on the console for observation, by the Real Time Monitor program running in the background. The DAC_VIEW_DRV output can be viewed with the scope and the PWM_DAC_DRV output should be filtered to view the signal.

Figure 24 & 25 shows the CC project view of QINV1 function demo in C & ASM framework respectively.
Figure 24. CC Project view of QINV1 demo in C framework

Figure 25. CC Project view of QINV1 inASM framework
7.8. Software Test Bench (STB) for QINV2

The RAMP_GEN module is configured to generate 50Hz asymmetric ramp output that swings between +1 and –1. The Q15 output of the RAMP_GEN module is scaled by 1/128 to cover the range from –1/128 to +1/128 in Q15 format. User can change the range by modifying the symbolic constant SCALE_FACTOR (SF) in the main source file, which is set to 8 as the default parameter. The Acceptable value for the scale factor is from 0 to 15. For a given SCALE_FACTOR, the scaling value is:

\[
\text{Scaling Value} = \frac{1}{2^{\text{SCALE \_ FACTOR}}}
\]

The output of the QINV2 and the output of RAMP_GEN module are sent through PWM_DAC_DRV, DAC_VIEW_DRV and also logged by the DATA_LOG module. The logged information is graphed on the console for observation, by the Real Time Monitor program running in the background. The DAC_VIEW_DRV output can be viewed with the scope and the PWM_DAC_DRV output should be filtered to view the signal.

Figure 27 & 28 shows the CC project view of QINV2 function demo in C & ASM framework respectively.
Figure 27. CC Project view of QINV2 demo in C-framework

Figure 28. CC Project view of QINV2 demo in ASM framework