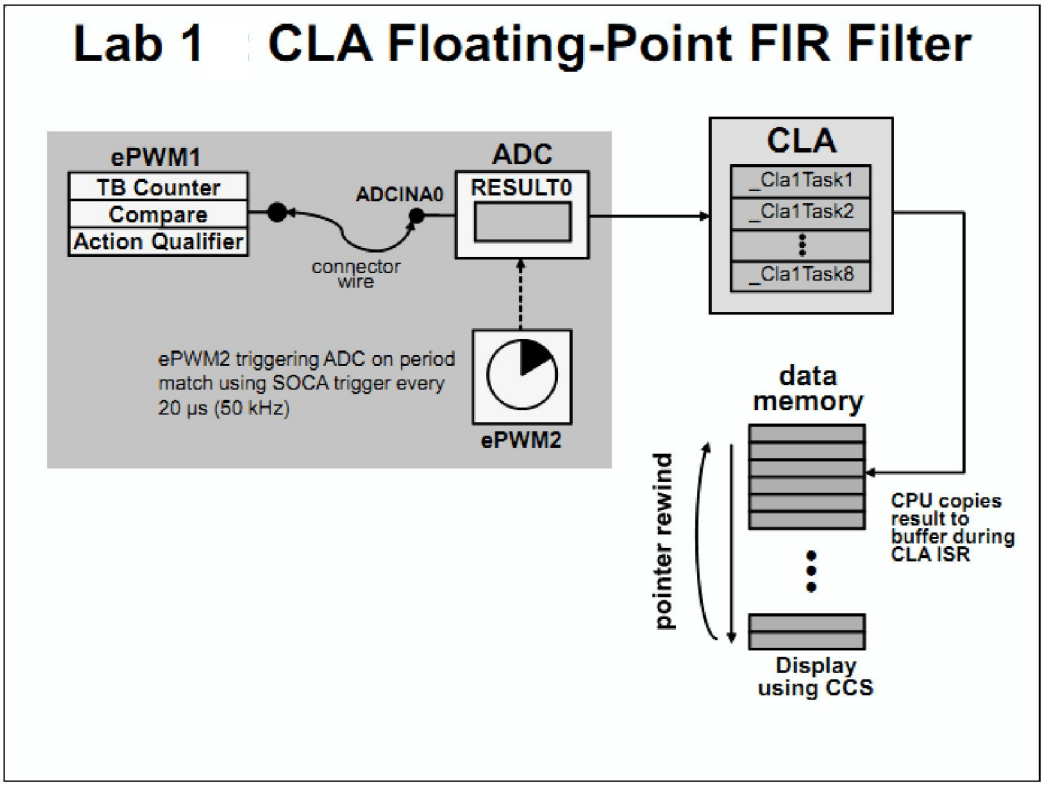
Lab 1: How to use CLA

The objective of this lab is to become familiar with operation of the CLA. In the previous lab, the CPU was used to filter the ePWM1A generated 2 kHz, 25% duty cycle symmetric PWM waveform. In this lab, the PWM waveform will be filtered using the CLA. The CLA will directly read the ADC result register and a task will run a low-pass FIR filter on the sampled waveform. The filtered result will be stored in a circular memory buffer. Note that the CLA is operating concurrently with the CPU. As an operational test, the filtered and unfiltered waveforms will be displayed using the graphing feature of Code Composer Studio.



Enabling CLA Support in CCS

1. Open the build options by right-clicking on 2806xCLA in the C/C++ Projects window and select Properties. Then select the "**C/C++ Build**" Category. Be sure that the Tool Settings tab is selected.

2. Under "**C2000 Compiler**" select "**Runtime Model Options**". Notice the "**Specify CLA support**" is set to cla0. This is needed to assemble CLA code. Click OK to close the Properties window.

Inspect Lab.cmd

Open and inspect Lab.cmd. Notice that a section called "**Cla1Prog**" is being linked to L3DPSARAM. This section links the CLA program tasks (assembly code) to the CPU memory space. This memory space will be remapped to the CLA memory space during initialization.

Also, notice the two message RAM sections used to pass data between the CPU and CLA.
We are linking CLA code directly to the CLA program RAM because we are not yet using the flash memory. CCS will load the code for us into RAM, and therefore the CPU will not need to copy the CLA code into the CLA program RAM. In the next lab, we will modify the linking so that the CLA code is loaded into flash, and the CPU will do the copy.

Setup CLA Initialization

During the CLA initialization, the CPU memory block L3DPSARAM needs to be configured as CLA program memory. This memory space contains the CLA Task routines, which are coded in assembly. The CLA Task 1 has been configured to run an FIR filter. The CLA needs to be configured to start Task 1 on the ADCINT1 interrupt trigger. The next section will setup the PIE interrupt for the CLA.

Open ClaTasks.asm and notice that the .cdecls directive is being used to include the C header file in the CLA assembly file. Therefore, we can use the Peripheral Register Header File references in the CLA assembly code. Next, notice Task 1 has been configured to run an FIR filter. Within this code special instructions have been used to convert the ADC result integer (i.e. the filter input) to floating-point and the floating-point filter output back to integer.

Edit Cla.c to implement the CLA operation as described in the objective for this lab exercise. Configure the L3DPSARAM memory block to be mapped to CLA program memory space. Set Task 1 peripheral interrupt source to ADCINT1 and set the other Task peripheral interrupt source inputs to no source. Enable CLA Task 1 interrupt.

Open Main_10.c and add a line of code in main() to call the InitCla() function. There are no passed parameters or return values. You just type

InitCla();

at the desired spot in main().

In Main_10.c comment out the line of code in main() that calls the InitDma() function. The DMA is no longer being used. The CLA will directly access the ADC RESULT0 register.

Setup PIE Interrupt for CLA

Recall that ePWM2 is triggering the ADC at a 50 kHz rate. For this lab exercise, the ADC is triggering the CLA, and the CLA will directly read the ADC result register and run a task implementing an FIR filter. The CLA will generate an interrupt to the CPU, which will store the filtered results to a circular buffer implemented in the CLA ISR.

1. Remember that in Adc.c we commented out the code used to enable the ADCINT1 interrupt in PIE group 1. This is no longer being used. The CLA interrupt will be used instead.

2. Using the "PIE Interrupt Assignment Table" find the location for the CLA Task 1 interrupt "CLA1_INT1" and fill in the following information:

PIE group #: *# within group:*

This information will be used in the next step.

3. Modify the end of Cla.c to do the following:
- Enable the "**CLA1_INT1**" interrupt in the PIE (Hint: use the PieCtrlRegs structure)
- Enable the appropriate core interrupt in the IER register

4. Open and inspect DefaultIsr.c. Notice that this file contains the CLA interrupt service routine. Save and close all modified files.

Build and Load

1. Click the "**Build**" button and watch the tools run in the Console window. Check for errors in the Problems window.

2. Click the "**Debug**" button (green bug). The "**Debug Perspective**" view should open, the program will load automatically, and you should now be at the start of main(). If the device has been power cycled since the last lab exercise, be sure to configure the boot mode to EMU_BOOT_SARAM using the Scripts menu.

Run the Code – Test the CLA Operation

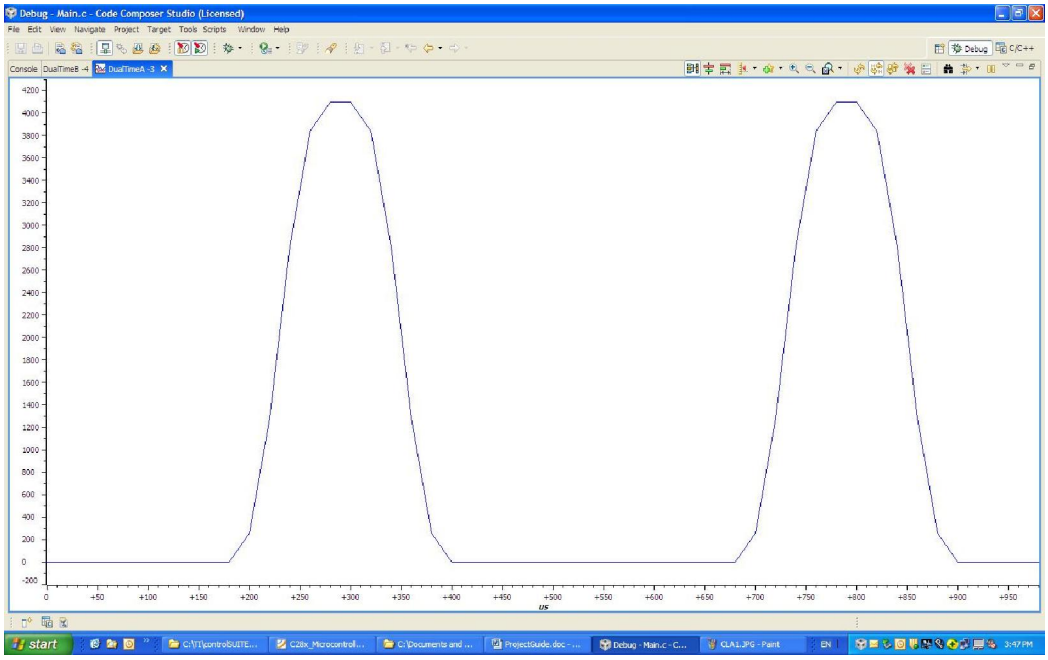
Note: For the next step, check to be sure that the jumper wire connecting **PWM1A (pin # GPIO-00)** to **ADCINA0 (pin # ADC-A0)** is in place on the Docking Station.

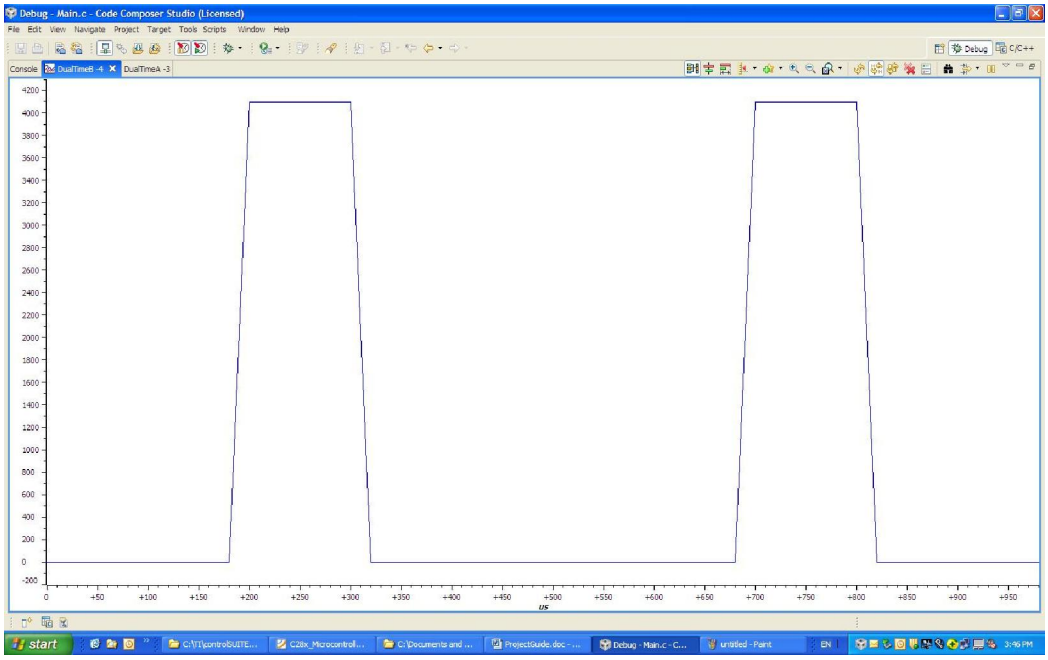
1. Run the code in real-time mode using the Script function: **Scripts -> Realtime Emulation Control -> Run_Realtime_with_Reset**, and watch the memory window update. Verify that the ADC result buffer contains updated values.

2. Setup a dual-time graph of the filtered and unfiltered ADC results buffer. Click: **Tools -> Graph -> Dual Time** and set the following values:

Acquisition Buffer Size	50
DSP Data Type	16-bit unsigned integer
Sampling Rate (Hz)	50000
Start Address A	AdcBufFiltered
Start Address B	AdcBuf
Display Data Size	50
Time Display Unit	µs

3. The graphical display should show the filtered PWM waveform in the Dual Time A display and the unfiltered waveform in the Dual Time B display.

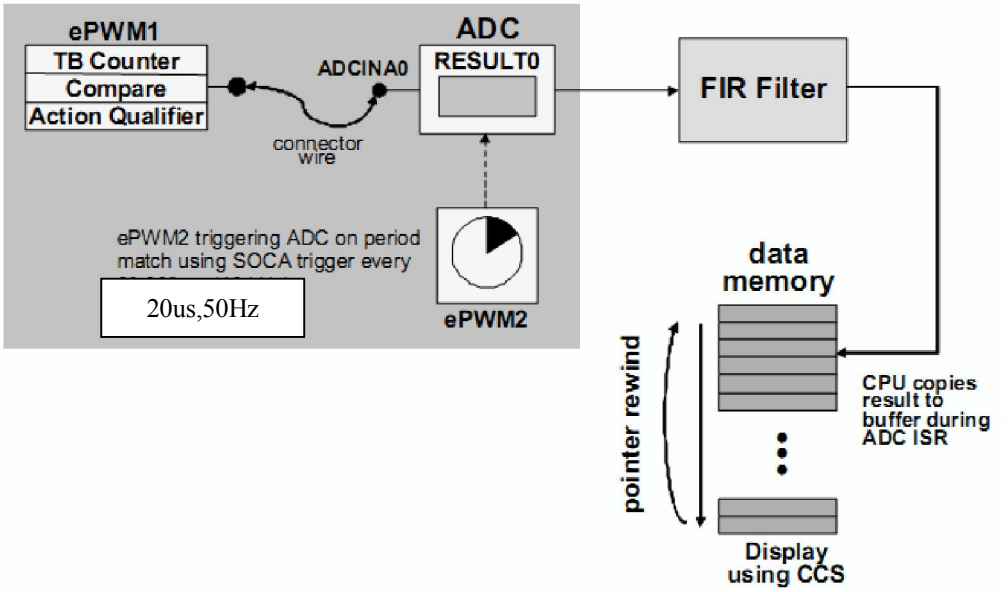




4. Fully halt the CPU (real-time mode) by using the Script function: **Scripts -> Realtime Emulation Control -> Full_Halt.**

Lab 2: How to use IQmath & Floating-Point FIR Filter

The objective of this lab is to become familiar with IQmath and floating-point programming. ePWM1A was setup to generate a 2 kHz, 25% duty cycle symmetric PWM waveform. The waveform was then sampled with the on-chip analog-to-digital converter. In this lab the sampled waveform will be passed through an FIR filter and displayed using the graphing feature of Code Composer Studio. The filter math type (IQmath and floating-point) will be selected in the "IQmathLib.h" file.



Project Build Options

1. Setup the build options by right-clicking on Lab8 in the C/C++ Projects window and select Properties. Then select the "C/C++ Build" Category. Be sure that the Tool Settings tab is selected.

2. We need to setup the include search path to include the IQmath header file. Under "**C2000 Compiler**" select "**Include Options**". In the box that opens click the Add icon (first icon with green plus sign). Then in the "Add directory path" window type:

${PROJECT_ROOT}/../IQmath/include

Click OK to include the search path.

3. Next, we need to setup the library search path to include the IQmath library. Under "**C2000 Linker**" select "**File Search Path**". In the top box click the Add icon. Then in the "Add file path" window type:

${PROJECT_ROOT}/../IQmath/lib/IQmath.lib

Click OK to include the library file.

In the bottom box click the Add icon. In the "**Add directory path**" window type:

${PROJECT_ROOT}/../IQmath/lib

Click OK to include the library search path.
Finally, select OK to save and close the build options window.

Include IQmathLib.h

In the C/C++ Projects window edit Lab.h and uncomment the line that includes the IQmathLib.h header file. Next, in the Function Prototypes section, uncomment the function prototype for IQssfir(), the IQ math single-sample FIR filter function. In the Global Variable References section uncomment the four _iq references, and comment out the reference to AdcBuf[ADC_BUF_LEN]. Save the changes and close the file.

Select a Global IQ value

In the C/C++ Projects window under the Includes folder open: IQmathLib.h. Confirm that the GLOBAL_Q type (near beginning of file) is set to a value of 24. If it is not, modify as necessary:

```
#define     GLOBAL_Q          24
```

Recall that this Q type will provide 8 integer bits and 24 fractional bits. Dynamic range is therefore -128 < x < +128, which is sufficient for our purposes in the workshop.
Notice that the math type is defined as IQmath by:

```
#define     MATH_TYPE         IQ_MATH
```

Close the file.

IQmath Single-Sample FIR Filter

1. Open and inspect DefaultIsr.c. Notice that the ADCINT_ISR calls the IQmath single-sample FIR filter function, IQssfir(). The filter coefficients have been defined in the beginning of Main.c. Also, as discussed in the lecture for this module, the ADC results are read with the following instruction:

```
*AdcBufIQPtr=_IQmpy(ADC_FS_VOLTAGE,_IQ12toIQ((_iq)AdcResult.ADCRESULT0
));
```

2. Open and inspect Lab.h. Notice that, as discussed in the lecture for this module, ADC_FS_VOLTAGE is defined as:

```
#if MATH_TYPE == IQ_MATH
      #define ADC_FS_VOLTAGE      _IQ(3.0)
 #else          // MATH_TYPE is FLOAT_MATH
      #define ADC_FS_VOLTAGE      _IQ(3.0/4096.0)
#endif
```

3. Open and inspect the IQssfir() function in Filter.c. This is a simple, non-optimized coding of a basic IQmath single-sample FIR filter. Close the inspected files.

Build and Load

1. Click the "**Build**" button and watch the tools run in the Console window. Check for errors in the Problems window.

2. Click the "**Debug**" button (green bug). The "**Debug Perspective**" view should open, the program will load automatically, and you should now be at the start of main().

Run the Code – Filtered Waveform

1.Open a memory window to view some of the contents of the filtered ADC results buffer. The address label for the filtered ADC results buffer is AdcBufFiltered in the "Data" memory page. Set the format to 32-Bit Signed Integer. Right-click in the memory window, select Configure… and set the Q-Value to 24 (which matches the IQ format being used for this variable). Then click OK to save the setting. We will be running our code in real-time mode, and will need to have the window continuously refresh.

Note: For the next step, check to be sure that the jumper wire connecting **PWM1A (pin # GPIO-00) to ADCINA0 (pin # ADC-A0)** is in place on the Docking Station.

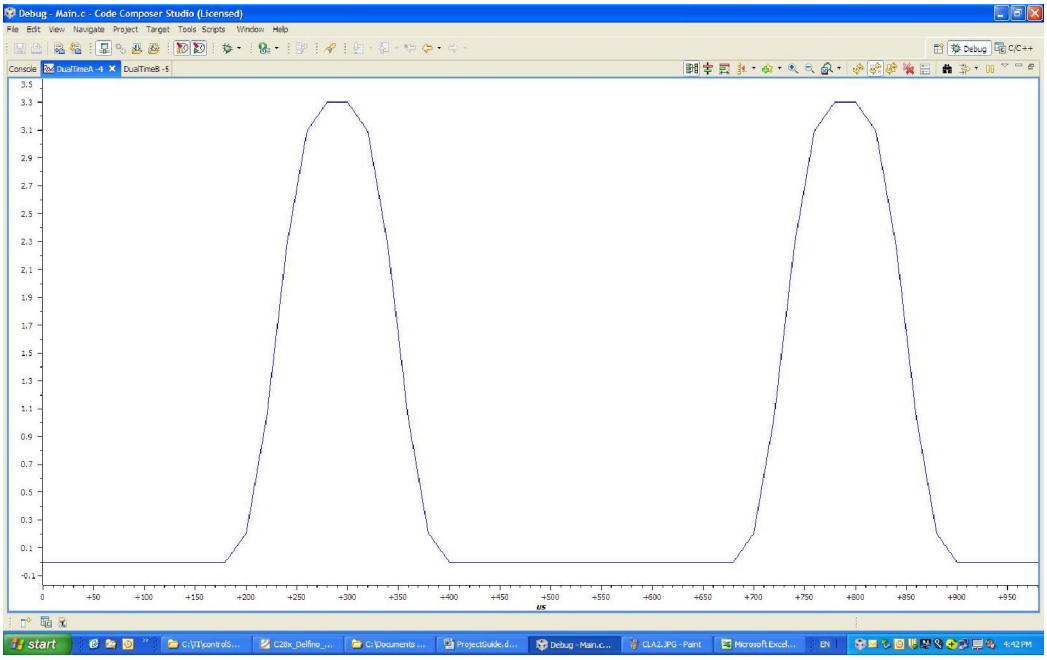
2.Run the code in real-time mode using the Script function: **Scripts -> Realtime Emulation Control -> Run_Realtime_with_Reset**, and watch the memory window update. Verify that the ADC result buffer contains updated values.

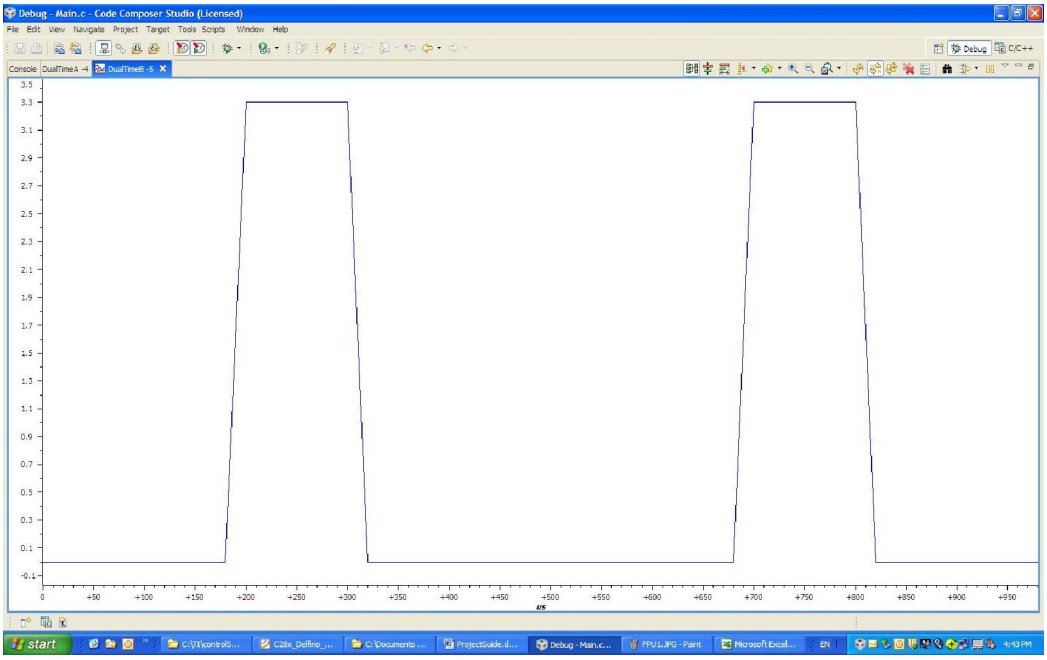
3.Open and setup a dual-time graph to plot a 48-point window of the filtered and unfiltered ADC results buffer. Click: **Tools -> Graph -> Dual Time** and set the following values:

Acquisition Buffer Size	50
DSP Data Type	32-bit signed integer
Q Value	24
Sampling Rate (Hz)	50000
Start Address A	AdcBufFiltered
Start Address B	AdcBuf
Display Data Size	50
Time Display Unit	µs

Select OK to save the graph options.

4. The graphical display should show the generated FIR filtered 2 kHz, 25% duty cycle symmetric PWM waveform in the Dual Time A display and the unfiltered waveform generated in the previous lab exercise in the Dual Time B display. Notice the shape and phase differences between the waveform plots (the filtered curve has rounded edges, and lags the unfiltered plot by several samples). The amplitudes of both plots should run from 0 to 3.0.



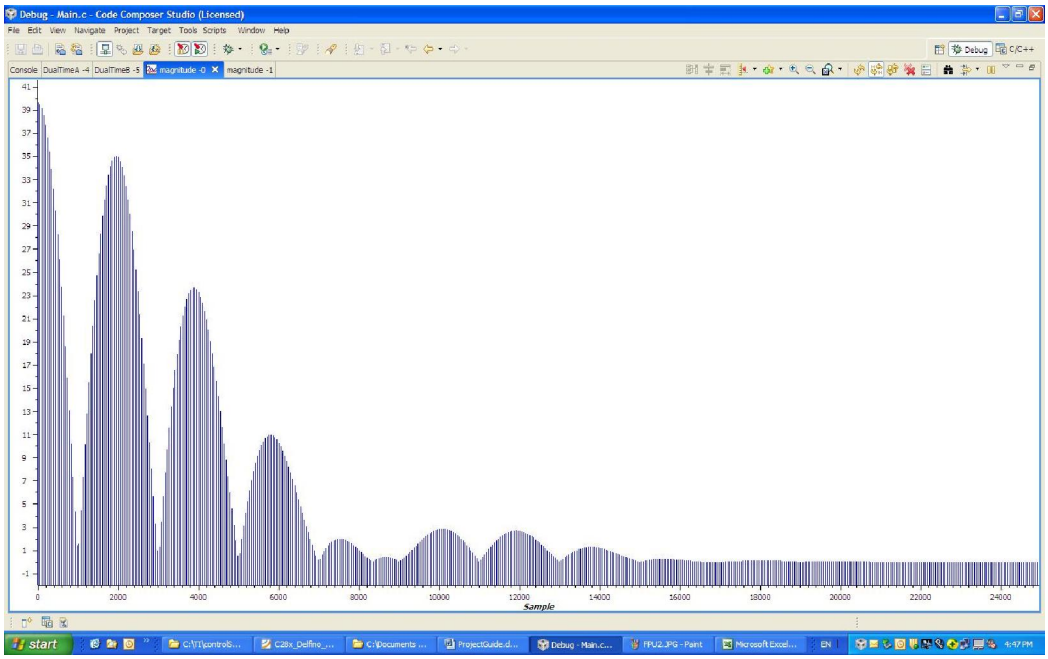


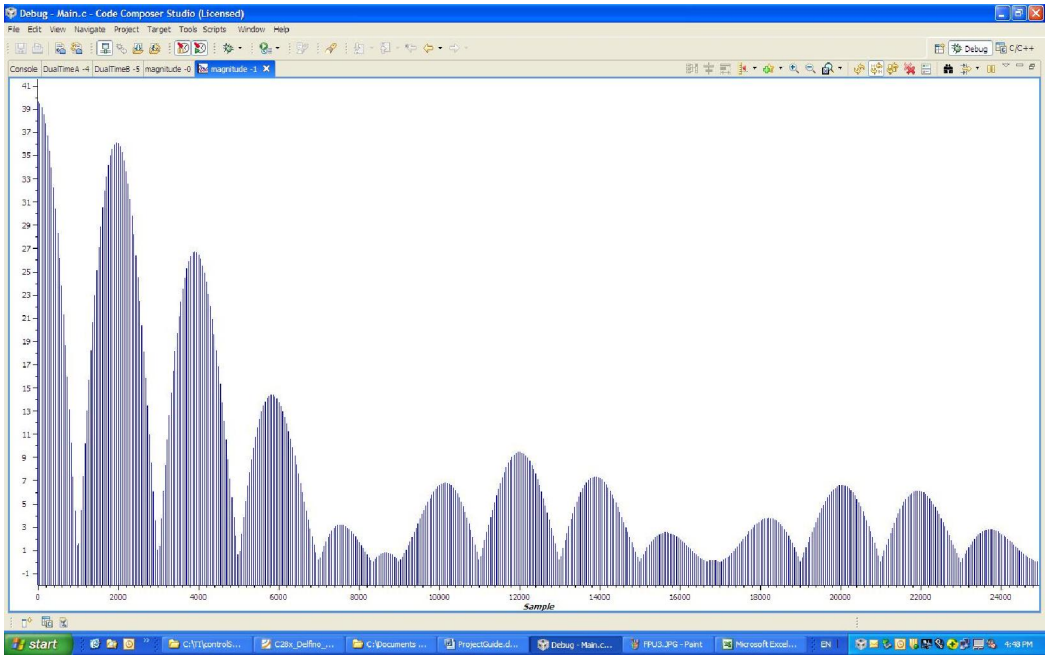
5. Open and setup two frequency domain plots – one for the filtered and another for the unfiltered ADC results buffer. Click: **Tools -> Graph -> FFT** Magnitude and set the following values:

	GRAPH #1	GRAPH #2
Acquisition Buffer Size	50	50
DSP Data Type	32-bit signed integer	32-bit signed integer
Q Value	24	24
Sampling Rate (Hz)	50000	50000
Start Address	AdcBufFiltered	AdcBuf
Data Plot Style	Bar	Bar
FFT Order	10	10

Select OK to save the graph options.

6. The graphical displays should show the frequency components of the filtered and unfiltered 2 kHz, 25% duty cycle symmetric PWM waveforms. Notice that the higher frequency components are reduced using the Low-Pass FIR filter in the filtered graph as compared to the unfiltered graph.





7. Fully halt the CPU (real-time mode) by using the Script function: **Scripts -> Realtime Emulation Control -> Full_Halt**.

Changing Math Type to Floating-Point

Switch to the "C/C++ Perspective" view by clicking the C/C++ icon in the upper right-hand corner. In the C/C++ Projects window under the Includes folder open: IQmathLib.h. Edit IQmathLib.h to define the math type as floating-point. Change #define

from: ***#define MATH_TYPE IQ_MATH***

to: ***#define MATH_TYPE FLOAT_MATH***

Save the change to the IQmathLib.h and close the file.

Run the Code – Floating-Point Filtered Waveform

1. Change the dual-time and FFT Magnitude graphs to display 32-bit floating-point rather than 32-bit signed integer. Click the "**Show the Graph Properties**" icon for each graph and change the DSP Data Type to 32-bit floating-point.

2. Run the code (real-time mode) by using the Script function: **Scripts -> Realtime Emulation Control -> Run_Realtime_with_Reset**.

3. The graphical display should show the generated FIR filtered 2 kHz, 25% duty cycle symmetric PWM waveform in the Dual Time A display and the unfiltered waveform in the Dual Time B display. The FFT Magnitude graphical displays should show the frequency components of the filtered and unfiltered 2 kHz, 25% duty cycle symmetric PWM waveforms.

4. Fully halt the CPU (real-time mode) by using the Script function: **Scripts -> Realtime Emulation Control -> Full_Halt**.

Terminate Debug Session and Close Project

1. Terminate the active debug session using the Terminate All button. This will close the debugger and return CCS to the "C/C++ Perspective" view.

2. Next, close the project by right-clicking on Lab8 in the C/C++ Projects window and select Close Project.

Lab 3 How to use VCU

The Texas Instruments TMS320C28x Viterbi, Complex Math and CRC Unit (VCU) Library is a collection of highly optimized application functions written for the C28x + VCU. These functions enable the programmer to accelerate the performance of communications-based algorithms by up to a factor of 8X over C28x alone.

Open the Project

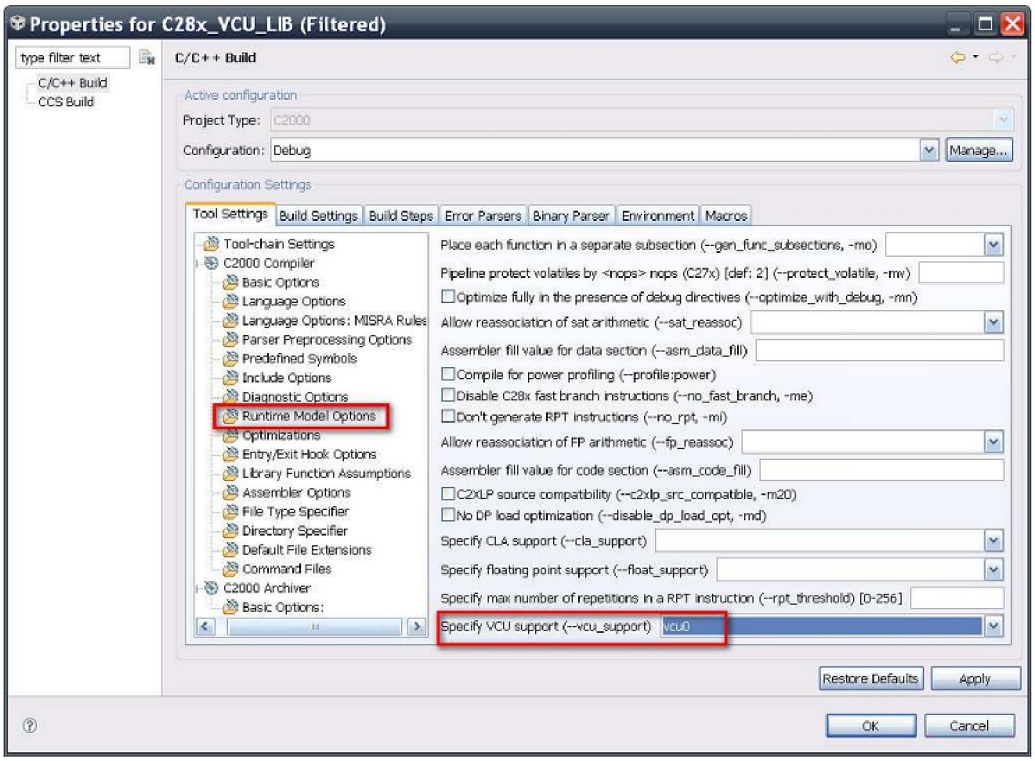
A project named **2806x_CFFT_64p** has been created for this lab. Open the project by clicking on **Project -> Import Existing CCS/CCE Eclipse Project**. The "**Import**" window will open then click Browse… next to the "**Select root directory**" box. Navigate to: **2806xProject FPU&CLA&VCU\2806xVCU\v100\2806x_CFFT_64p** and click OK. Then click Finish to import the project. All build options have been configured the same as the previous lab.

Project Build Options

The library was built using C28x codegen tools v6.0.1 in CCS v4.2.2 with the following options:

-v28 -mt -ml -g --keep_unneeded_statics --vcu_support=vcu0

You can specify VCU support in the Runtime Model Options menu of the compiler's build property page (see red highlighted boxes in figure below)



NOTE: Codegen v6.0.1 (and above) is available through the Software Updates option in the help menu. Older versions of CCS (4.2.1 and below) will not display the vcu_support option.

Build and Load

Click the "Build" button and watch the tools run in the Console window. Check for errors in the Problems window.

Click the "Debug" button (green bug). The "Debug Perspective" view should open, the program will load automatically, and you should now be at the start of main(). If the device has been power cycled since the last lab exercise, be sure to configure the boot mode to EMU_BOOT_SARAM using the Scripts menu.

Run the Code

Run the code in real-time mode using the Script function: **Scripts -> Realtime Emulation Control -> Run_Realtime_with_Reset**.

Terminate Debug Session and Close Project

Terminate the active debug session using the **Terminate All** button. This will close the debugger and return CCS to the "C/C++ Perspective" view.

Next, close the project by right-clicking on **2806x_CFFT_64p** in the C/C++ Projects window and select Close Project.