D. Richard Brown III
Associate Professor
Worcester Polytechnic Institute
Electrical and Computer Engineering Department
drb@ece.wpi.edu

October 19-20, 2009

DIGITAL SIGNAL PROCESSING AND APPLICATIONS WITH THE TMS320C6713 DSK

Day 1 handouts

WPI

Workshop Goals

- Correctly install Texas Instruments Code Composer Studio IDE and DSK drivers
- Become familiar with
  - DSP basics
  - TMS320C6713 floating point DSP architecture
  - TMS320C6713 DSP starter kit (DSK)
  - Code composer studio integrated development environment (IDE)
  - Matlab design and analysis tools
- Learn how to program the C6713
  - Writing and compiling code
  - Fixing errors
  - Downloading code to the target and executing
  - Debugging
- Write and run useful programs on the C6713 DSK
- Learn about DSP applications
- Learn where to find help

Take Home Items

- "*Digital Signal Processing and Applications with the C6713 and C6416 DSK*" by Chassaing & Reay, 2008
- Texas Instruments TMS320C6713 DSK including
  - DSK board with TMS320C6713 DSP chip
  - USB cable
  - Power supply
  - CD with Code composer studio IDE (v3.1) and electronic documentation
  - DSK technical reference manual
  - DSK quick start installation guide
  - Matlab/Simulink trial CD and other promotional material

C6713 DSK Overview

- 225 MHz TMS320C6713 *floating point* DSP
- AIC23 stereo codec (ADC and DAC)
  - Ideal for audio applications
  - 8-96 kHz sample rates
- Memory
  - 16 MB dynamic RAM
  - 512 kB nonvolatile FLASH memory
- General purpose I/O
  - 4 LEDs
  - 4 DIP switches
- USB interface to PC

C6713 DSK Functional Block Diagram

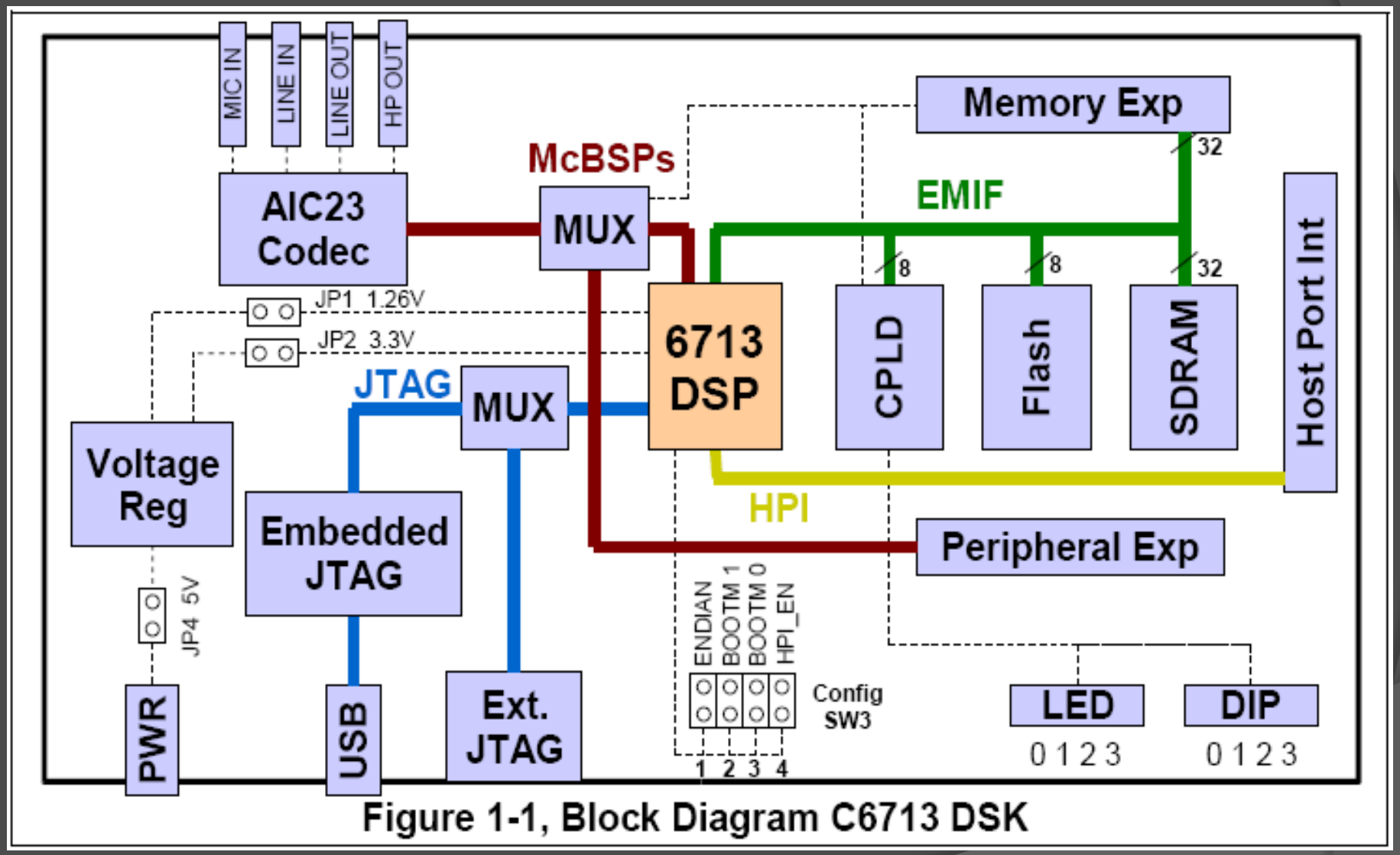
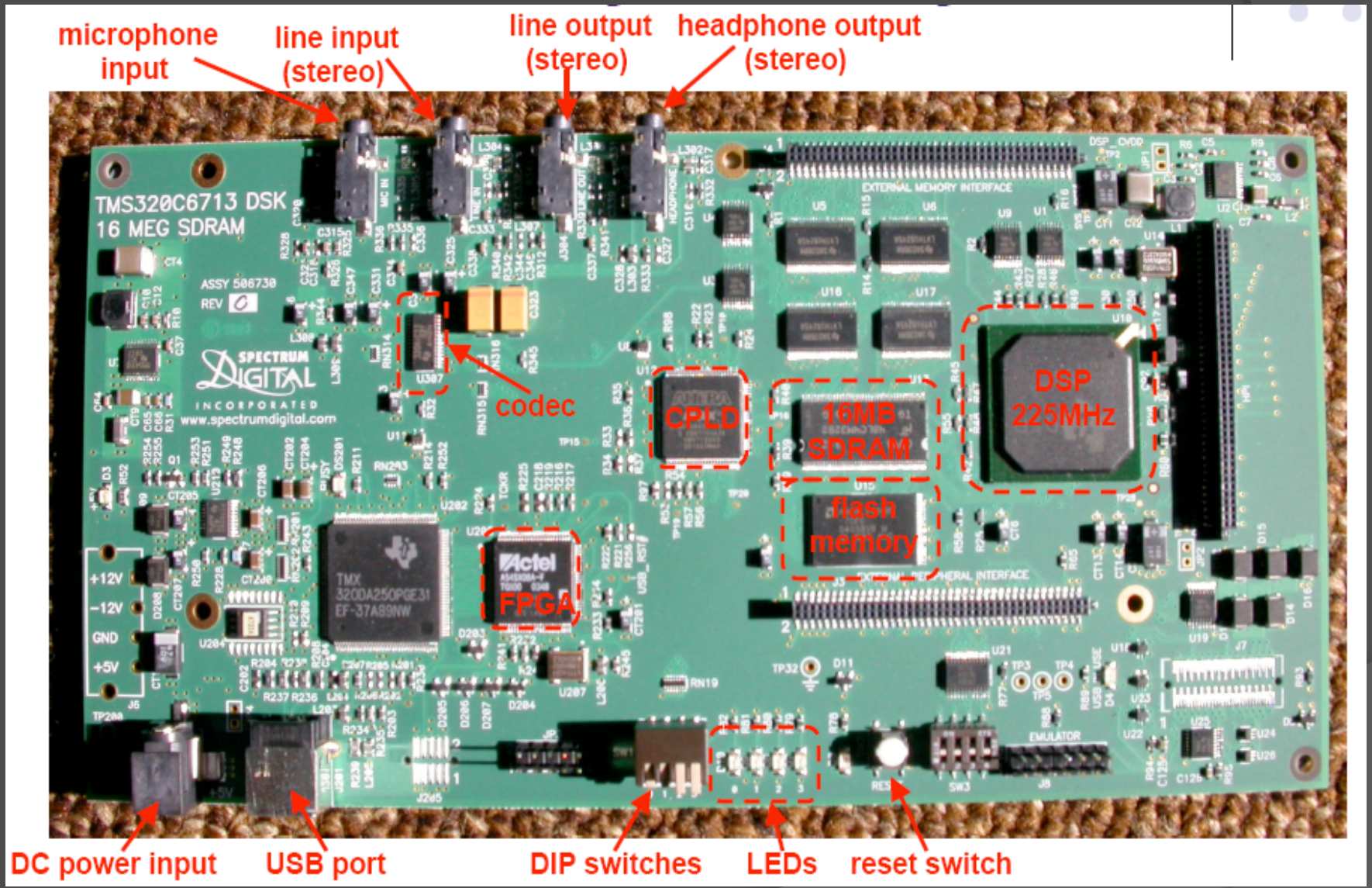


Figure 1-1, Block Diagram C6713 DSK

C6713 DSK Physical Layout

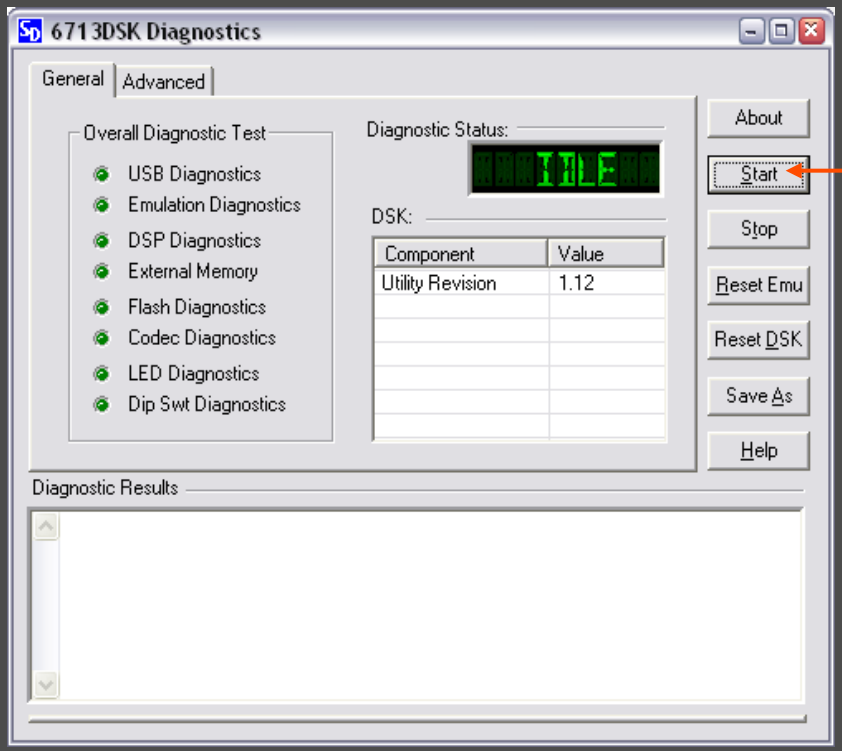


Is my DSK working?
DSK Power On Self Test

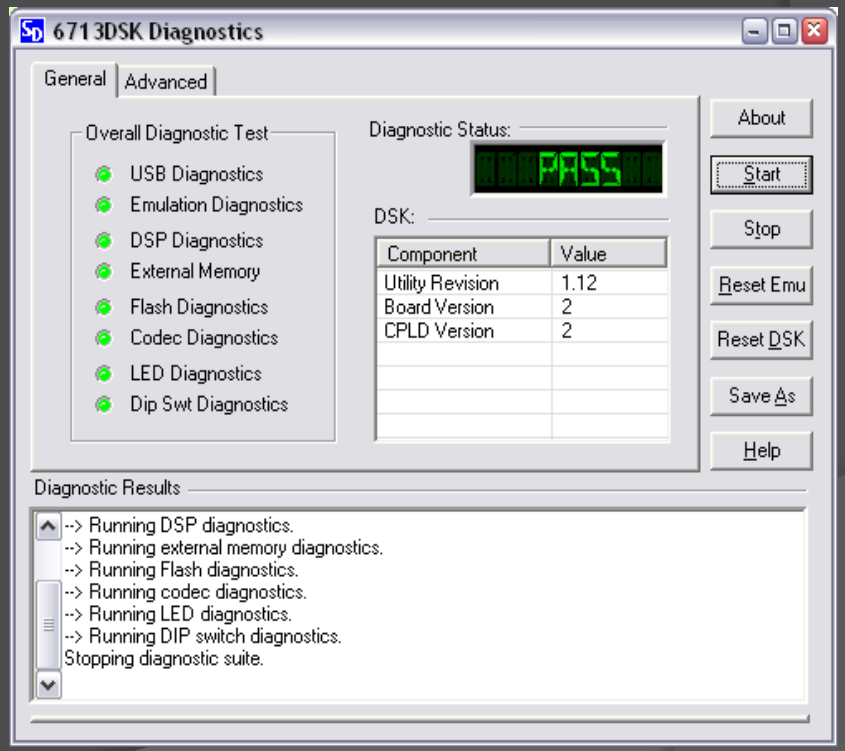
- Power up DSK and watch LEDs
- Power On Self Test (POST) program stored in FLASH memory automatically executes
- POST takes 10-15 seconds to complete
- All DSK subsystems are automatically tested
- During POST, a 1kHz sinusoid is output from the AIC23 codec for 1 second
  - Listen with headphones or watch on oscilloscope
- If POST is successful, all four LEDs blink 3 times and then remain on

Is my DSK working? DSK Diagnostic Utility

- Install CCS 3.1
  - Directions in "Quick Start Installation Guide"
  - More detailed directions available on spinlab web site
  - Diagnostic utility automatically installed

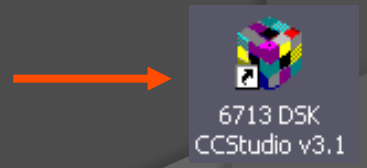


press start



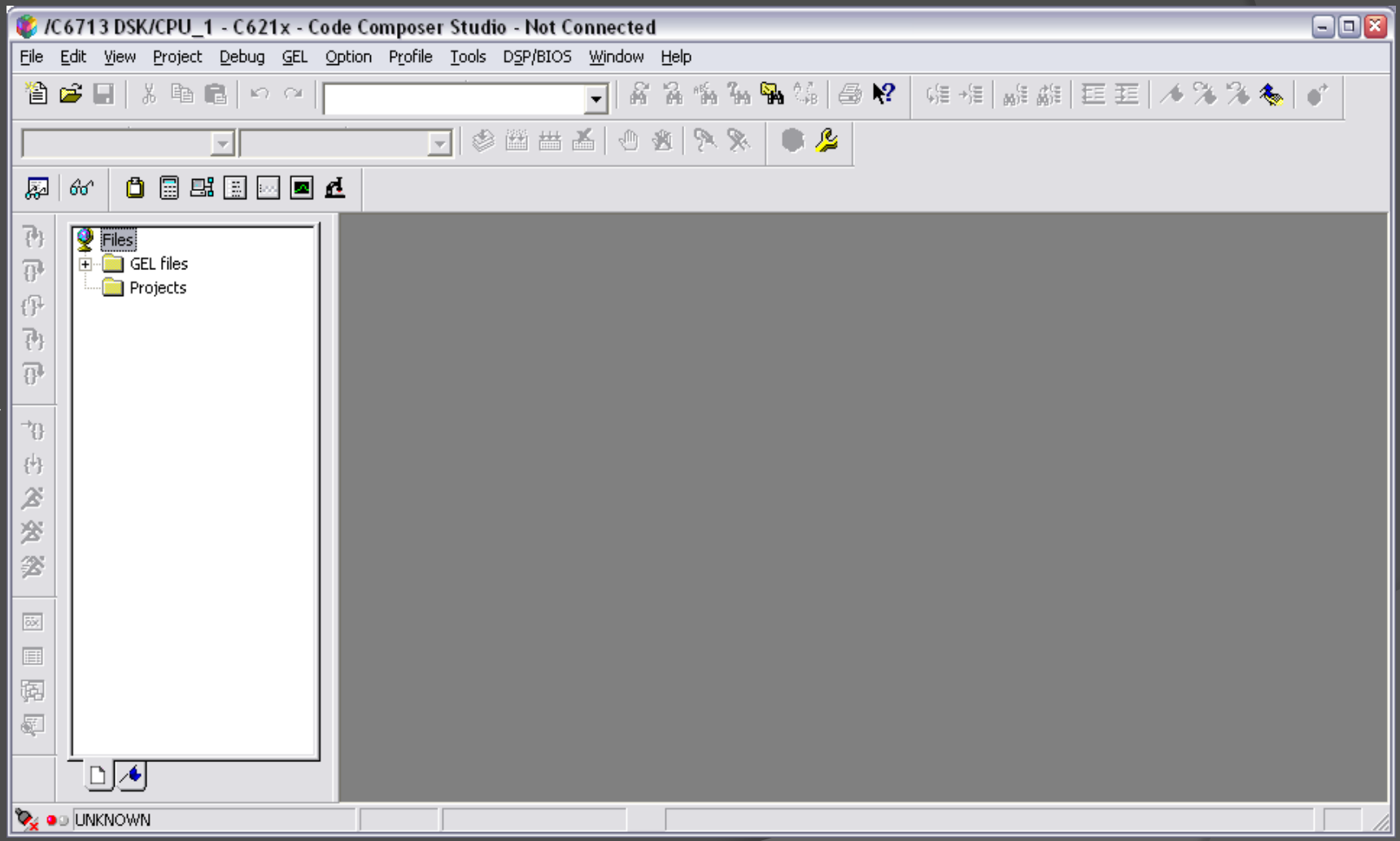
Code Composer Studio IDE

- Connect power supply to DSK
- Wait for POST to complete
- Connect USB cable from PC to DSK
  - If this is the first time connecting the DSK, you may be asked to install a driver. The driver is on the Code Composer Studio CD and will automatically be found by Windows if the CD is in the drive.
- Launch Code Composer Studio C6713 DSK

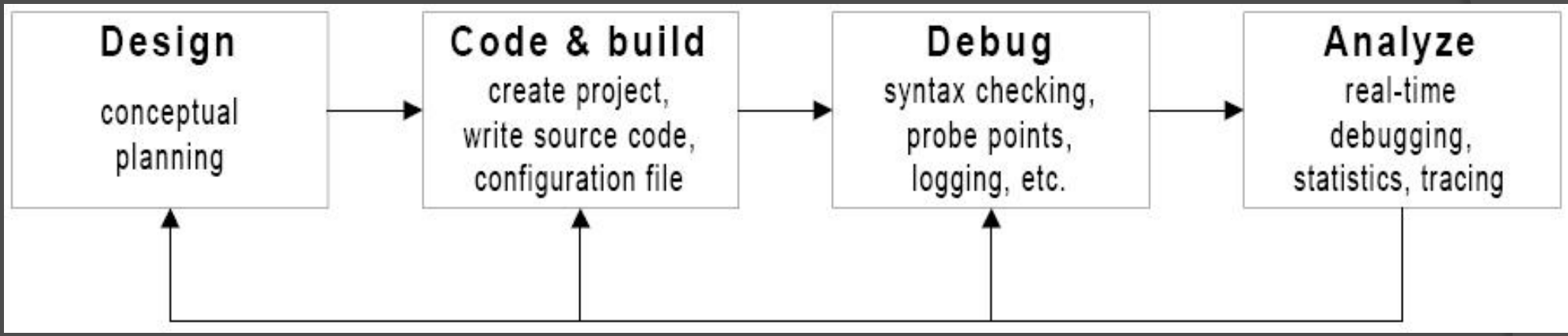


- CCS will load and wait for your input

Code Composer Studio IDE



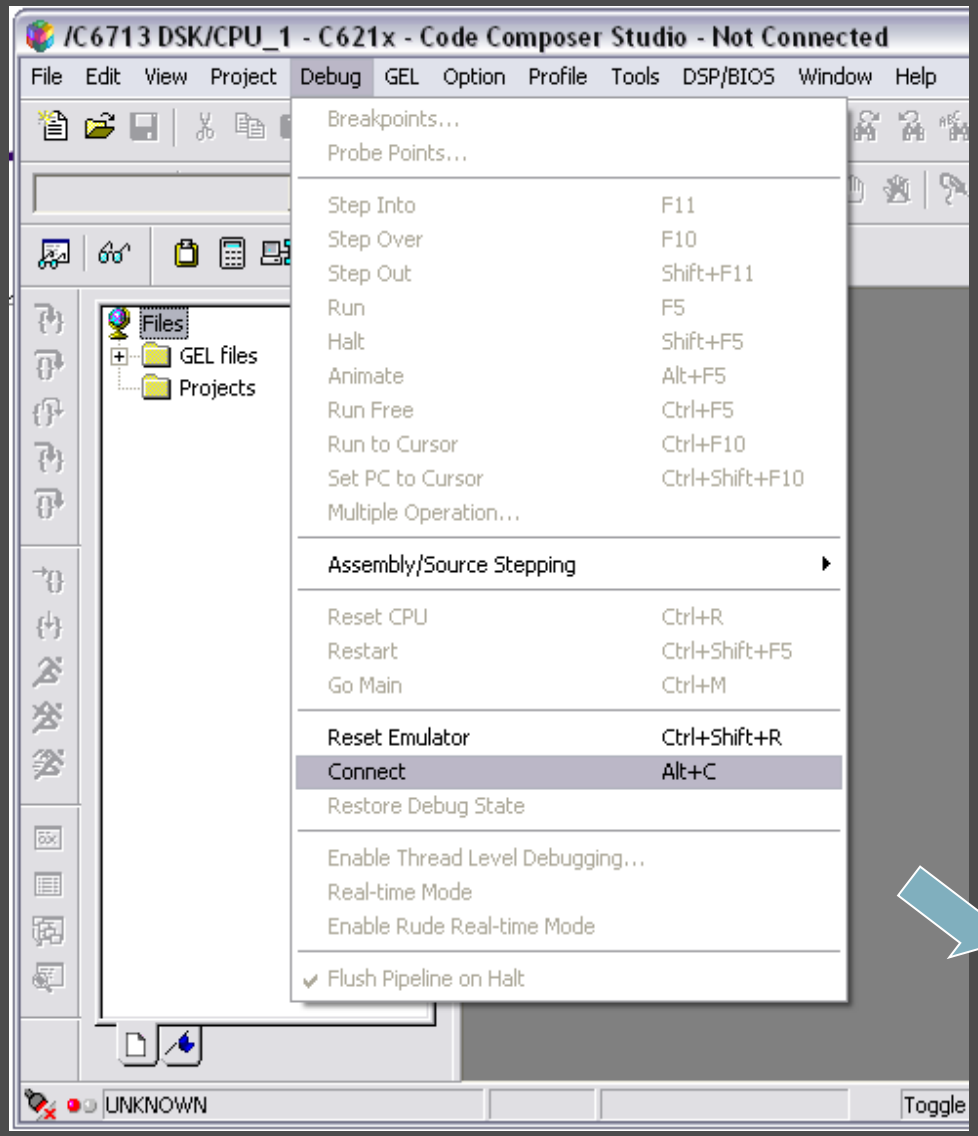
CCS Integrated Development Environment

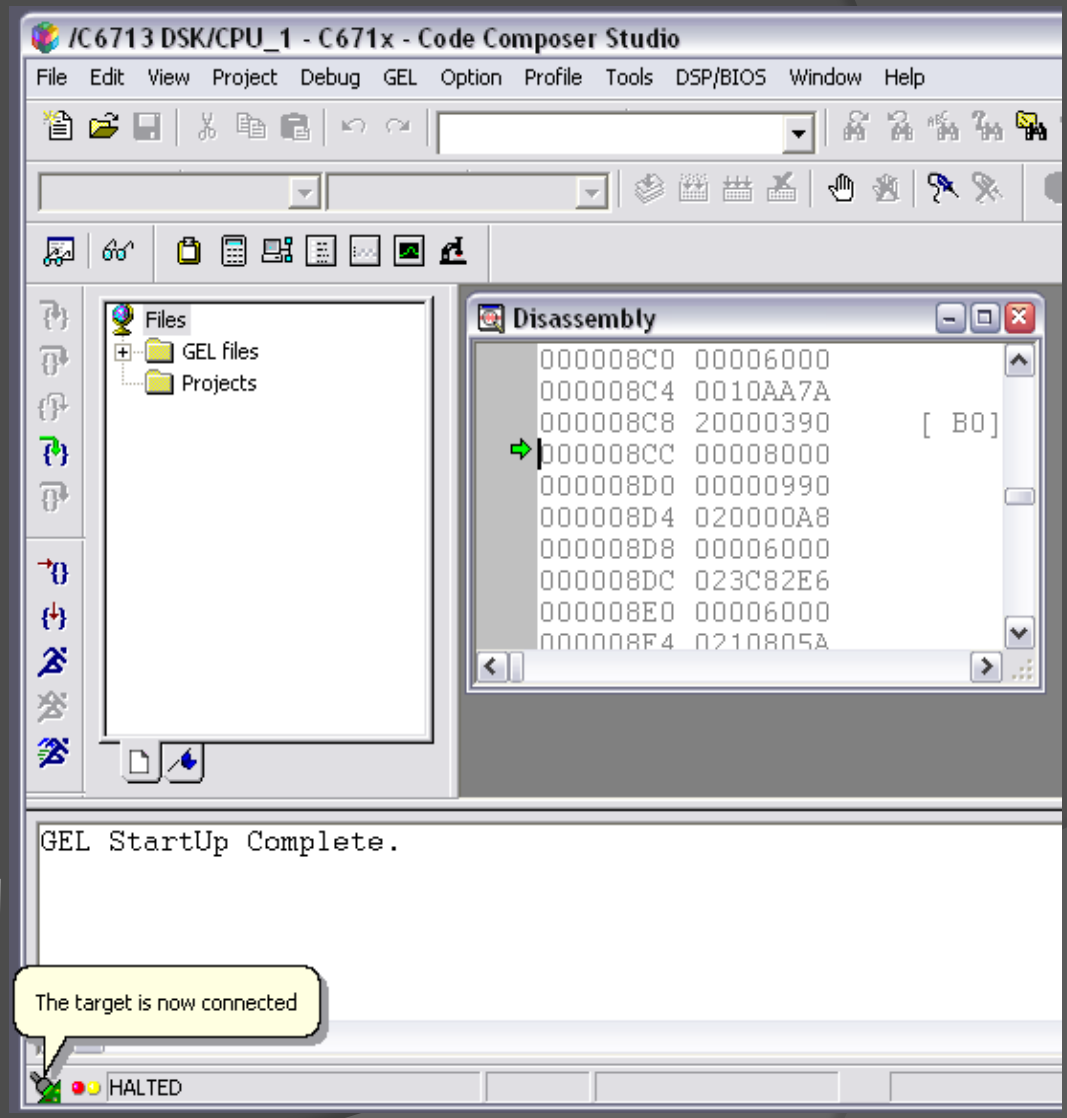


Useful TI documentation (available online or on your hard drive):
SPRU509F.PDF CCS v3.1 IDE Getting Started Guide
C6713DSK.HLP C6713 DSK specific help material

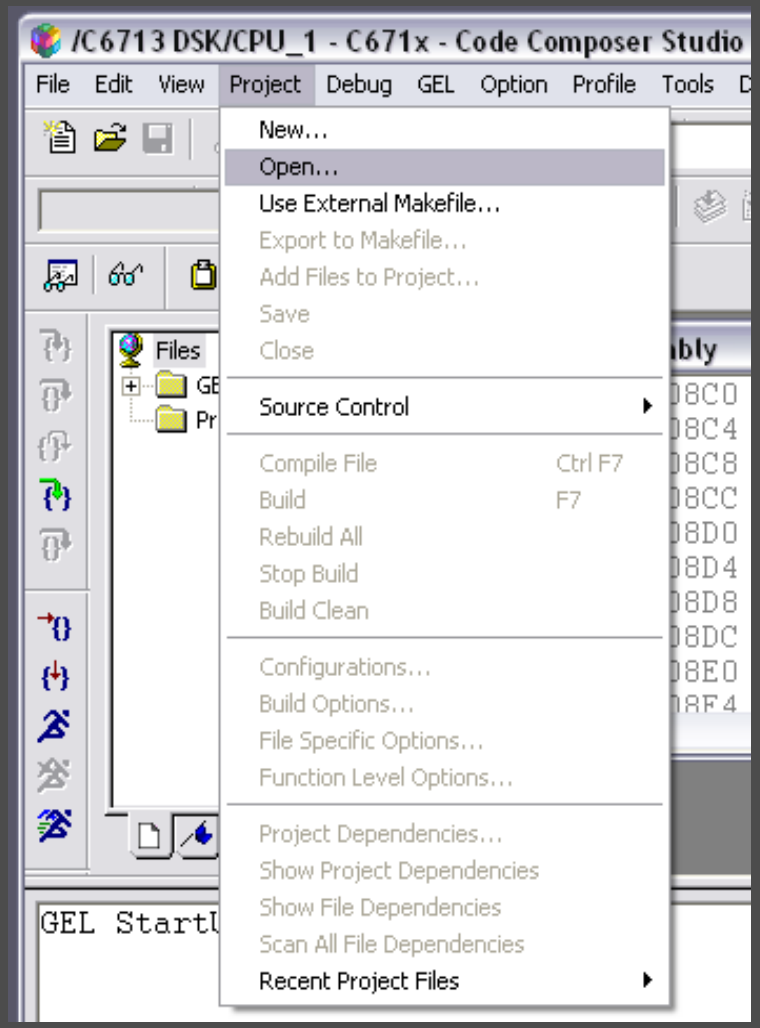
Note that your DSK includes CCS v3.1.

Connecting to the C6713 DSK

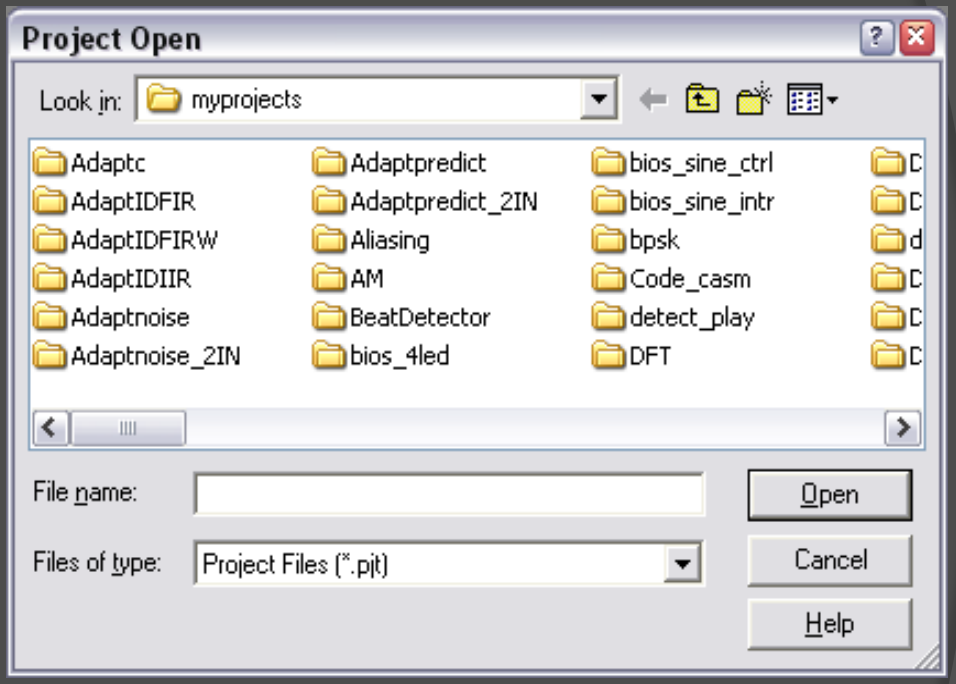




Opening an Existing Project



Project->Open

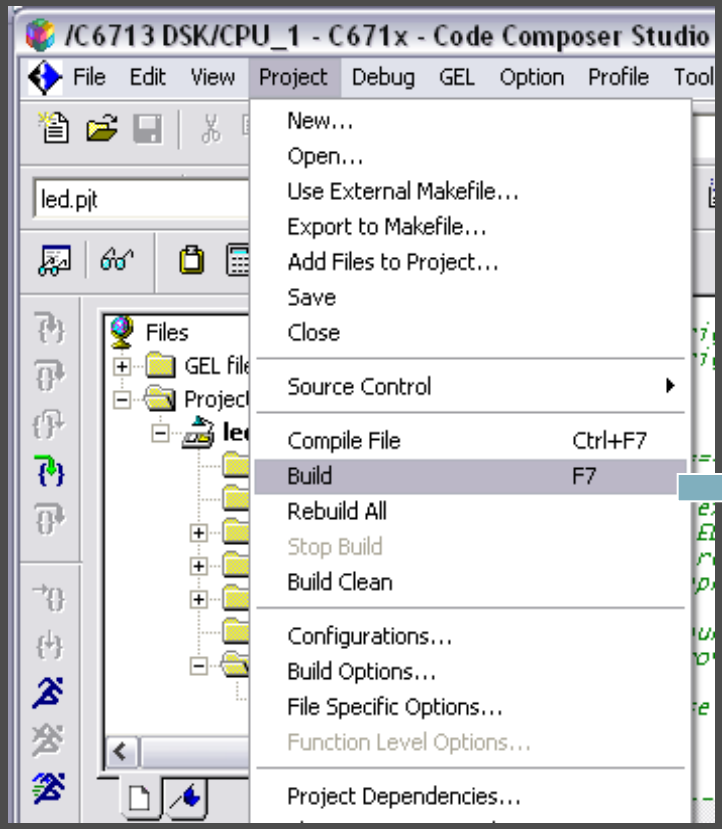


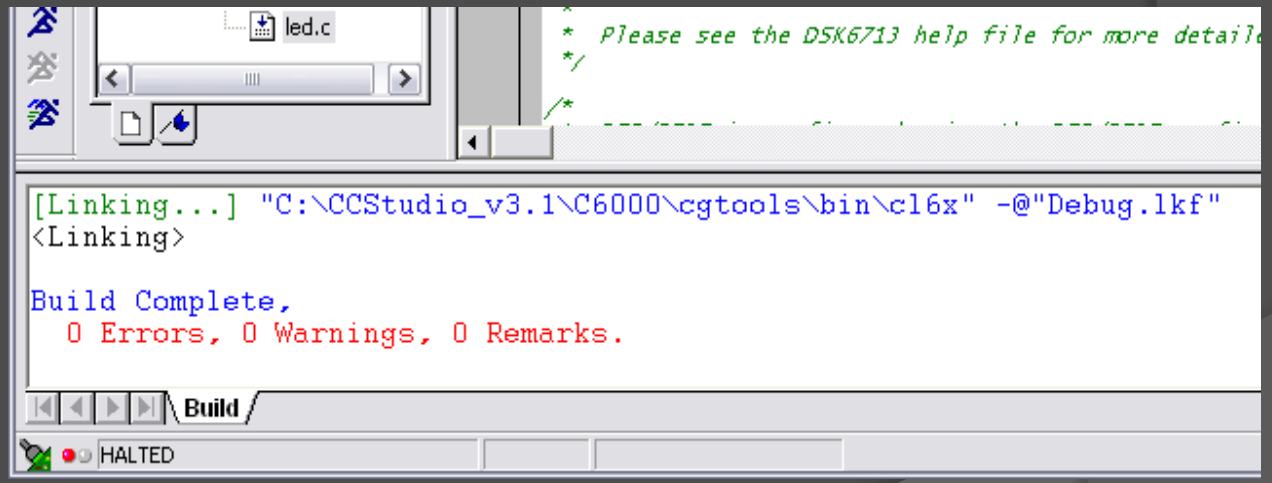
Select a .PJT file and press "Open". Chassaing example projects should be in c:\CCStudio_v3.1\myprojects\

Other example projects for the C6713 can be found in c:\CCStudio_v3.1\examples\dsk6713

Compiling/Building a Project

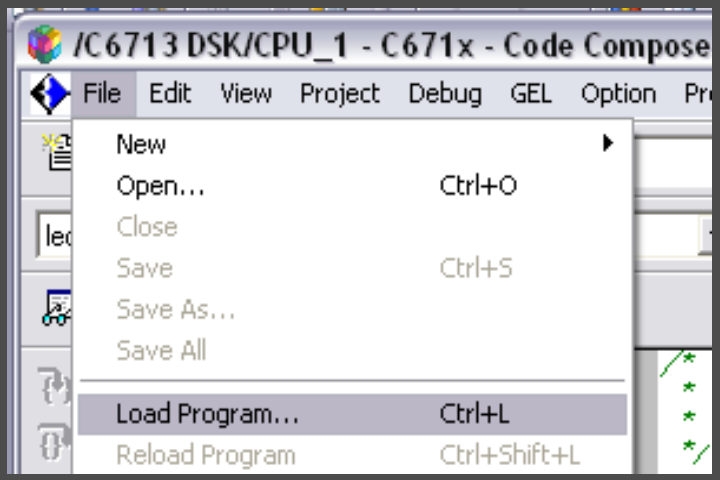
Project->Build (F7)

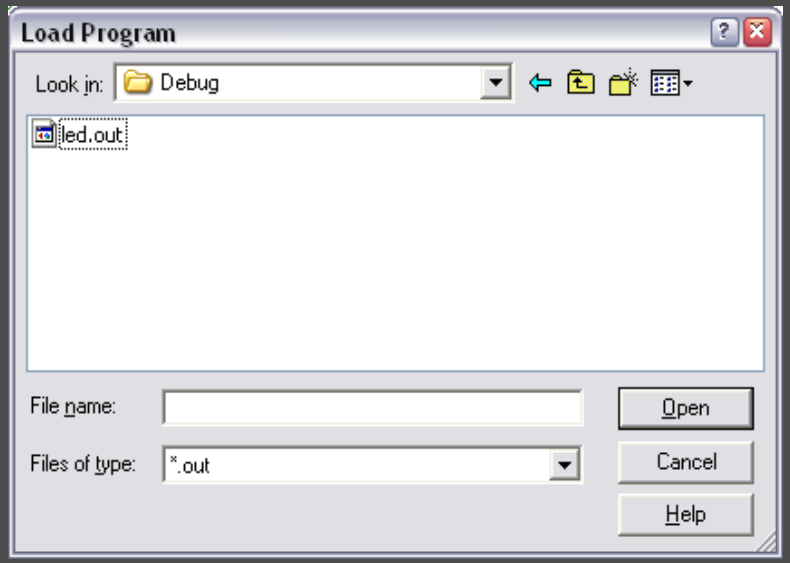




Loading and Running a Project on the C6713 DSK

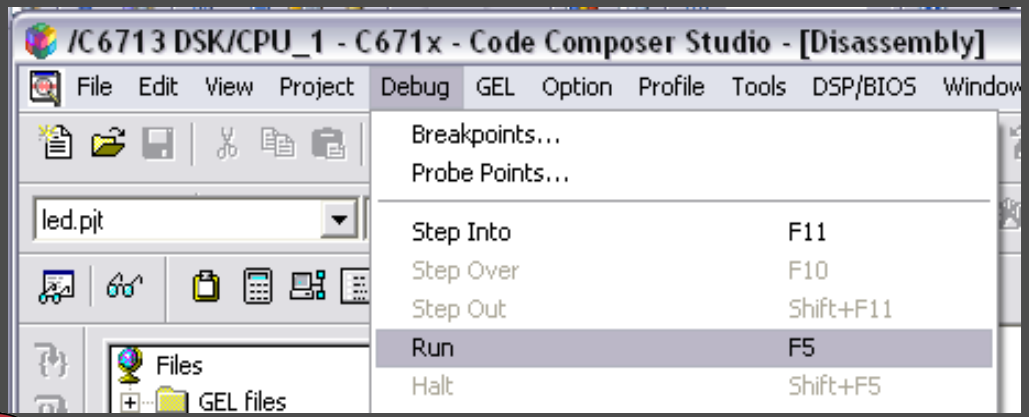
File-> Load Program (ctrl+L)

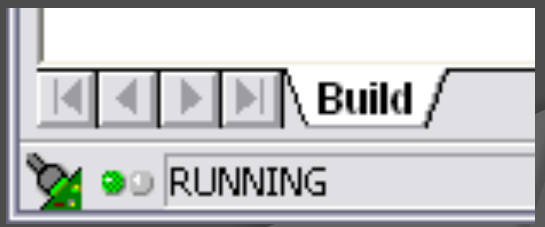




Select the .out file in the project\Debug directory. Program is sent to DSK.

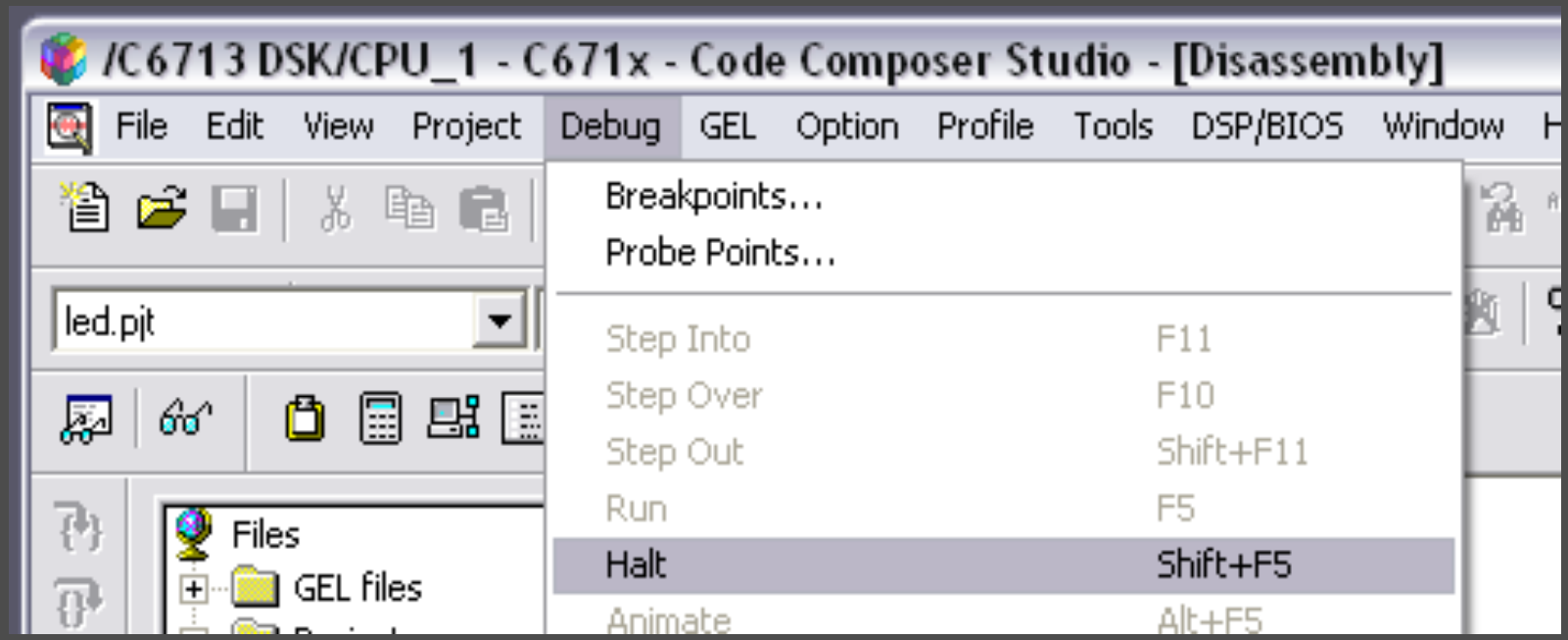
Debug->Run (F5 or the Run button)





Halting a Running Program on the C6713 DSK

Debug->Halt (shift+F5 or the Halt button ).



Tip: Fixing the search path

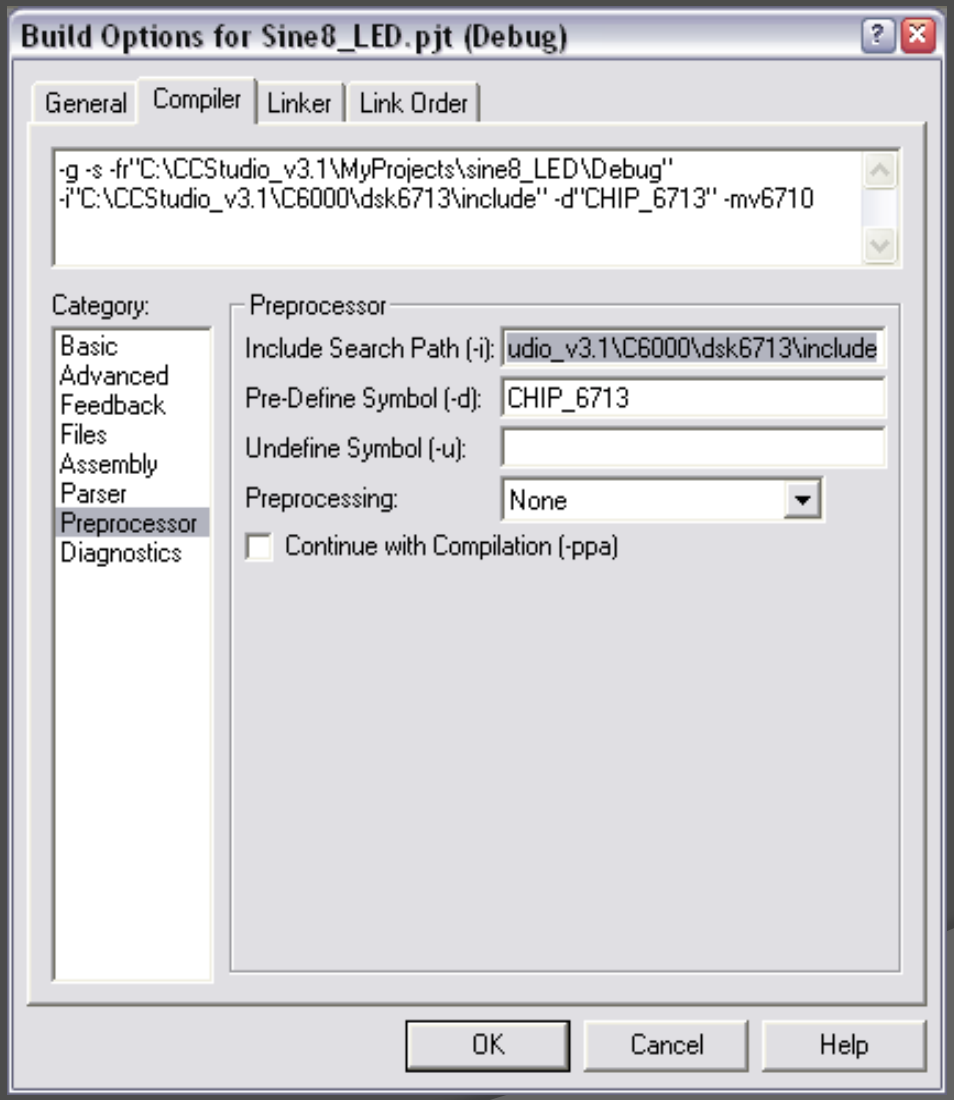
Add C:\CCStudio_v3.1\C6000\dsk6713\include to the search path

Project ->

Build Options ->

[Compiler tab] ->

[Preprocessor category]



Tip: Problems Finding Files During Linking

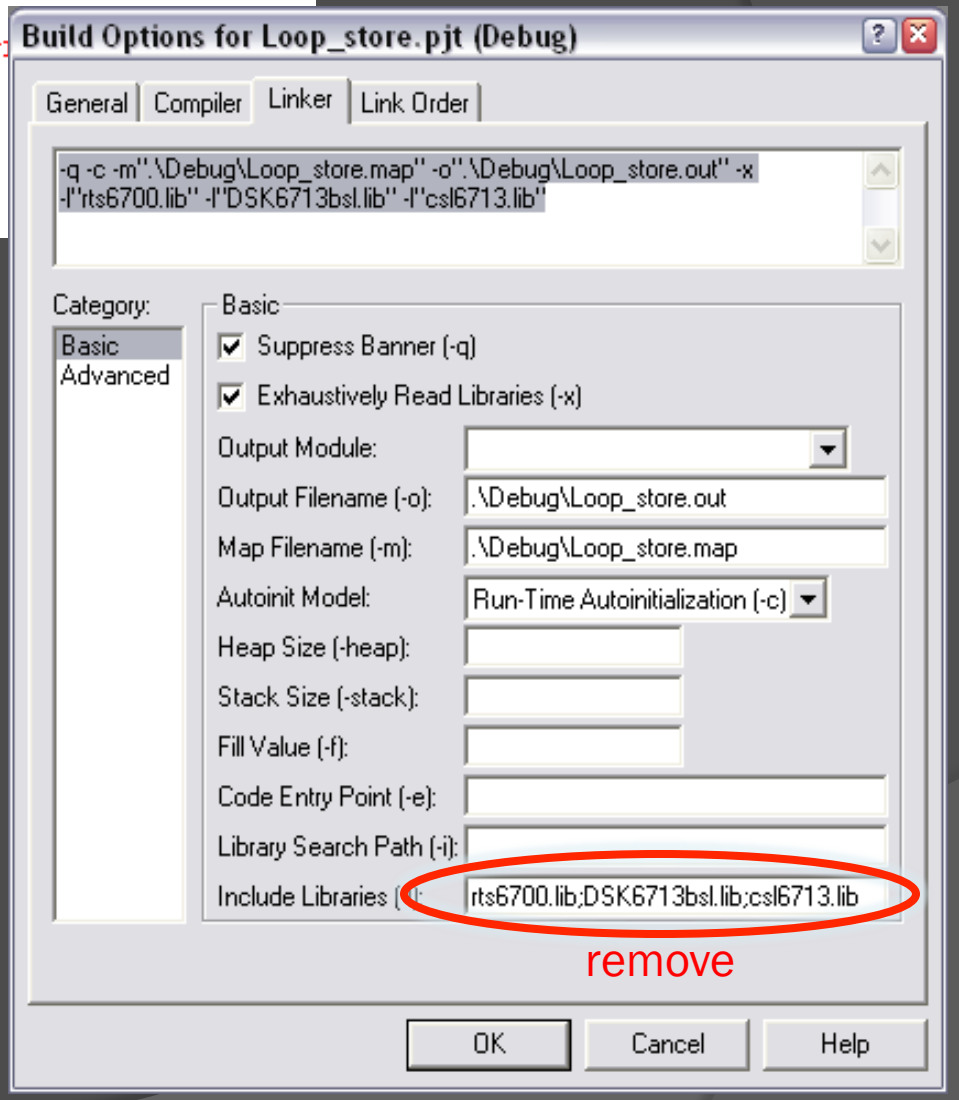
```
[Loop_store.c] "C:\CCStudio_v3.1\C6000\cgtools\bin\cl6x" -g -q -fr"C:/

[Linking...] "C:\CCStudio_v3.1\C6000\cgtools\bin\cl6x" -@"Debug.lkf"
<Linking>
>> C:\DOCUME~1\drb\LOCALS~1\Temp\TI5643, line 21:   er
                can't find input file 'DSK6713bsl.lib'

>> Compilation failure

Build Complete.
```

Problem is caused by a bad path for the include libraries in the linker options (Project -> Build Options -> Linker tab)

A fix for this is to remove rts6700.lib, DSK6713bsl.lib, and csl6713.lib from the linker options and add these files manually (Project -> Add files to Project…)

C:\CCStudio_v3.1\c6000\cgtools\lib\rts6700.lib
C:\CCStudio_v3.1\c6000\csl\lib\csl6713.lib
C:\CCStudio_v3.1\c6000\dsk6713\lib\dsk6713bsl.lib

Or you can add the appropriate directories to the library search path.



Tip: Fixing the memory model

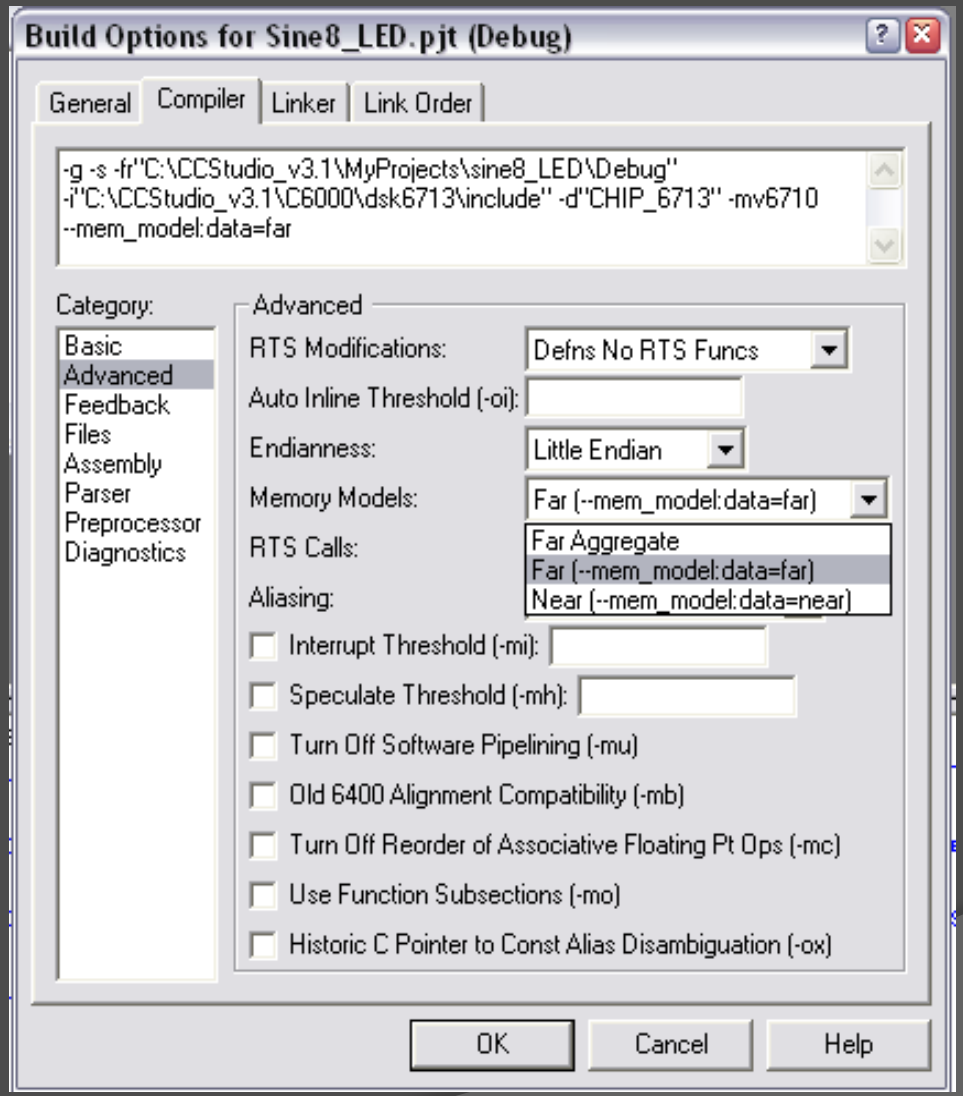
Change the memory model to "data=far"

Project ->

Build Options ->

[Compiler tab] ->

[Advanced category]

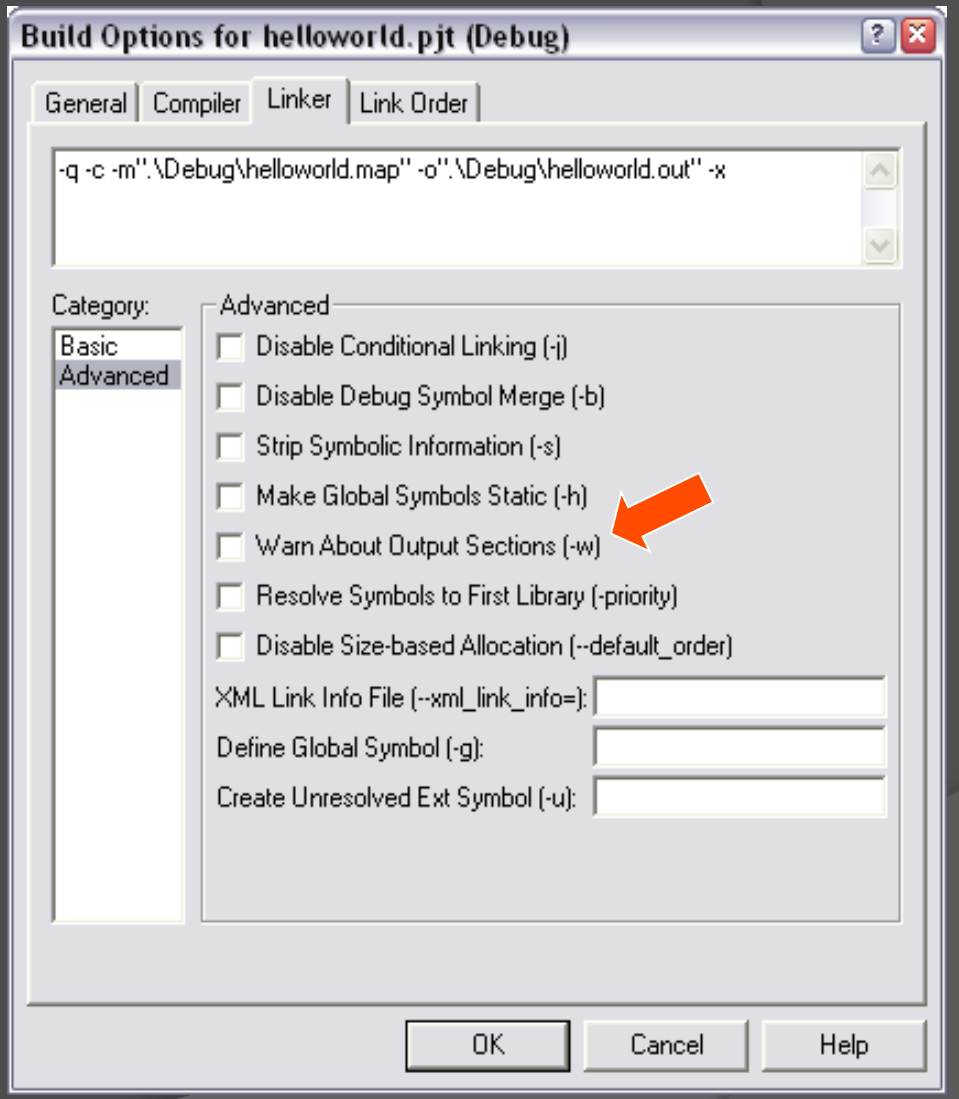


Optional: Suppress Linker Warnings

Project->Build Options [linker tab]

In the Advanced category, uncheck "warn about output sections".

Alternatively, put values for stack and heap in the Basic category.

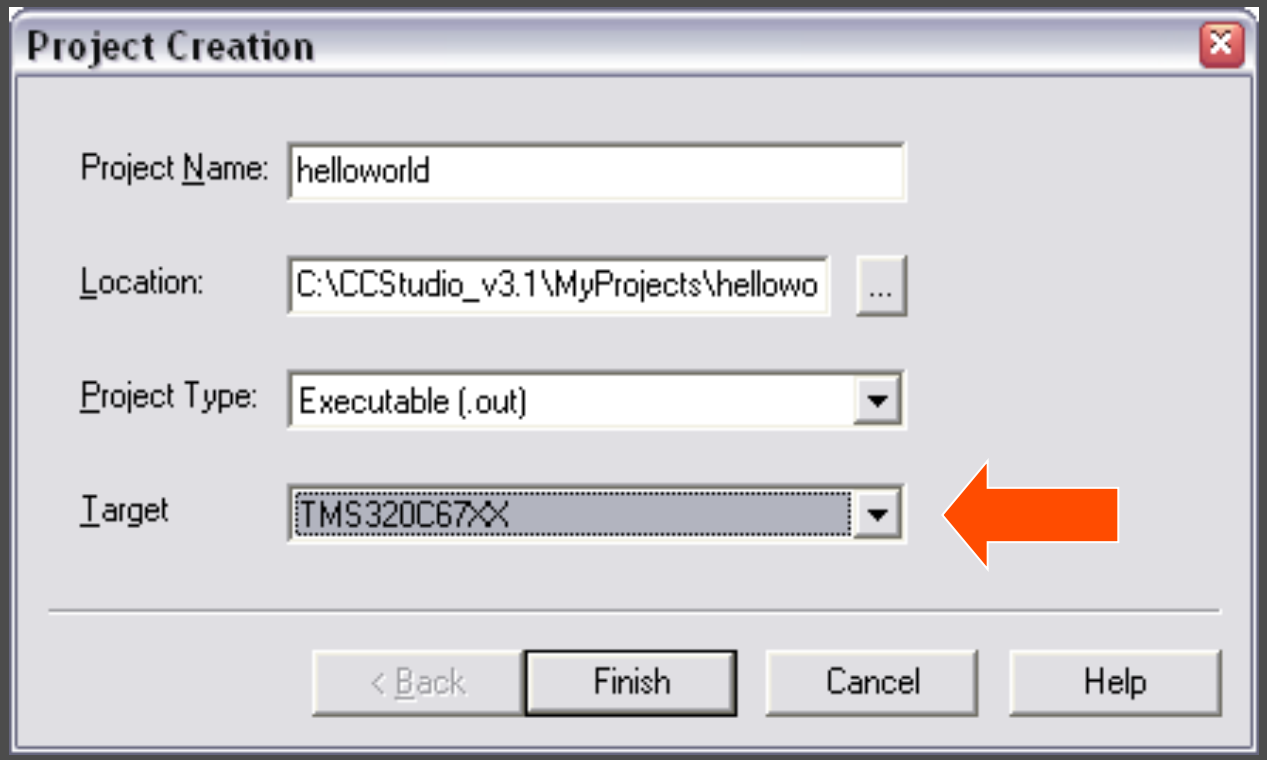


Things to Try

- Open the Sin8_LED project and fix the search path and the memory model (see previous pages). Then build, load, and run it.
  - Press DIP switch 0. You should see LED 0 light up and a 1kHz sinusoid should appear on the left channel of the codec. This is a good test to see if the DSK is working.
- Make an error in the source code Sin8_LED.c and build the project to see what happens.
- Change the amplitude of the sinusoid (gain variable), rebuild, reload, and see what happens.
- Modify the code to generate a 500Hz sinusoid.
- Open, build, and load other projects in "myprojects"

Creating a New Project (1 of 5)

1. Create new project
   Project->New



Creating a New Project (2 of 5)

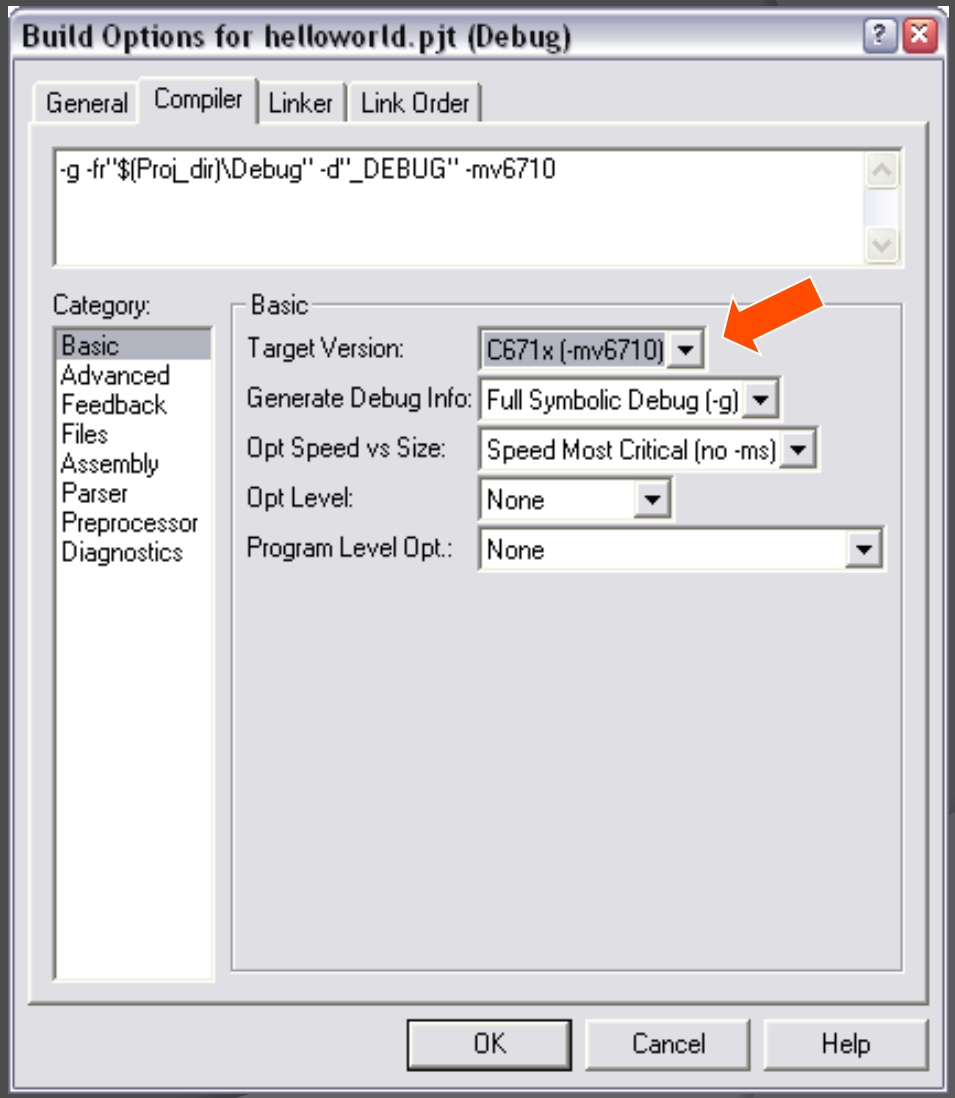
2. Write your C code:
   File->New->Source File
3. Save it in your project directory (make sure it has a .c extension):
   File->Save
4. Add your C code to the project:
   Project->Add Files to Project

Creating a New Project (3 of 5)

5. Add required support files to project
   Project->Add Files to Project
   a) myprojects\support\c6713dsk.cmd
      [linker command file – this or another cmd file is required]
   b) c6000\cgtools\lib\rts6700.lib
      [run-time support library functions - required]
6. Add optional support files to project, e.g.
   Project->Add Files to Project
   a) myprojects\support\vectors_poll.asm or vectors_intr.asm
      [used to set up interrupt vectors]
   b) c6000\dsk6713\lib\dsk6713bsl.lib
      [DSK board support library functions – useful for interfacing to the codec, DIP switches, and LEDs]
   c) c6000\csl\lib\csl6713.lib
      [chip support library functions]

Creating a New Project (4 of 5)

7. Set up the build options for C6713:
   Project -> Build Options
   (compiler tab)
   - Make sure target version is C671x
   - Also make sure Opt(imization) Level is "none" - this will help with debugging



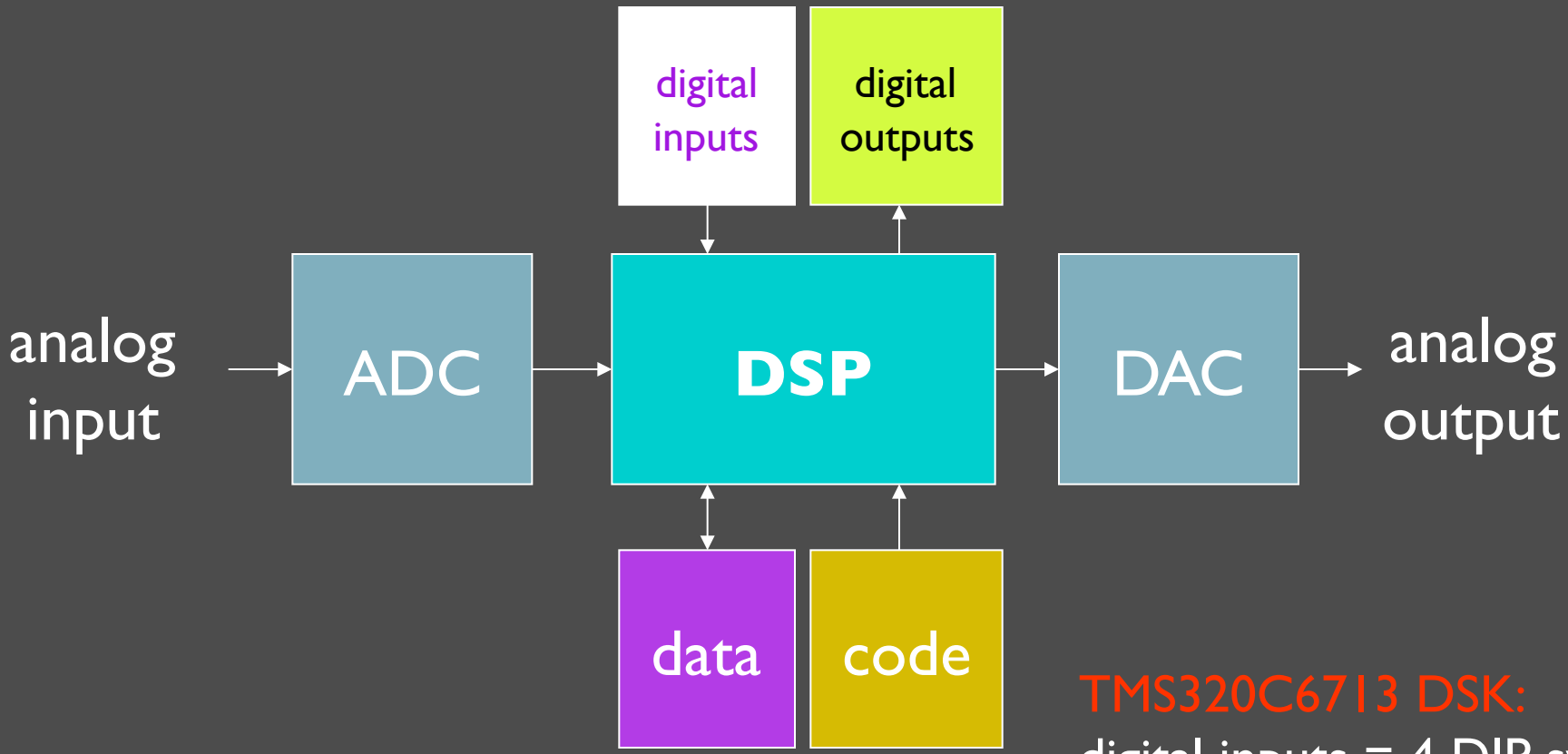
Creating a New Project (5 of 5)

8. Scan all file dependencies to automatically bring all header files and includes into the project:
   Project -> Scan all file dependencies
9. Build the project:
   Project -> Build
10. If successful, load the .out file to the DSK:
    File -> Load Program
    Select the Debug directory. Select the .out file.
11. Run it:
    Debug -> Run or F5 or the run button.

A Simple Program to Try: "helloworld"

```
// helloworld.c
// D. Richard Brown III
// 19-Oct-2009

#include <stdio.h>

void main()
{

        printf("Hello world.\n");

}
```

More Interesting Programs: Interfacing With the Real World



TMS320C6713 DSK:
digital inputs = 4 DIP switches
digital outputs = 4 LEDs
ADC and DAC = AIC23 codec

Interfacing With the DIP Switches and LEDs

LED and DIP switch interface functions are provided in dsk6713bsl.lib.

- Initialize DIP/LEDs with
  DSK6713_DIP_init() and/or DSK6713_LED_init()
- Read state of DIP switches with
  DSK6713_DIP_get(n)
- Change state of LEDs with
  DSK6713_LED_on(n) or
  DSK6713_LED_off(n) or
  DSK6713_LED_toggle(n)

where n=0, 1, 2, or 3.

Documentation is available in C:\CCStudio_v3.1\docs\hlp\c6713dsk.hlp

Interfacing With the AIC23 codec: C6x Interrupt Basics

- Interrupt sources must be mapped to interrupt events
  - 16 "interrupt sources" (timers, serial ports, …)
  - 12 "interrupt events" (INT4 to INT15)
- Interrupt events have associated "interrupt vectors". An "interrupt vector" is a special pointer to the start of the "interrupt service routine" (ISR).
- Interrupt vectors must be set up in your code (usually in the file "vectors.asm").
- You are also responsible for writing the ISR.

Setting Up an Interface With the AIC23 Codec (step 1 of 3)

We can write the ISR first:

```
49  interrupt void serialPortRcvISR()
50  {
51      Uint32 temp;
52
53      temp = MCBSP_read(DSK6713_AIC23_DATAHANDLE);   // read L+R channels
54      MCBSP_write(DSK6713_AIC23_DATAHANDLE,temp);    // write L+R channels
55  }
```

Remarks:

- MCBSP_read() requests samples from the codec's ADC
- MCBSP_write() sends samples to the codec's DAC
- This ISR simply reads in samples and then sends them back out.

Codec Data Format and How To Separate the Left/Right Channels

```
// we can use the union construct in C to have
// the same memory referenced by two different variables
union {Uint32 combo; short channel[2];} temp;
```


```
// the McBSP functions require that we
// read/write data to/from the Uint32 variable
temp.combo = MCBSP_read(DSK6713_AIC23_DATAHANDLE);
MCBSP_write(DSK6713_AIC23_DATAHANDLE, temp.combo);

// but if we want to access the left/right channels individually
// we can do this through the short variables
Leftchannel = temp.channel[1];
Rightchannel = temp.channel[0];
```

Setting Up an Interface With the AIC23 Codec (step 2 of 3)

- Now we can set up the interrupt vector to point to the ISR.
- In this example, our ISR is called "serialPortRcvISR".
- We will link the codec interrupt event to INT15.
- Here is the appropriate code in the vectors.asm file:

```
150 INT15:
151     MVKL .S2 _serialPortRcvISR, B0
152     MVKH .S2 _serialPortRcvISR, B0
153     B    .S2 B0
154     NOP
155     NOP
156     NOP
157     NOP
158     NOP
```

Setting Up an Interface With the AIC23 Codec (step 3 of 3)

Initialization steps:

1. Initialize the DSK
2. Open the codec with the default configuration.
3. Configure multi-channel buffered serial port (McBSP)
4. Configure codec parameters, e.g. set the sampling rate
5. Configure and enable interrupts
6. Do normal processing (we just enter a loop here)

```
21 interrupt void serialPortRcvISR(void);                    // ISR function prototype
22
23 void main()
24 {
25     DSK6713_init();     // Initialize the board support library, must be called first
26     hCodec = DSK6713_AIC23_openCodec(0, &config);          // Open the codec
27
28     // Configure buffered serial ports for 32 bit operation
29     // This allows transfer of both right and left channels in one read/write
30     MCBSP_FSETS(SPCR1, RINTM, FRM);
31     MCBSP_FSETS(SPCR1, XINTM, FRM);
32     MCBSP_FSETS(RCR1, RWDLEN1, 32BIT);
33     MCBSP_FSETS(XCR1, XWDLEN1, 32BIT);
34
35     DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_48KHZ);    // set the sampling rate
36
37     // Interrupt setup
38     IRQ_globalDisable();            // Globally disables interrupts
39     IRQ_nmiEnable();                // Enables the NMI interrupt
40     IRQ_map(IRQ_EVT_RINT1,15);      // Maps an event to a physical interrupt
41     IRQ_enable(IRQ_EVT_RINT1);      // Enables the event
42     IRQ_globalEnable();             // Globally enables interrupts
43
44     while(1)
45     {
46     }
47 }
```

Setting the Sampling Rate

Here we open the codec with the default configuration:

```
26      hCodec = DSK6713_AIC23_openCodec(0, &config);          // Open the codec
```

The structure "config" is declared in dsk6713_aic23.h

Rather than editing the header file, we can change the sampling frequency after the initial configuration:

```
35      DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_48KHZ);    // set the sampling rate
```

Frequency definitions are in dsk6713_aic.h

```
/* Frequency Definitions */
#define DSK6713_AIC23_FREQ_8KHZ      1
#define DSK6713_AIC23_FREQ_16KHZ     2
#define DSK6713_AIC23_FREQ_24KHZ     3
#define DSK6713_AIC23_FREQ_32KHZ     4
#define DSK6713_AIC23_FREQ_44KHZ     5
#define DSK6713_AIC23_FREQ_48KHZ     6
#define DSK6713_AIC23_FREQ_96KHZ     7
```

Other Codec Configuration

- Line-level input volume (individually controllable for left and right channels)
- Headphone output volume (individually controllable for left and right channels)
- Digital word size (16, 20, 24, or 32 bit)
- Other settings, e.g. byte order, etc. For more details, see:
  - dsk6713_aic23.h
  - Codec datasheet (TLV320AIC23B)
  - C:\CCStudio_v3.1\docs\hlp\c6713dsk.hlp

Some Things to Try

- Make a new project that:
  - Polls DIP switch 0. If pressed, light up all four LEDs.
  - Sets the sampling rate of the AIC23 codec to 44.1kHz.
  - Uses an ISR to sample the left and right channels.
  - Multiplies the left and right channels by a variable gain.
  - Outputs the modified samples to the left and right channels.
- Bonus: Swap the channels, i.e. Left_in -> Right_out, Right_in -> Left_out, when DIP switch 0 is pressed.
- Bonus: Try changing the input/output volumes (hint: look at default configuration in dsk6713_aic23.h)

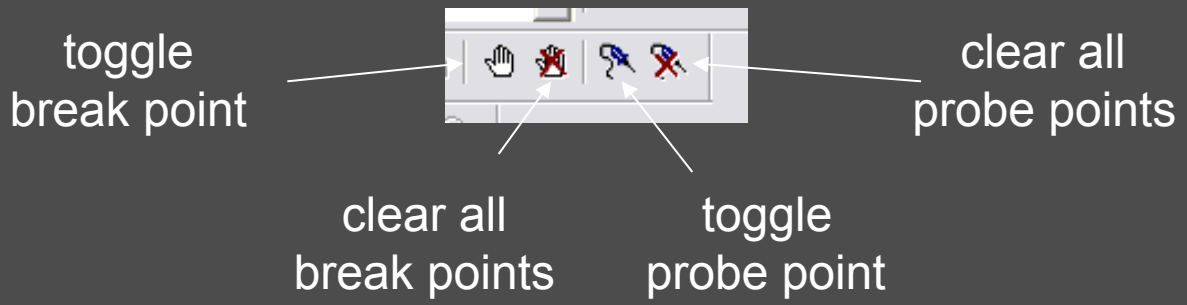
Lunch Break

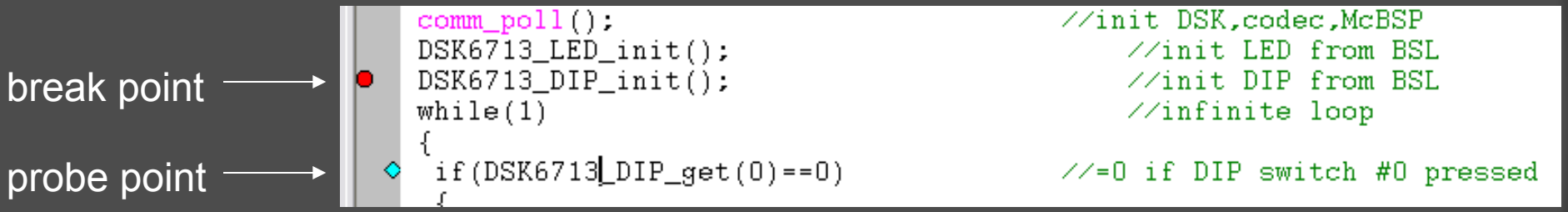
Workshop resumes at 1:30pm…

Debugging and Other Useful Features of the CCS IDE

- Breakpoints
- Probe points
- Watch variables
- Plotting arrays of data
- Animation
- General Extension Language (GEL)

Breakpoints and Probe Points



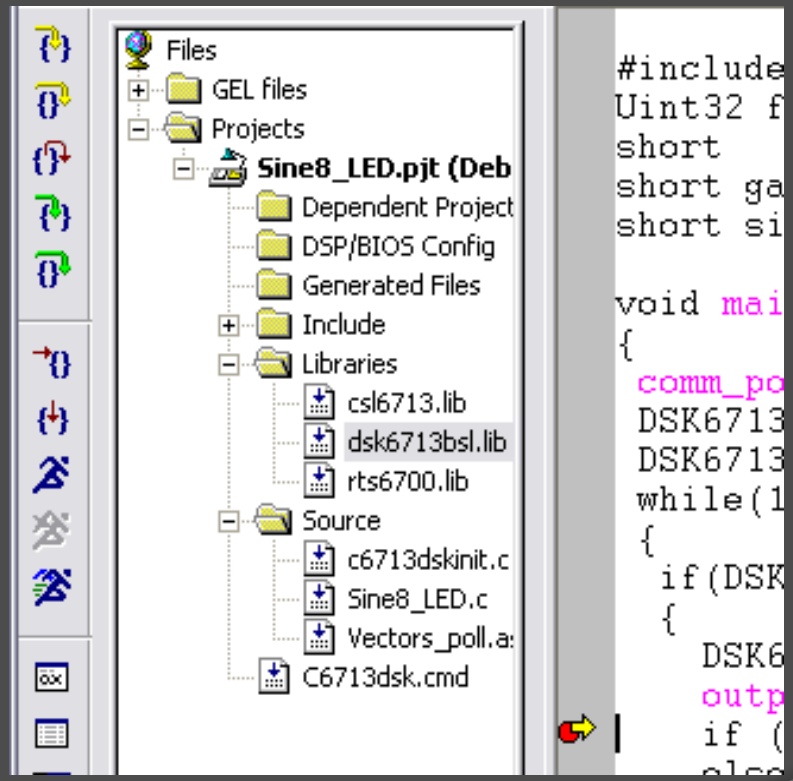


- **Breakpoints:** stop code execution at this point to allow state examination and step-by-step execution.
- **Probe points:** force window updates and/or read/write samples from/to a file at a specific point in your code.

Breakpoints

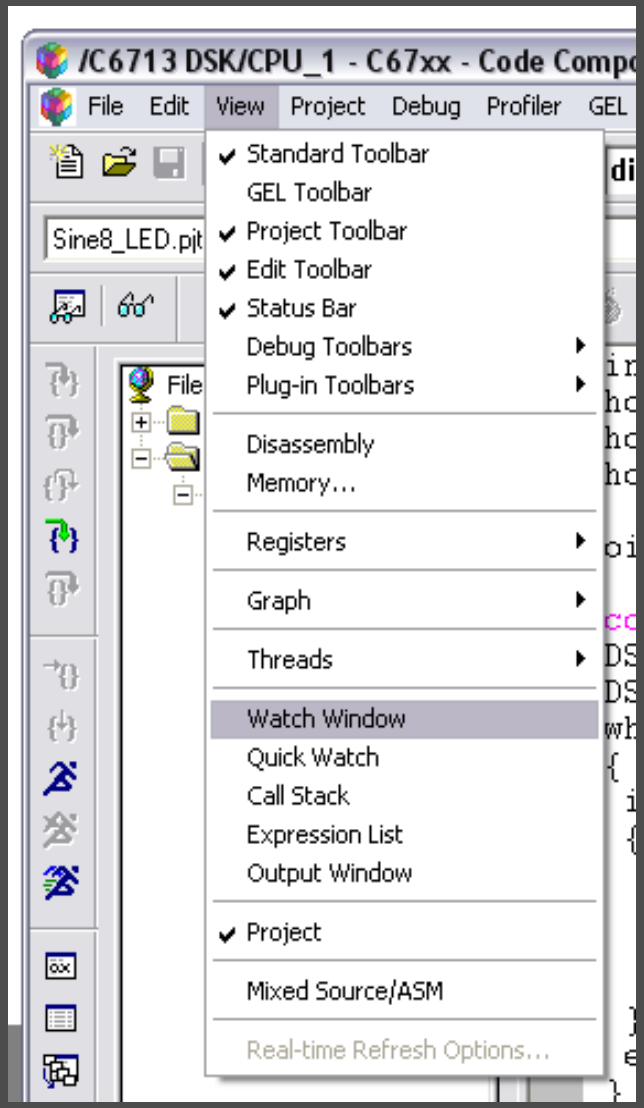
source step into
source step over
step out
ASM step into
ASM step over

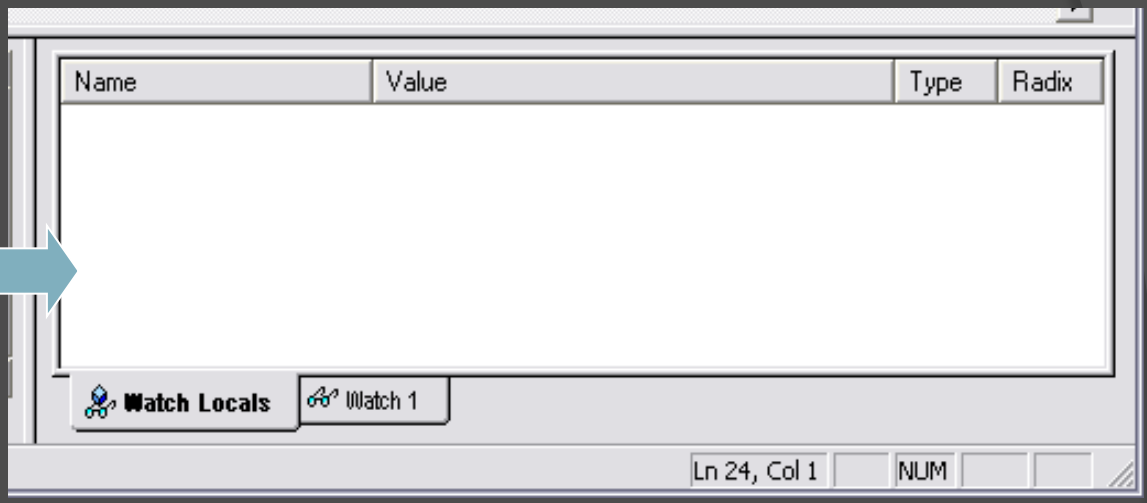
run to cursor
set progam counter to cursor

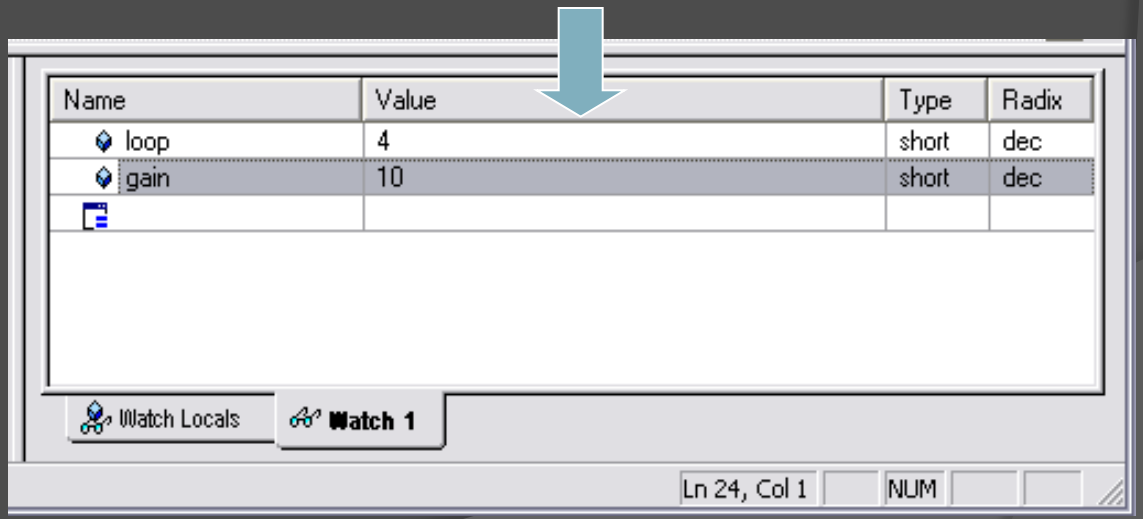


"Run to Cursor" is a handy shortcut instead of setting a breakpoint

Watch Variables



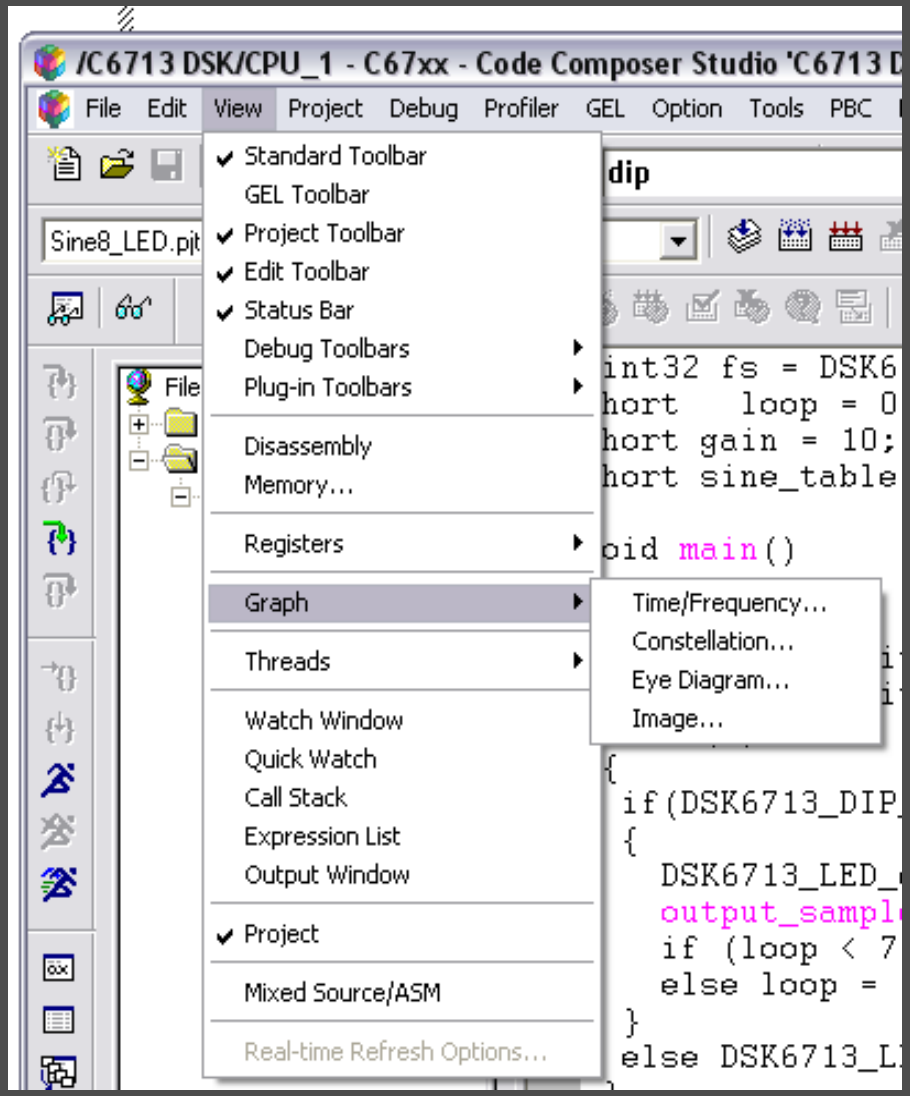


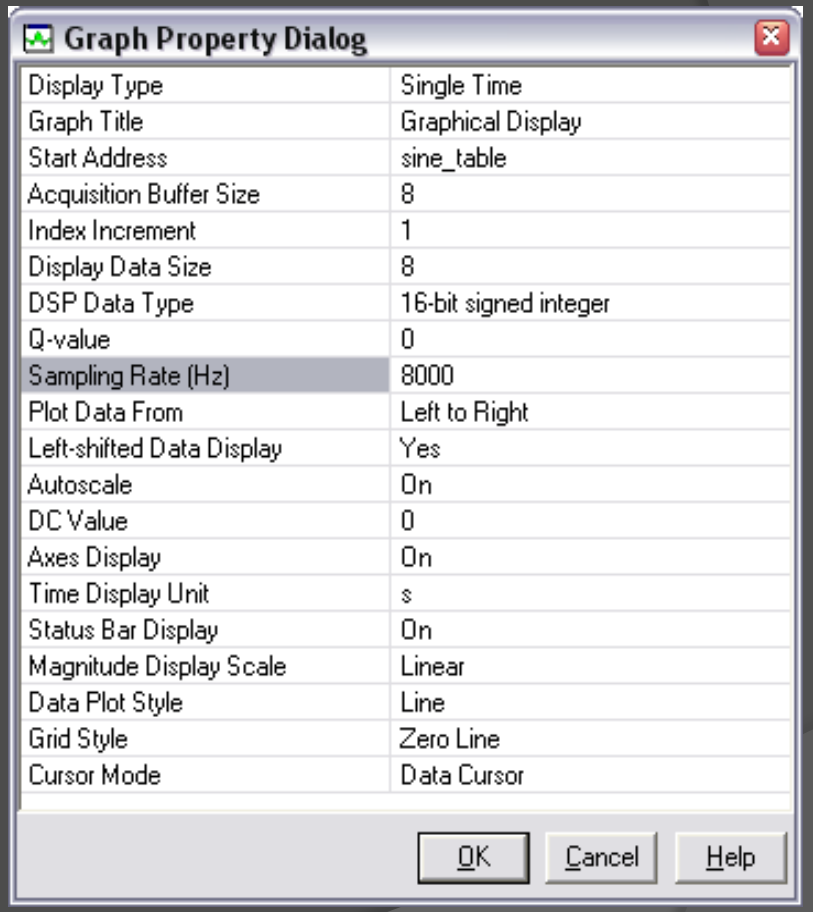


Watch Variables

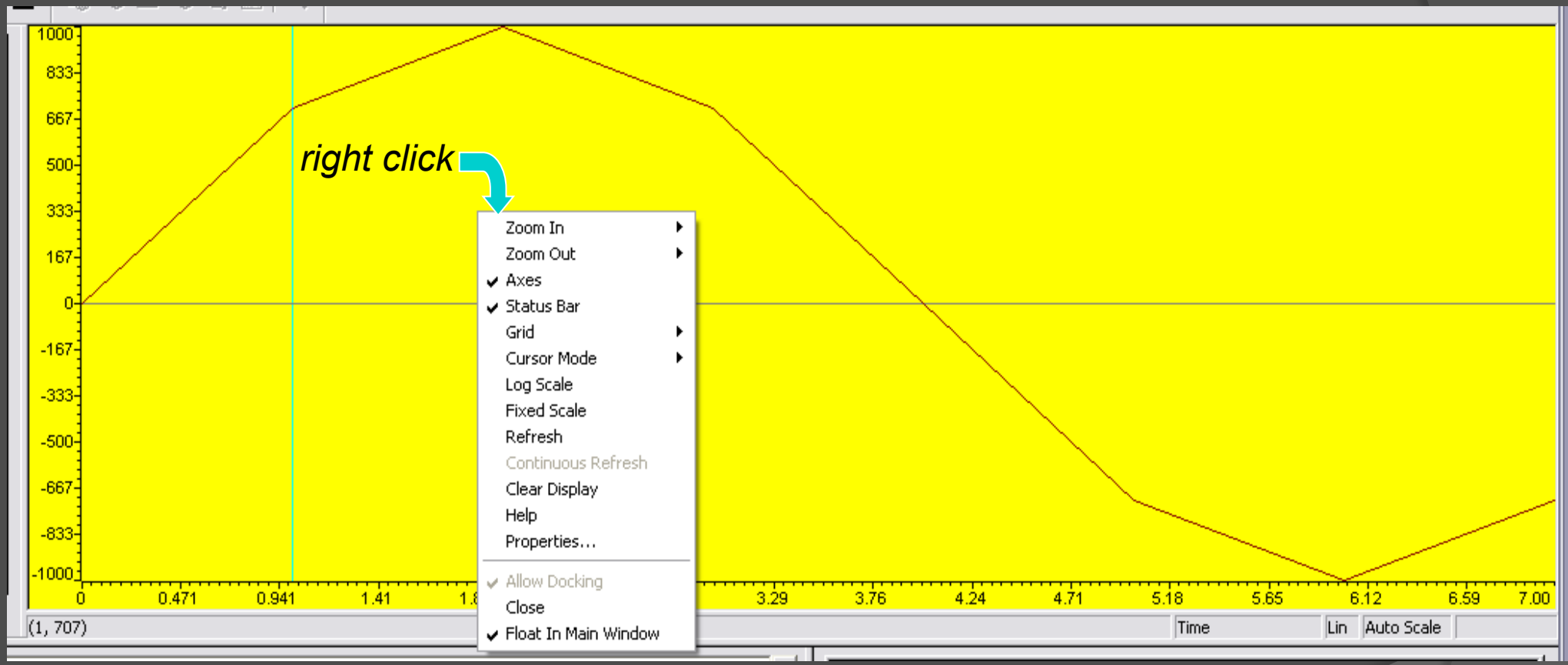
- In the Watch Locals tab, the debugger *automatically* displays the Name, Value, and Type of the variables that are *local* to the currently executing function.
- In the Watch tab, the debugger displays the Name, Value, and Type of the local and global variables and expressions that *you specify*.
- Can add/delete tabs.

Plotting Arrays of Data





Graph Windows: Plotting Arrays of Data



Probe Points

- Differ from breakpoints: Halt the DSP momentarily, perform an action, and then automatically resume execution.
- Some useful functions of probe points:
  - Connect probe point to graph window.
  - Connect probe point to file I/O to facilitate repeatable testing.
  - Note that probe points cause problems with real-time operation.
- For more details, see CCS Getting Started Guide (SPRU509F.PDF) or CCS help.

Using a Probe Point for File I/O

- Basic idea:
  - Create an input data file with the signals you want to use to test your code.
  - Place a probe point in your code to read the contents of datafile into an array in the DSK memory.
  - Place another probe point in your code to write the results (stored in an array on the DSK) to a datafile on the computer.

Example Matlab code to generate input data (noisy sinusoid)

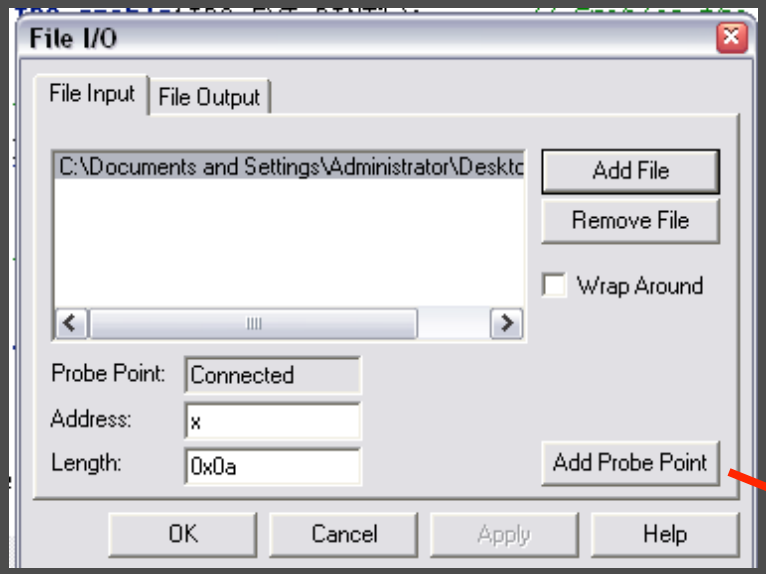
```
fs = 44100;
t = 0:1/fs:1;
f = 1000;
x = 0.5*sin(2*pi*f*t) + 0.1*randn(1,length(t));
fid = fopen('inputdata.dat','wt');
fprintf(fid,'1651 4 000 0 801\n');
fprintf(fid,'%f\n',x);
fclose(fid);
```

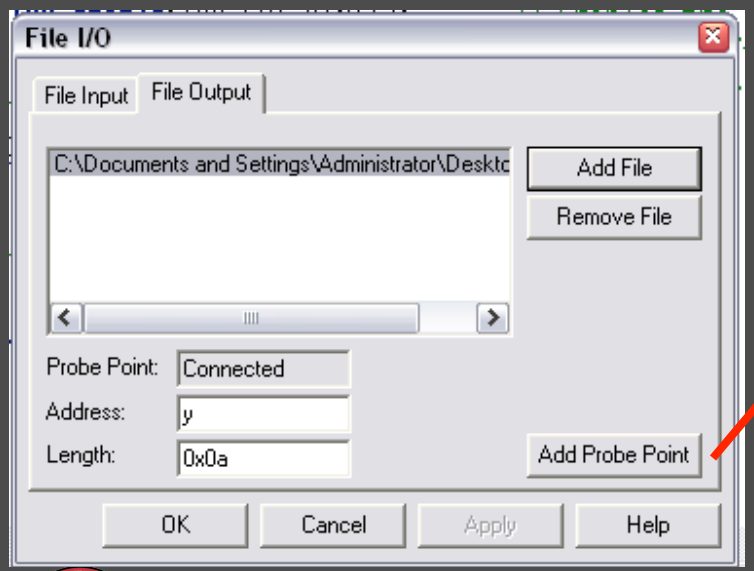
Setting Probe Points in CCS

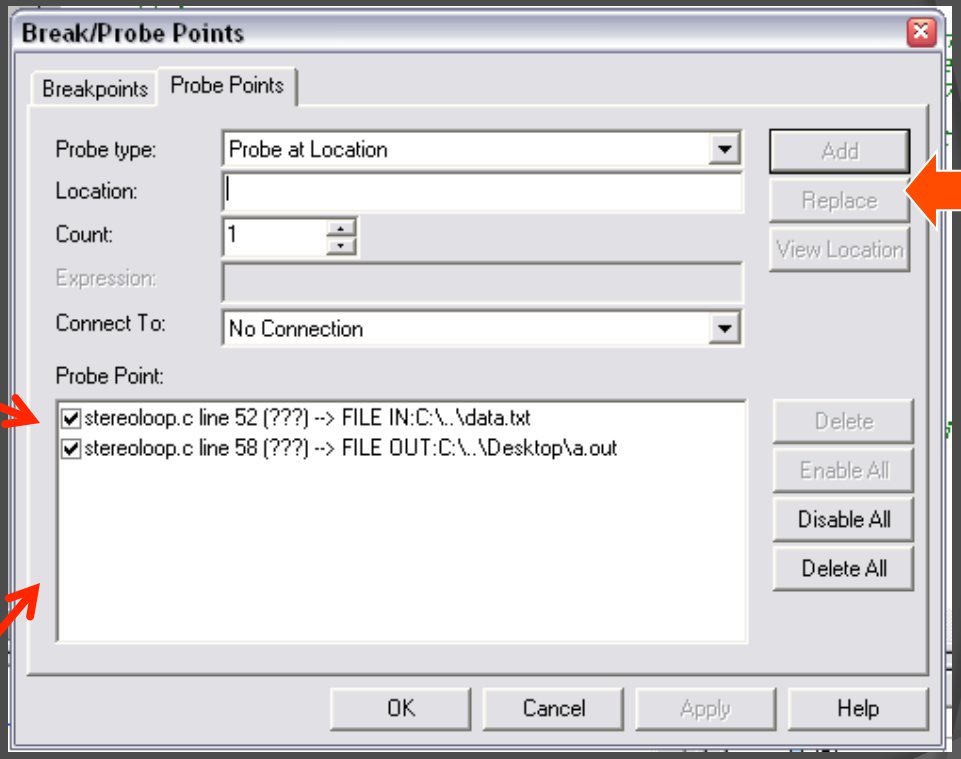
```
// use file i/o to fill up x array with data from file
printf("Reading datafile...\n");
y[0] = x[0];
for (i=1;i<10;i++)
    y[i] = y[i-1]+x[i];

// use file i/o to write y array to data file
printf("Writing datafile...\n");
```

Connecting Probe Points to File I/O

- File -> File I/O



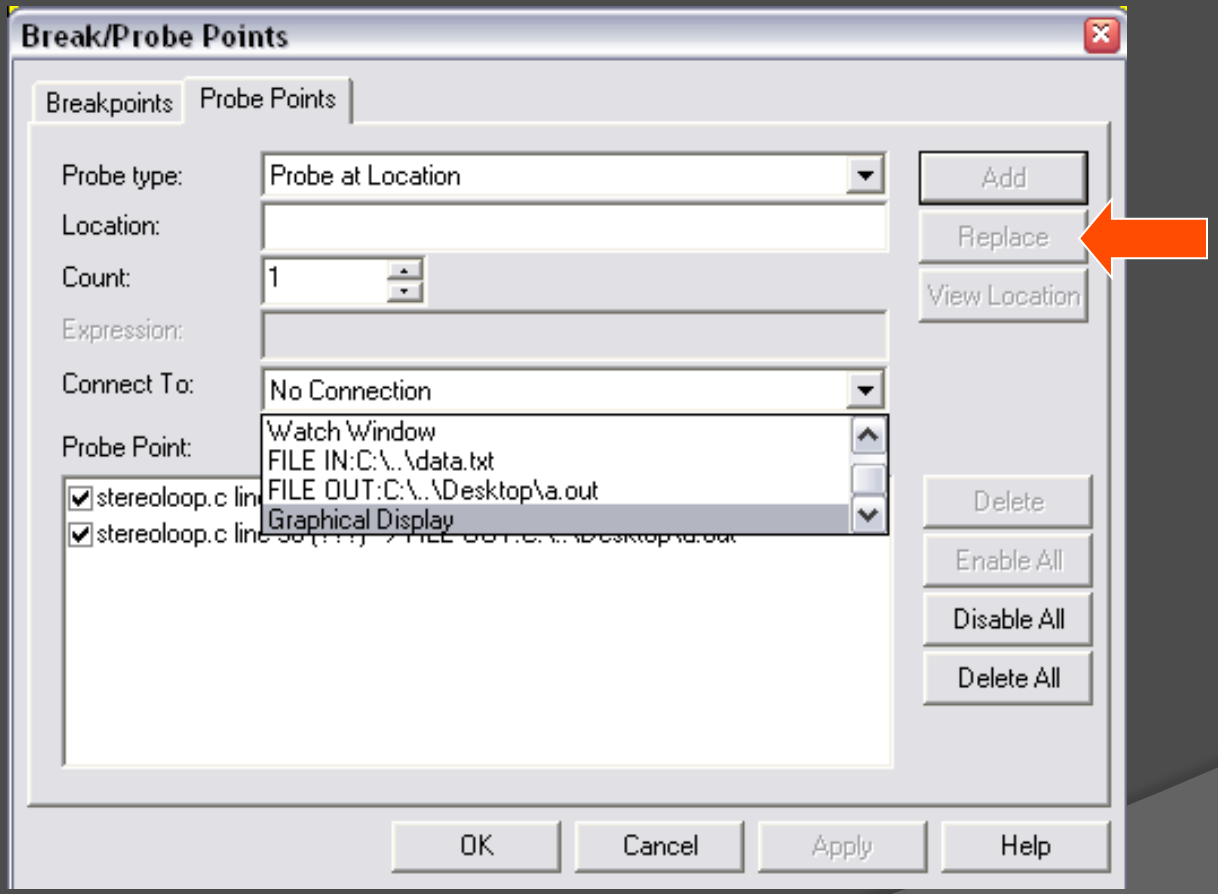




See SPRU509 for more details.

Using a Probe Point to Update a Graph Window

- First create a graph using View->Graph
- Then go to Debug->Probe Points to connect a probe point to the graph

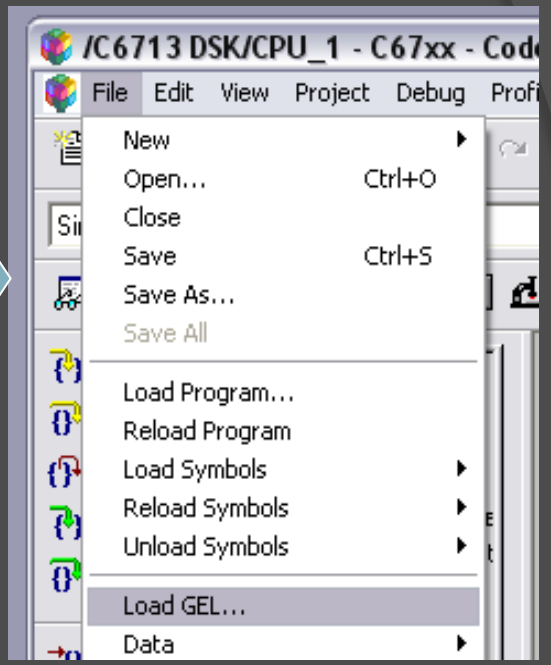


Animation 

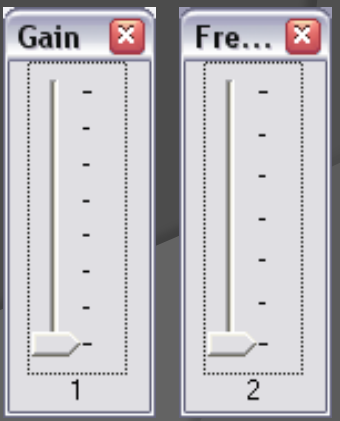
- Runs the program until a breakpoint is encountered
  - At the breakpoint, execution stops and all windows not connected to any Probe Points are updated.
  - Program execution then automatically resumes
  - Useful for updating graphical displays
- Note: Animation will cause problems with real-time operation
- Can pause execution at each breakpoint:
  Option->Customize: Debug Properties tab
  Animate Speed (0-9s) (zero = no pause)

General Extension Language

- Create functions to extend the functionality of Code Composer Studio
- GEL files are not loaded with a project



- Often used to change variables "on-the-fly"
- Examples from Chassaing textbook: sin2sliders.pjt and sin2sliders.gel



General Extension Language

- Useful GEL files can be pretty simple
- From sin2sliders.gel:

```
/*Sine2sliders.gel Two sliders to vary gain and frequency*/

menuitem "Sine Parameters"

slider Gain(1,8,1,1,gain_parameter)          /*incr by 1,up to 8*/
    {
        gain = gain_parameter;               /*vary gain*/
    }

slider Frequency(2,8,2,2,frequency_parameter) /*incr by 2,up to 8*/
    {
        frequency = frequency_parameter;      /*vary frequency*/
    }
```

- Syntax details can be found in CCS help:
  Help->Contents->Making a Code Composer Studio Project -> Building and Running your Project -> Automating Tasks with General Extension Language (GEL)

Some Things to Try

- Try out the debugging tools on the code you wrote in the morning session
  - breakpoints
  - watch variables
  - step into, step over, step out
- Try out the CCS plotting tools
  - Modify your code to have a buffer (i.e., store samples in an array) and plot the contents in a graph window
- Try out file I/O with probe points and/or updating a graph with probe points
- Try to have CCS animate a plot window via probe points and/or animation
- Modify your stereo in/out project to have the output gain changeable via a GEL slider

Finite Impulse Response (FIR) Filters

- Frequently used in real-time DSP systems
  - Simple to implement
  - Guaranteed to be stable
  - Can have nice properties, e.g. linear phase
- Input/output relationship

$$y[n] = \sum_{m=0}^{M-1} h[m]x[n-m]$$

x = input, y = output, h = filter coefficients, M = # of filter coefficients

Creating FIR Filters

1. Design filter
   - Type: low pass, high pass, band pass, band stop, ...
   - Filter order M
   - Desired frequency response

Matlab

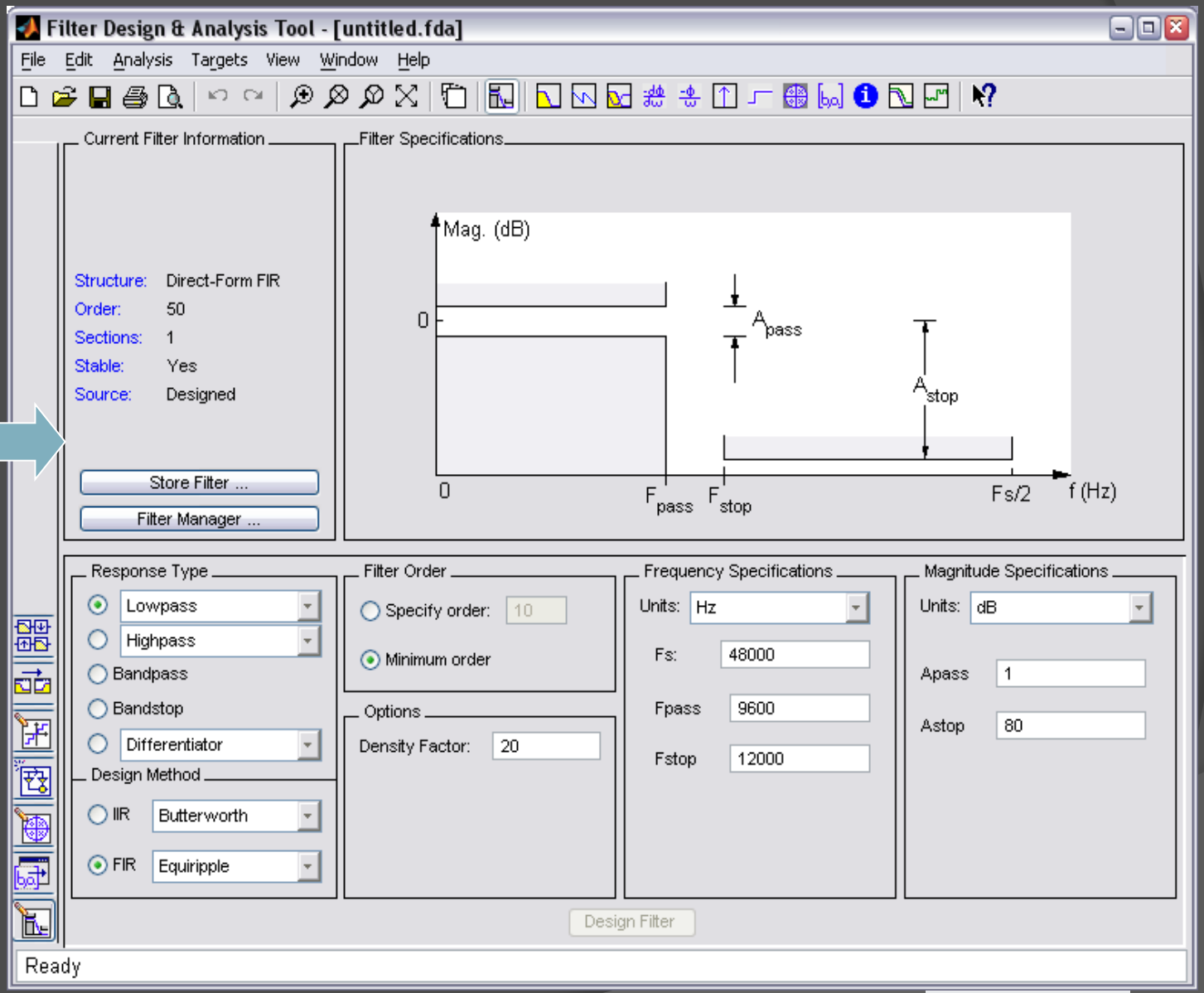
2. Decide on a realization structure
3. Decide how coefficients will be quantized.
4. Compute quantized coefficients
5. Decide how everything else will be quantized (input samples, output samples, result of multiplies, result of additions)
6. Write code to realize filter

CCS

7. Test filter and compare to theoretical expectations

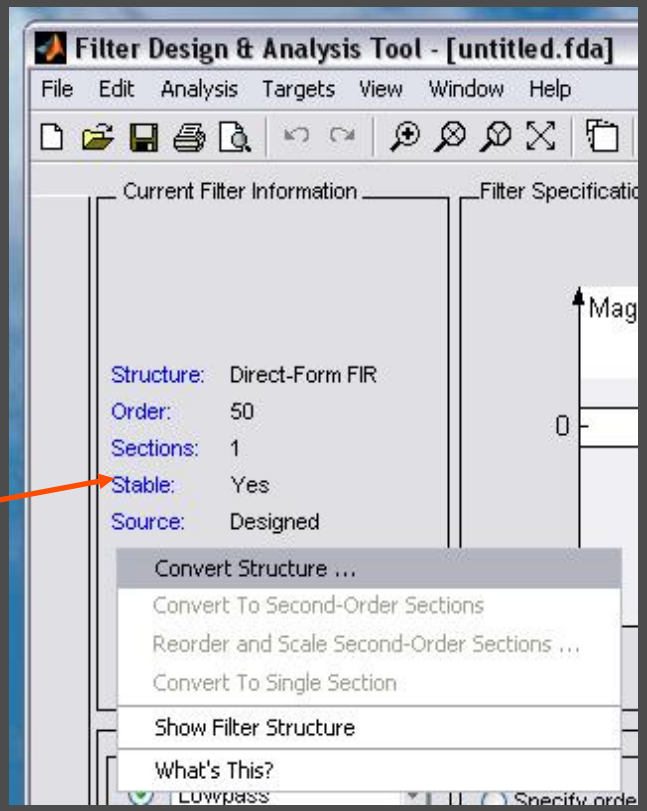
Designing FIR Filters

>> fdatool

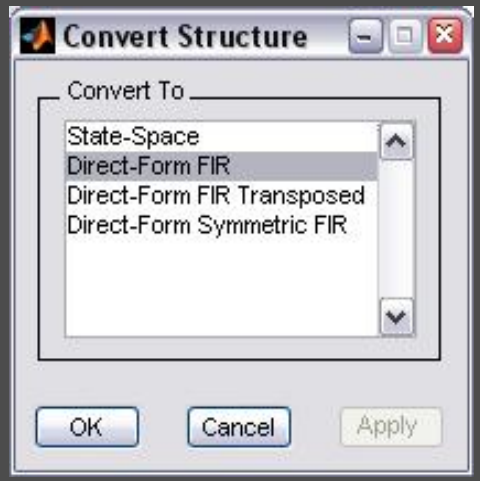


Filter Realization Structures

- Lots of different structures available
  - Direct form I, direct form II, transposed forms, cascade, parallel, lattice, …
  - All have same input/output relationship
  - Choice of structure affects computational complexity and how quantization errors are manifested through the filter

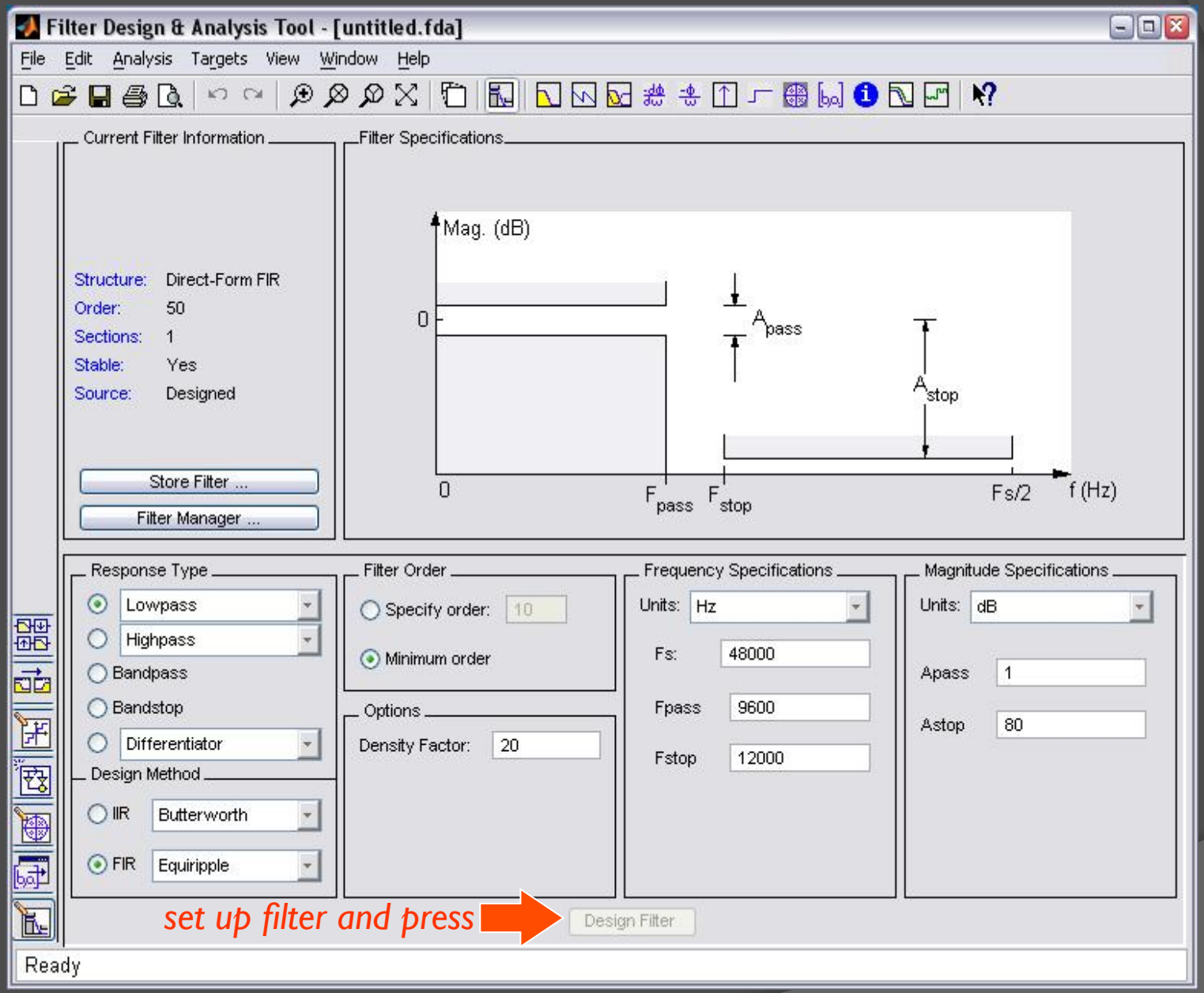


right click in this pane

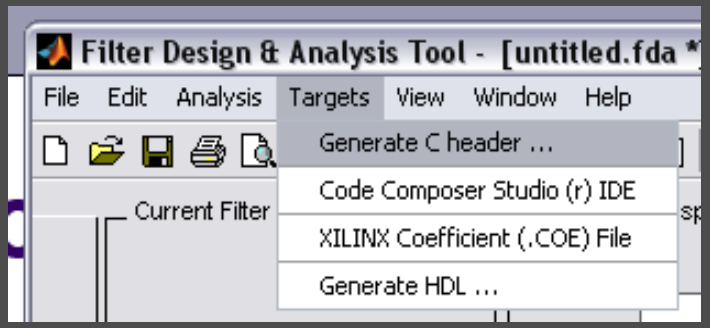


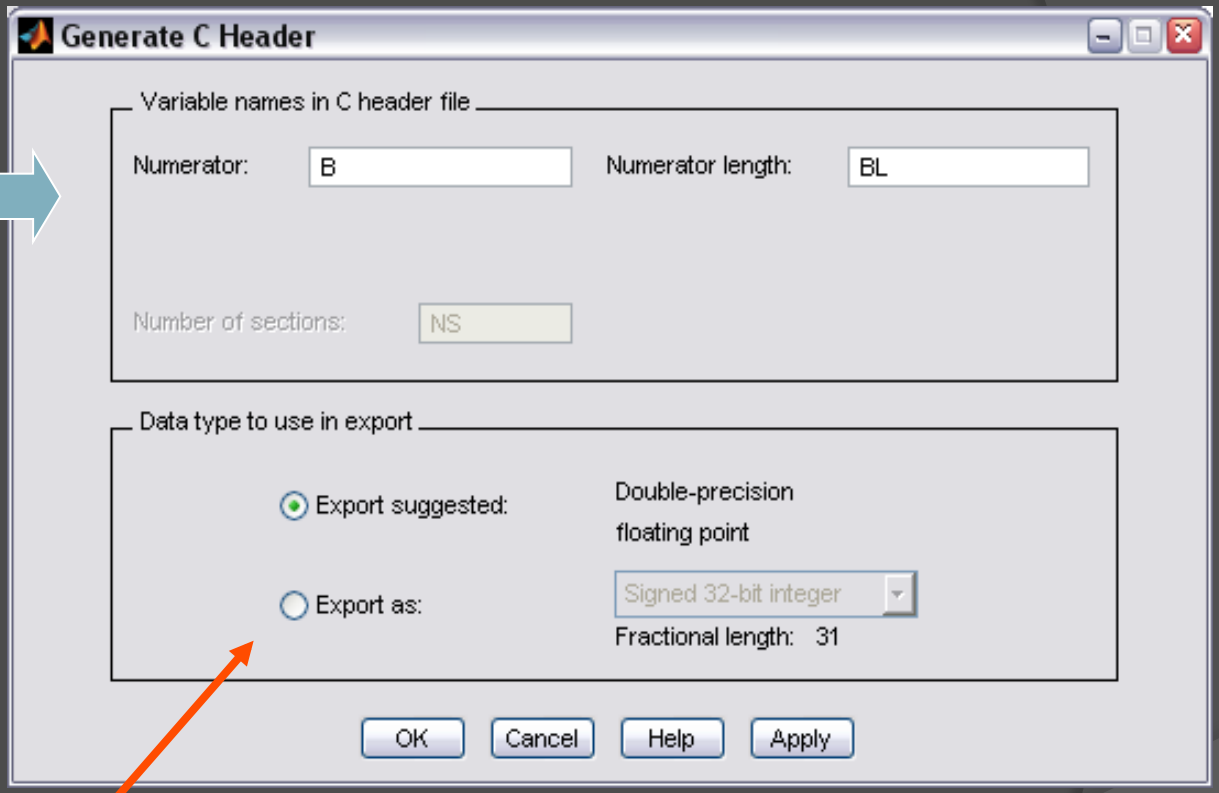
Focus on "Direct form" for now. We'll discuss other options when we look at IIR filtering tomorrow.

Compute FIR Filter Coefficients



Make Coefficient File For CCS





Here you can change the coefficient data type to match your desired quantization.

Example DP-FP Coefficient File

```
/*
 * Filter Coefficients (C Source) generated by the Filter Design and Analysis Tool
 *
 * Generated by MATLAB(R) 7.0 and the
 *
 * Generated on: 19-Aug-2005 13:04:09
 *
 */

/*
 * Discrete-Time FIR Filter (real)
 * -------------------------------
 * Filter Structure  : Direct-Form FIR
 * Filter Order      : 8
 * Stable            : Yes
 * Linear Phase      : Yes (Type 1)
 */

/* General type conversion for MATLAB generated C-code  */
#include "tmwtypes.h"
/*
 * Expected path to tmwtypes.h
 * C:\MATLAB7\extern\include\tmwtypes.h
 */
const int BL = 9;
const real64_T B[9] = {
    0.02588139692752,  0.08678803067191,   0.1518399865268,   0.2017873498839,
     0.2205226777929,   0.2017873498839,   0.1518399865268,  0.08678803067191,
    0.02588139692752
};
```

Include this header file in your project (otherwise you may get unknown datatype errors).

You can edit these variable names to agree with your code.

Quantization Considerations

- Key choice: floating point vs. fixed point
- Advantages of floating point math:
  - Less quantization error
  - Don't have to worry about scaling factors
  - Less likelihood of overflow/underflow
  - Much easier to code
- Disadvantages of floating point math:
  - Requires floating point DSP (higher cost, higher power)
  - Executes slower than fixed point
- C code allows you to "cast" variables into any datatype

Write Code to Realize FIR Filter

- Direct form I implies direct realization of the convolution equation

$$y[n] = \sum_{m=0}^{M-1} h[m]x[n-m]$$

- Some considerations:
  - Allocate buffer of length M for input samples.
  - Move input buffer pointer as new data comes in or move data?

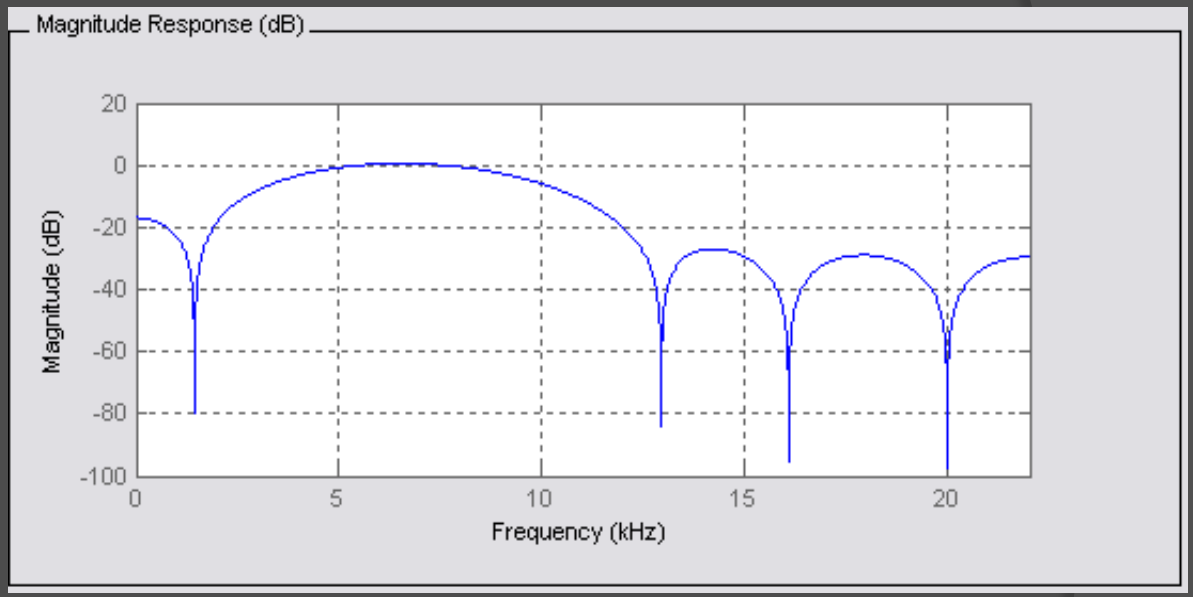
FIR Filter Example Code

```
interrupt void serialPortRcvISR()
{
        union {Uint32 combo; short channel[2];} temp;
        int i = 0;
        float result = 0.0;

        temp.combo = MCBSP_read(DSK6713_AIC23_DATAHANDLE);

        // Update array samples (move data - this is the slow way)
        for( i = N-1; i >= 1; i-- )
                samples[i] = samples[i-1];
        samples[0] = (float)temp.channel[0];        // store right channel

        // Filtering
        for( i = 0 ; i < N ; i++ )
                result += fir_coeff[i]*samples[i];
        temp.channel[0] = (short)result; // output to right channel
        MCBSP_write(DSK6713_AIC23_DATAHANDLE, temp.combo);
}
```

Note that all math here is floating point.
Filter coefficients are also assumed to be floating point.

Some Things to Try

- Try creating an FIR filter with the following specs:
  - Bandpass
  - 8th order Direct Form I
  - Least-squares design
  - 44100Hz sampling rate
  - Fstop1 = 3000Hz
  - Fpass1 = 4000Hz
  - Fpass2 = 8000Hz
  - Fstop2 = 12000Hz
  - Equal weighting in all bands
  - All floating point math (single or double precision)



- Use an oscilloscope and a function generator to compare the magnitude response of your filter to the theoretical prediction.

Workshop Day 1 Summary

What you learned today:

- Basics of the TMS320C6713 DSK and Code Composer Studio
- How to test the DSK
- How to open, build, load, and run existing projects
- How to create, build, load, and run new projects
- How to interface with DSK I/O (LEDs, DIP switches, and the AIC23 codec)
- How to debug code in CCS including
  - Setting and clearing breakpoints and probe points
  - Setting up watch variables
  - Plotting arrays of data
  - Animation
- How to use, modify, and create GEL files in CCS.
- How to use Matlab's filter design/analysis tool "fdatool"
- How to implement an FIR filter on the C6713

Workshop Day 1 Reference Material

- Chassaing and Reay textbook Chapters 1-2, and 4
- CCS Help system
- SPRU509F.PDF CCS v3.1 IDE Getting Started Guide
- C6713DSK.HLP C6713 DSK specific help material
- AIC23 Codec datasheet
- DSK Quick Start Guide (included in your DSK box)
- Spectrum Digital TMS320C6713 DSK reference (included in your DSK box)
- TMS320C6000 Programmer's Guide (SPRU198G.PDF)
- Matlab fdatool help (>> doc fdatool)
- Detailed CCS IDE and DSK drivers install guide at http://spinlab.wpi.edu/teaching.html

Latest TI documentation available at http://www.ti.com/sc/docs/psheets/man_dsp.htm