

Worksheet for the afternoon course

“Tune measurements simulated with a DSP card”

CAS Tuusula, June 2018

D. Alves, S. Sadovich, H. Schmickler

1. Introduction

In this course we will be replacing the betatron oscillations of the beam in the horizontal and vertical plane by two parallel LC resonators. The basic learning objective of the course is:

- Understand the different ways of stimulating transverse beam oscillations in time and frequency domain.
- Understand the signals of these transverse oscillations collected from a beam position monitors in time and frequency domain
- Generate various excitation modes with the help of an evaluation board carrying an “Analog Devices SHARC DSP processor”. The same DSP is also used for a treatment and display of the resulting “beam oscillation”.
The whole measurement sequence is programmed in C++, but the code is freely available and can be modified by skilled students.
- Generate a principle understanding of the necessary signal treatment for tune measurements and exercise the use of a simple two channel oscilloscope.

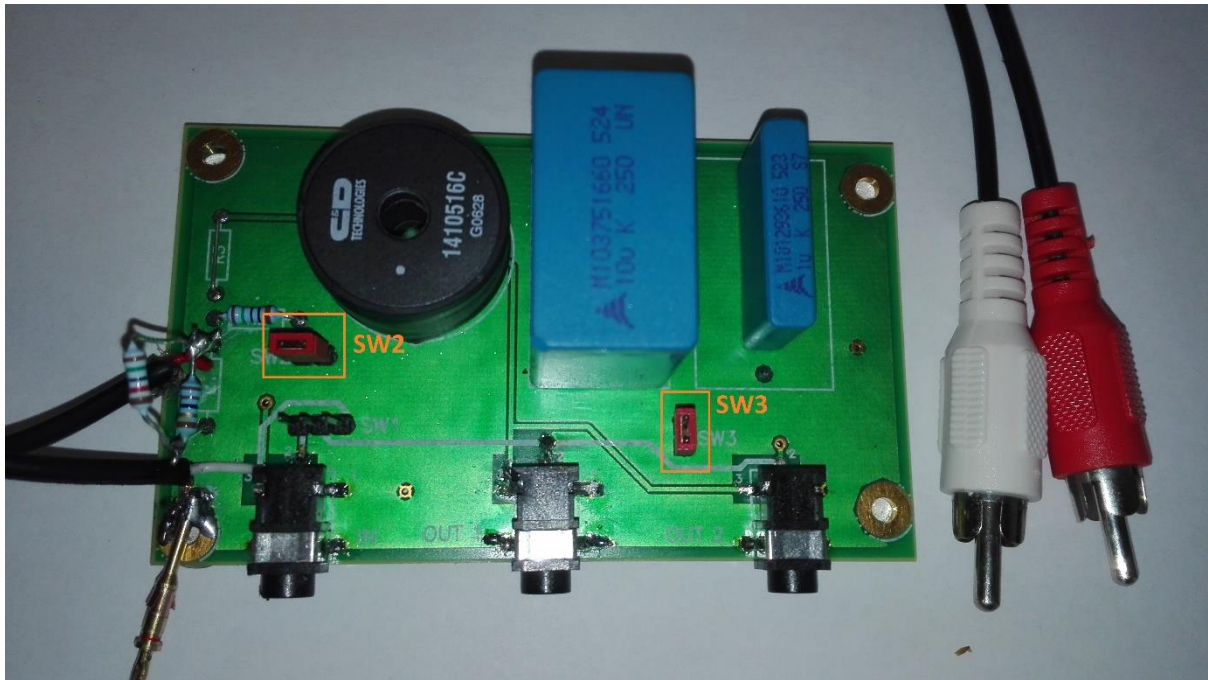
2. Description of the setup

Below the complete schematic of the two LC resonators representing the “beam”. The schematic has been generated with a freeware simulation tool, which is already installed on your computer.

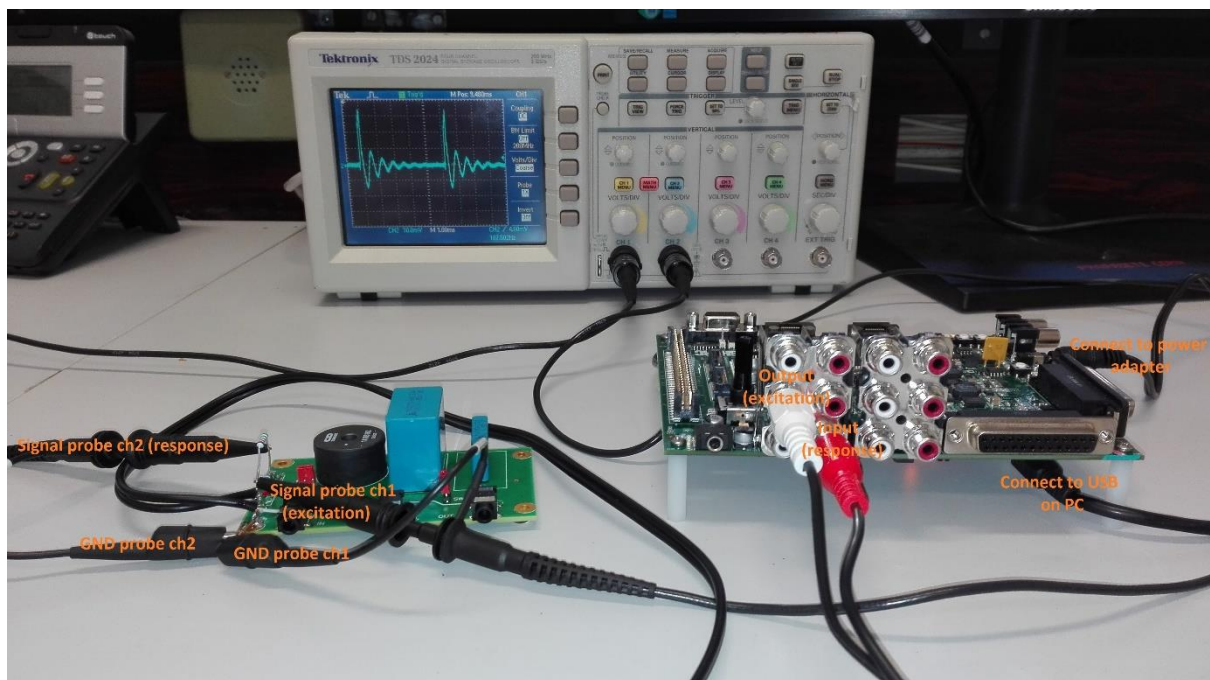
<http://www.linear.com/designtools/software/#LTspice>

With one jumper (SW3) on the beam PCB you can change the resonant frequency by adding one capacitor in parallel. Also you can put into another jumper (SW2) in order to put a resistor in parallel to the LC resonator. This emulates different damping times of the beam (i.e. chromaticity). During all exercises you can manipulate these jumpers and look at the resulting differences.

Please locate all components on the board...if you know a little of electronics you can read the values of the resistances from their colour code.

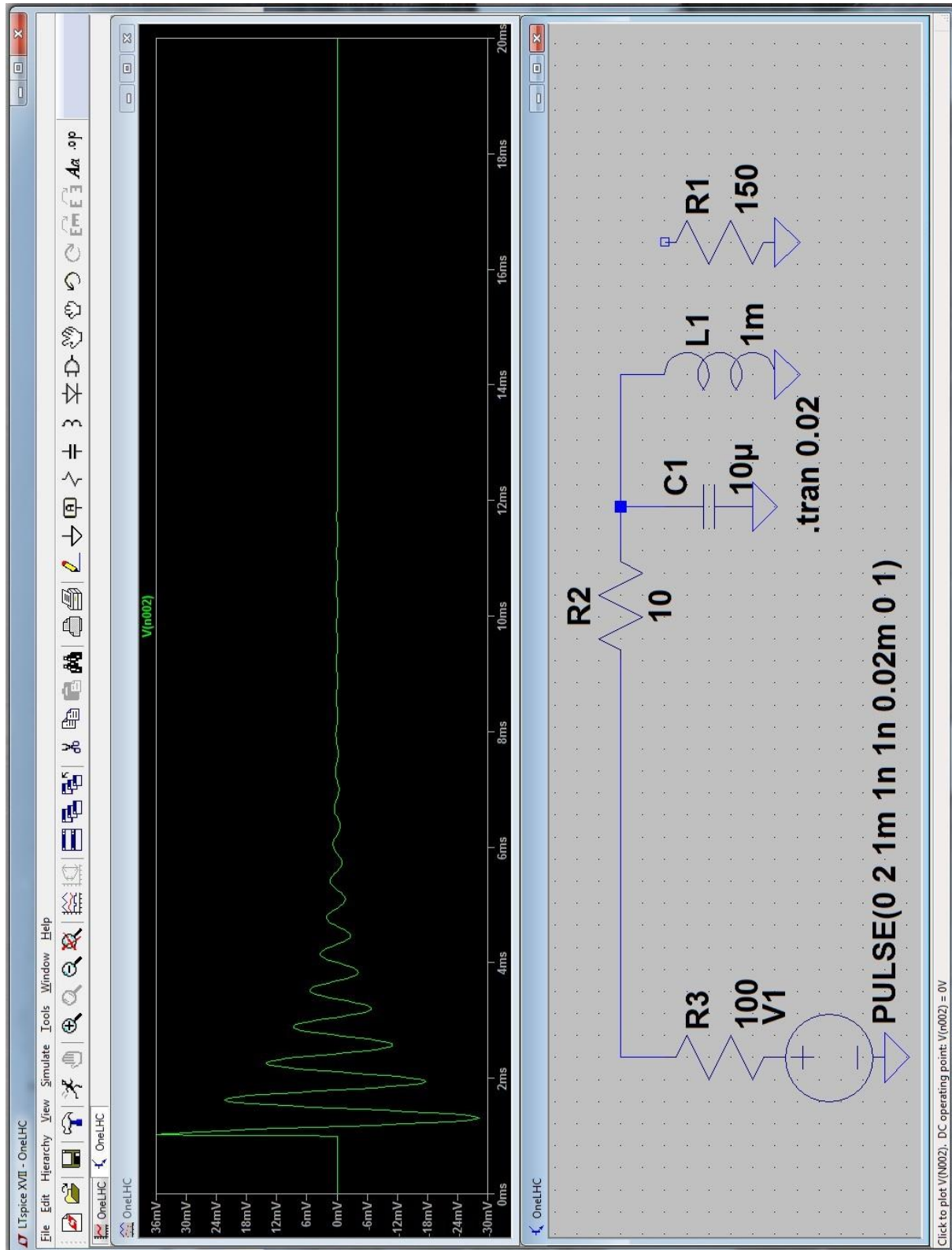


Both, the tool and the schematics are on the indico website of the CAS course and are installed on the course computer. This way every experiment can also be simulated and the results compared to real measurements. Some of the simulation snapshots will also be included in this document.



LTspice simulation:

The snapshot below shows the schematic (one beam) and its response for an excitation with a single pulse of 20 us length (exercise 3).

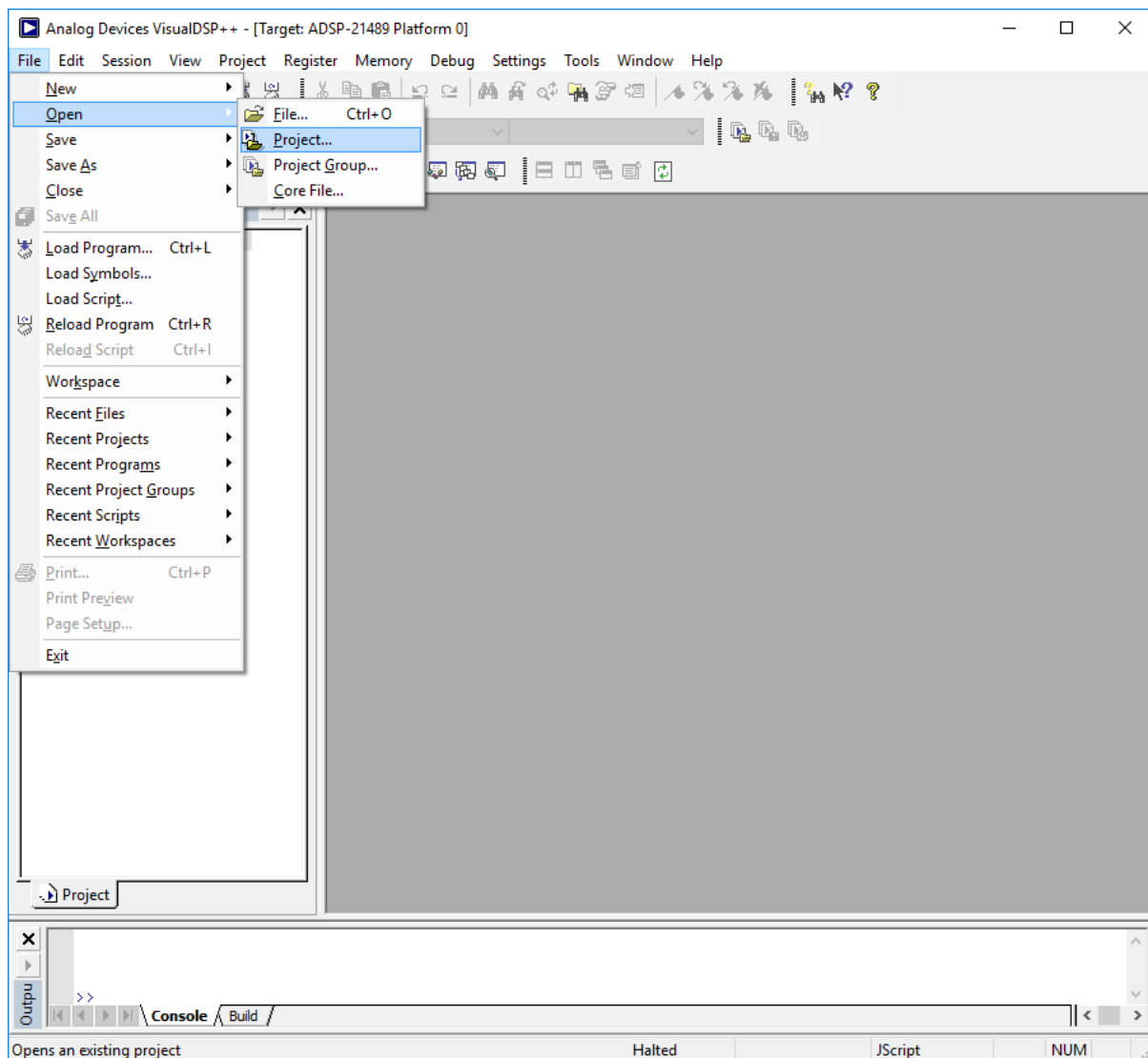


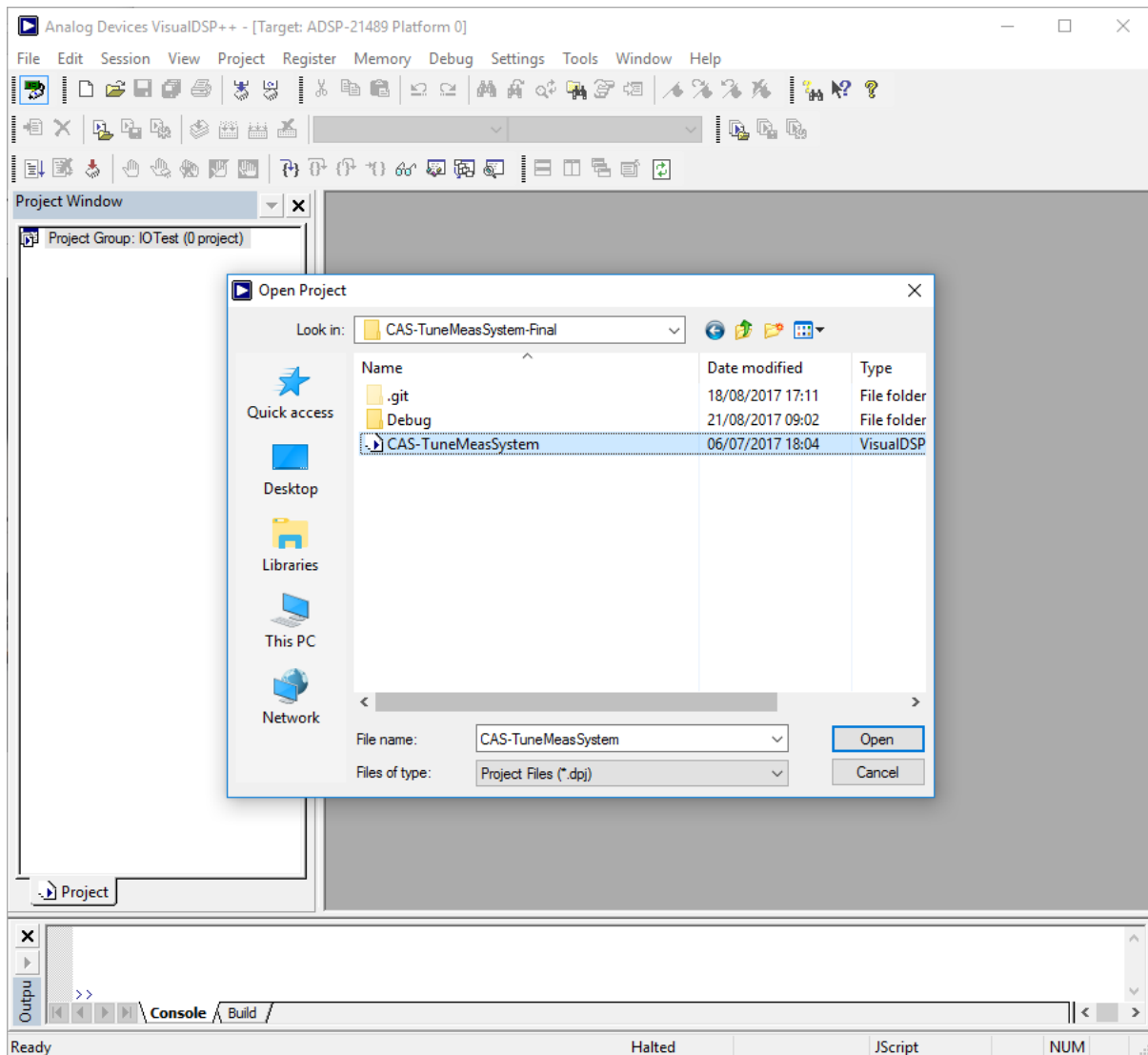
3. The DSP development environment

The DSP development environment is preinstalled on the note-books, which are linked by a USB cable to the evaluation kit.

After startup first the course project needs to be opened:

- Download zip file with project from the indico page of the afternoon courses in Tuusula.
- Unzip file to Desktop
- Import project into VisualDSP++

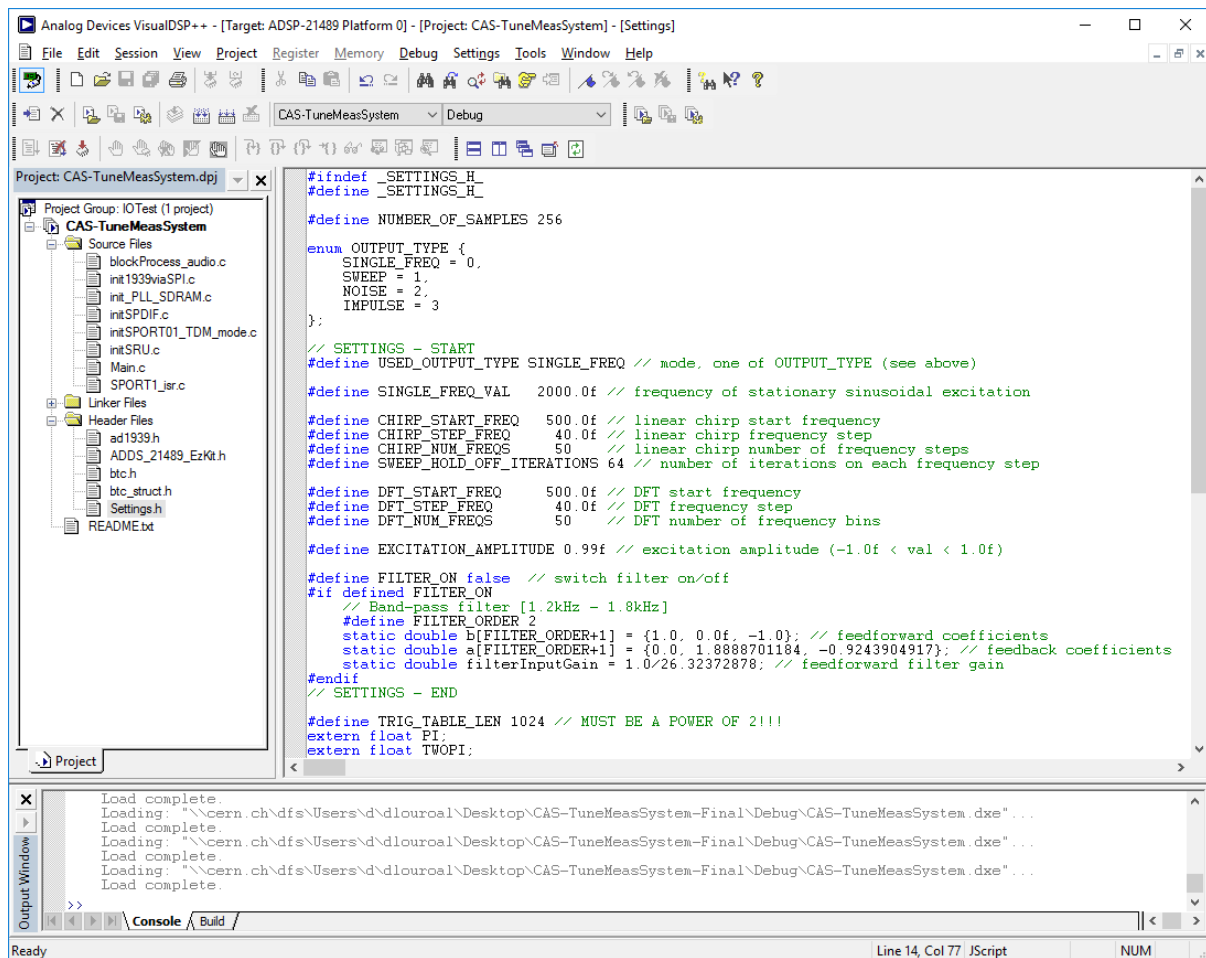




The software is not written with a graphical user interface. It is a single program, which can do all steps of the exercise, but for each step the program has to be recompiled after one has modified a parameter file, which configures the program for each step.

The parameter file is called "Settings.h" and the following picture shows part of its content. The meaning of each parameter shall be described in each exercise.

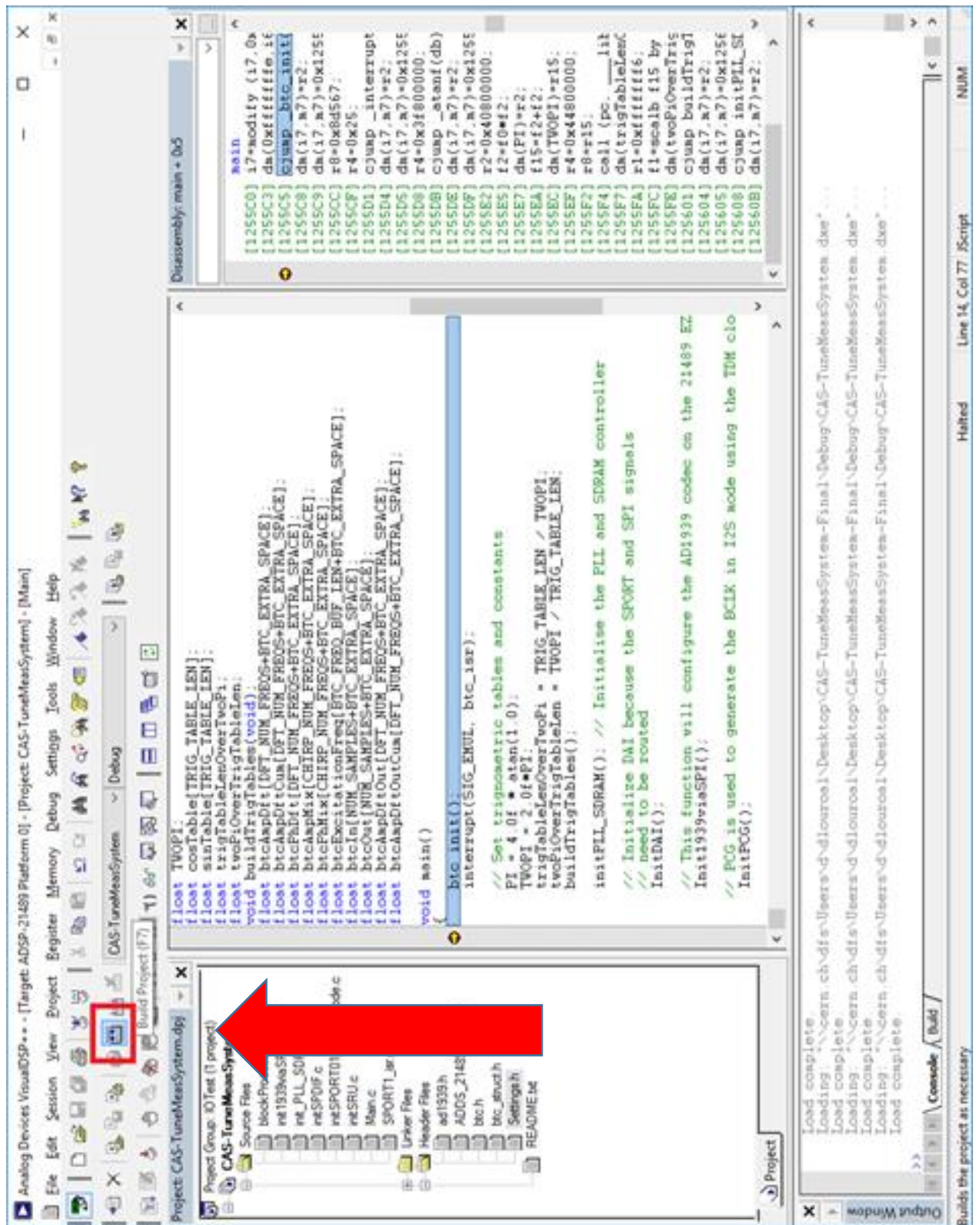
Configuration file: Settings.h



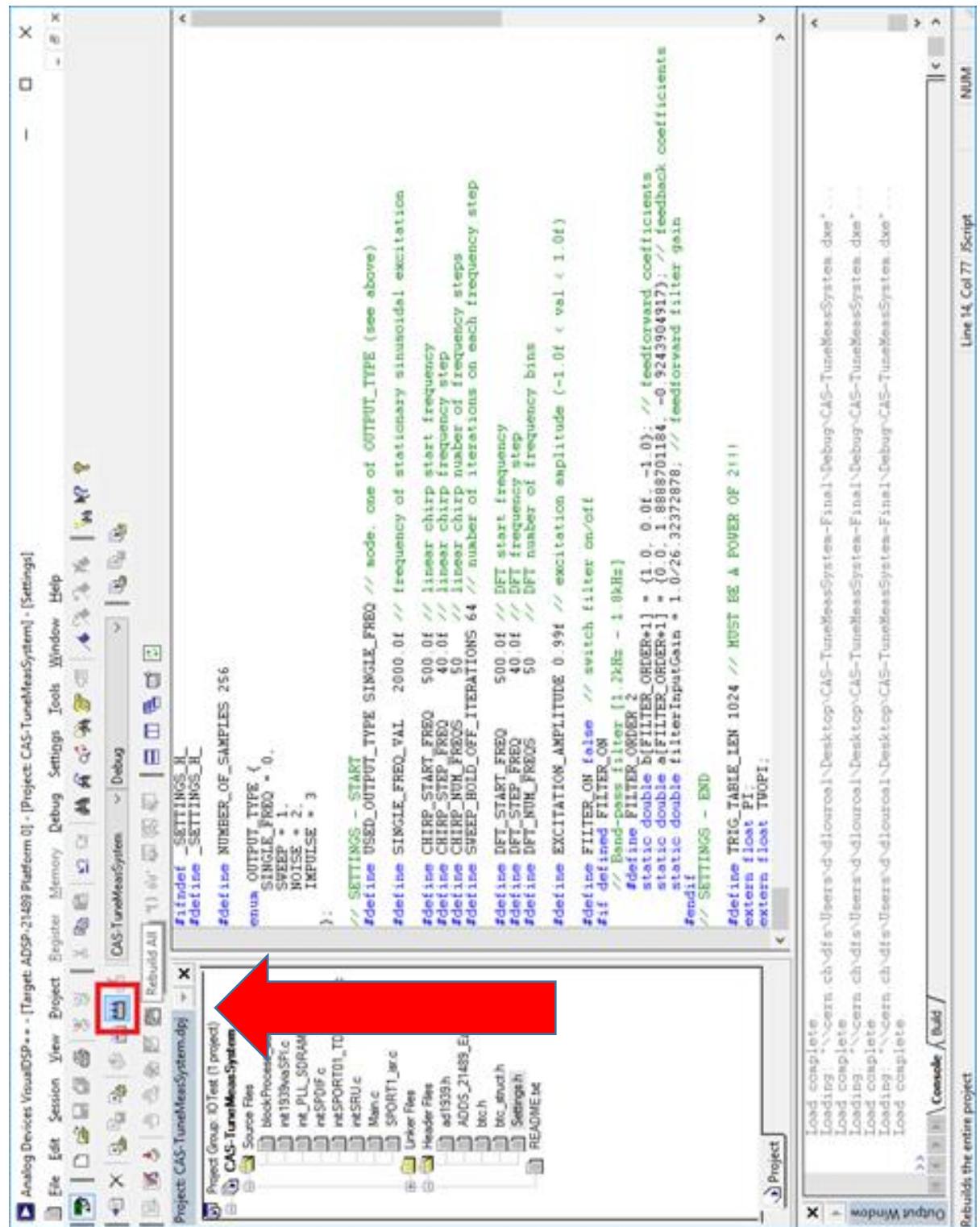
DSP environment, compiling.....

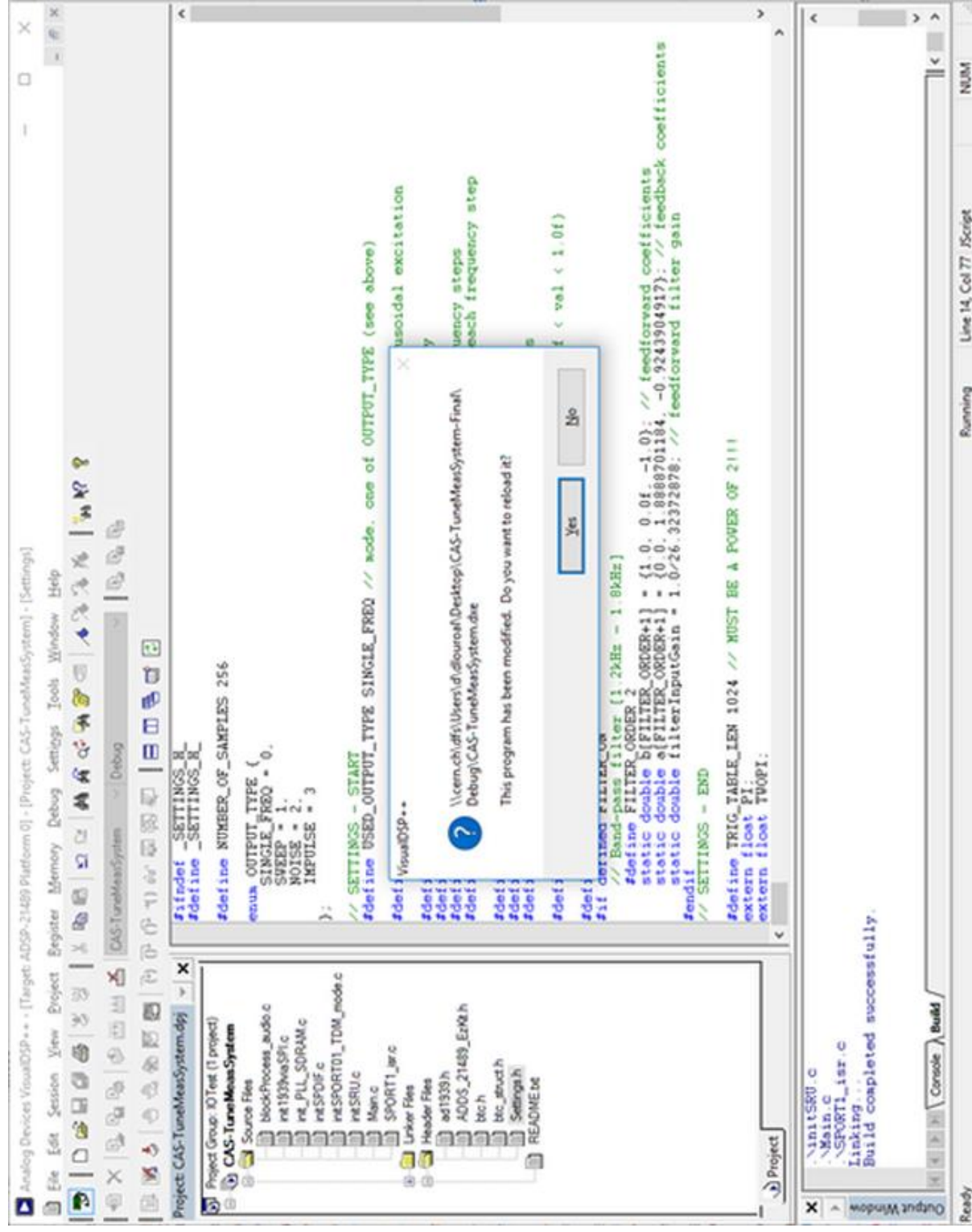
The following screen shots explain the various steps of invoking the program, compiling the code and loading the executable code into the DSP for execution. More explanations will be given in the course.

- Build project (first time you build it after importing):



- Build project (subsequent builds):

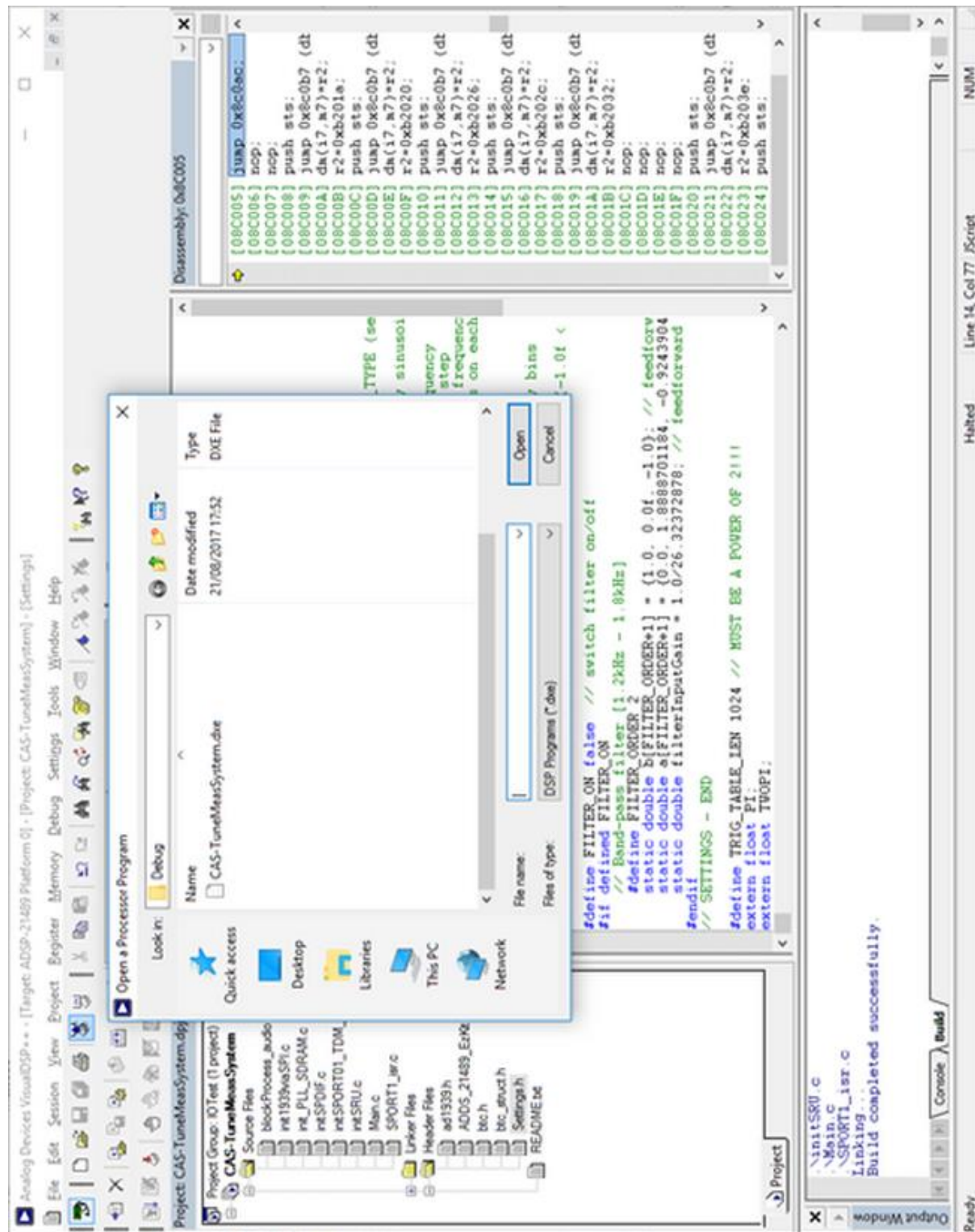




- Load compiled binary into DSP (automatic):

The screenshot shows the Analog Devices VisualDSP++ IDE interface. The top menu bar includes File, Edit, Session, View, Project, Register, Memory, Debug, Settings, Tools, Window, and Help. The toolbar contains various icons for file operations, debugging, and settings. The Project window on the left shows the project structure for 'CAS-TuneMeasSystem.dsp'. The Source Files list includes files like 'blockProcess_audio.c', 'm1133waSP.c', 'm1_PLL_SDRAM.c', 'm1SP01F.c', 'm1SPORT1_TDM.c', 'm1SRU.c', 'Main.c', 'SPORT1_hw.c', 'Linker Files', 'Header Files', 'ad1539.h', 'ADDS_21483_E2M', 'bch.h', 'bch_struct.h', 'Settings.h', and 'README.txt'. The Disassembly window on the right shows the assembly code for the 'Dissasembly: OutC005' section, starting with '[08C005] jump 0x8c0a0c;'. The Output Window at the bottom shows the build process, including the compilation of 'initSRU.c', 'Main.c', and 'SPORT1_hw.c', and the final message 'Build completed successfully.'

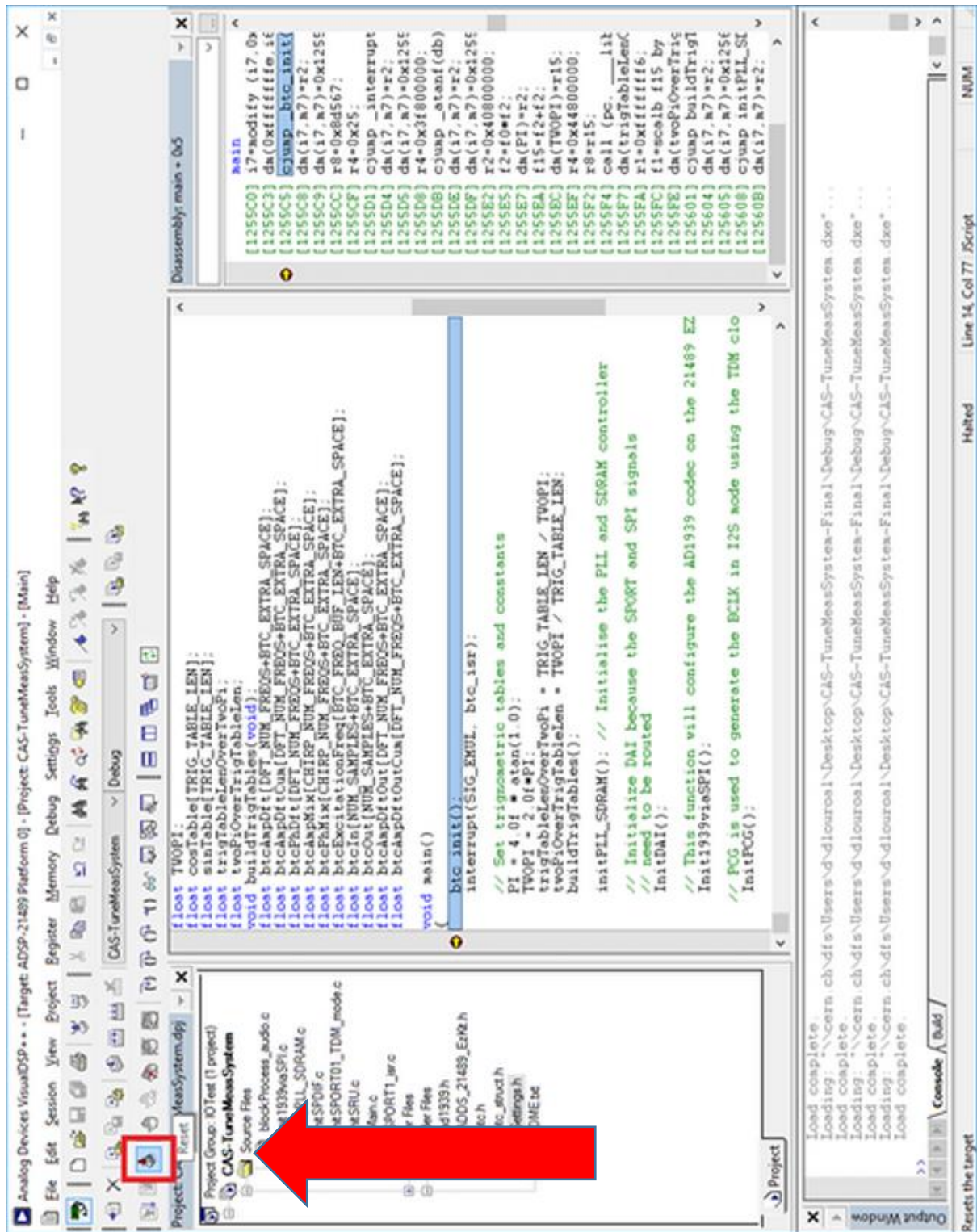
A red arrow points to the 'Load' button (represented by a disk icon) in the Project window toolbar, which is used to load the compiled binary into the DSP.



- Run code:

The screenshot displays the VisualDSP++ IDE interface for the project 'CAS-TuneMeasSystem'. The top toolbar shows the 'Run' button (a green play icon) highlighted with a red arrow. The main editor window shows the C source code for 'main.c', which includes initialization for trigonometric tables, constants, and the PLL/SRAM controller. The bottom window shows the disassembly of the 'main' function, with instructions like 'l17=modify (17,0x)', 'da(0xffffffe,1f)', and 'cjump_btc_init' visible. The rightmost pane shows the 'Output Window' with the message 'Load complete'.

- Stop and reset:



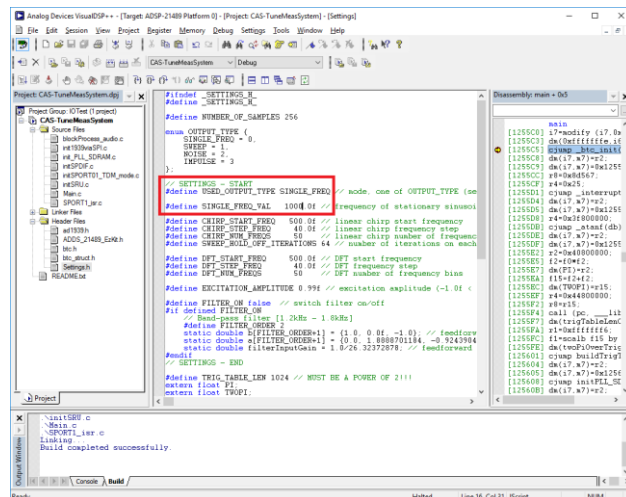
Exercises:

Exercise 0: Getting familiar

Put the setup into the mode of “continuous sine wave excitation” (=: SINGLE_FREQ) and make all connections to the DSP.

Use one channel of the oscilloscope to observe the excitation signal of the beam, the other channel to observe the beam oscillation. Choose as excitation frequency 1000 Hz, (re)build and run.

Parameter page:



Details:

```
// SETTINGS - START
#define USED_OUTPUT_TYPE SINGLE_FREQ // mode, one of OUTPUT_TYPE (see above)
#define SINGLE_FREQ_VAL 1000.0f // frequency of stationary sinusoidal exci

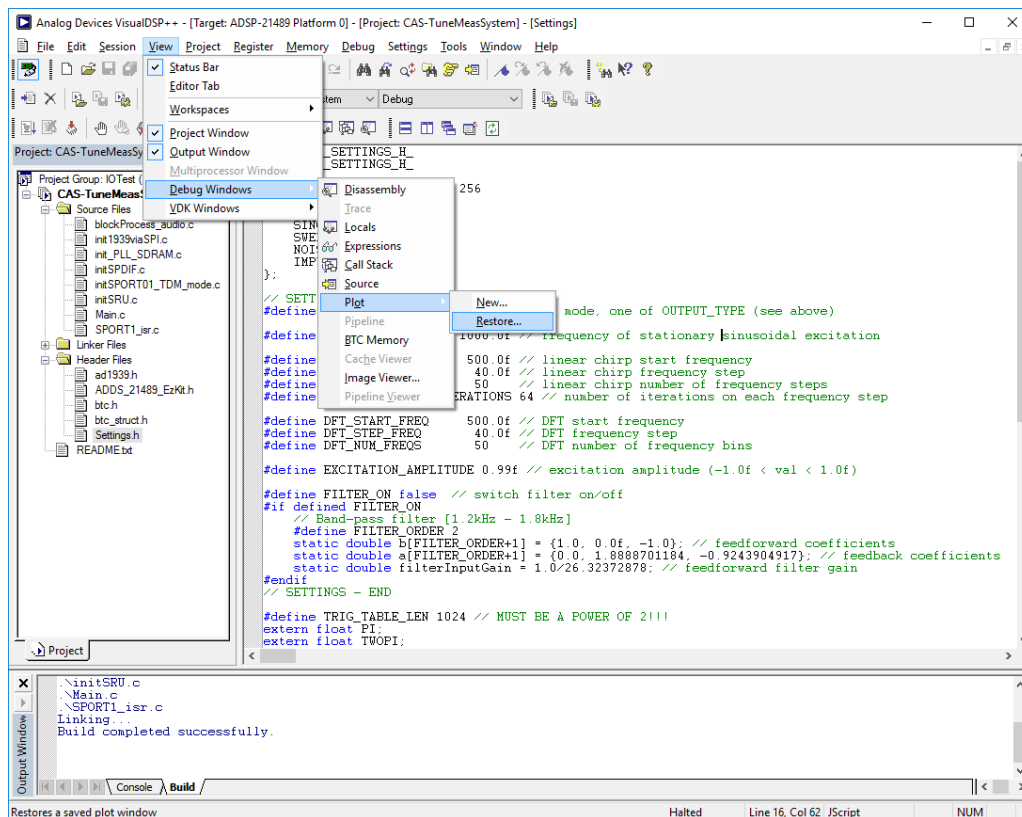
#define CHIRP_START_FREQ 500.0f // linear chirp start frequency
#define CHIRP_STEP_FREQ 40.0f // linear chirp frequency step
#define CHIRP_NUM_FREQS 50 // linear chirp number of frequency steps
#define SWEEP_HOLD_OFF_ITERATIONS 64 // number of iterations on each frequen

#define DFT_START_FREQ 500.0f // DFT start frequency
#define DFT_STEP_FREQ 40.0f // DFT frequency step
#define DFT_NUM_FREQS 50 // DFT number of frequency bins

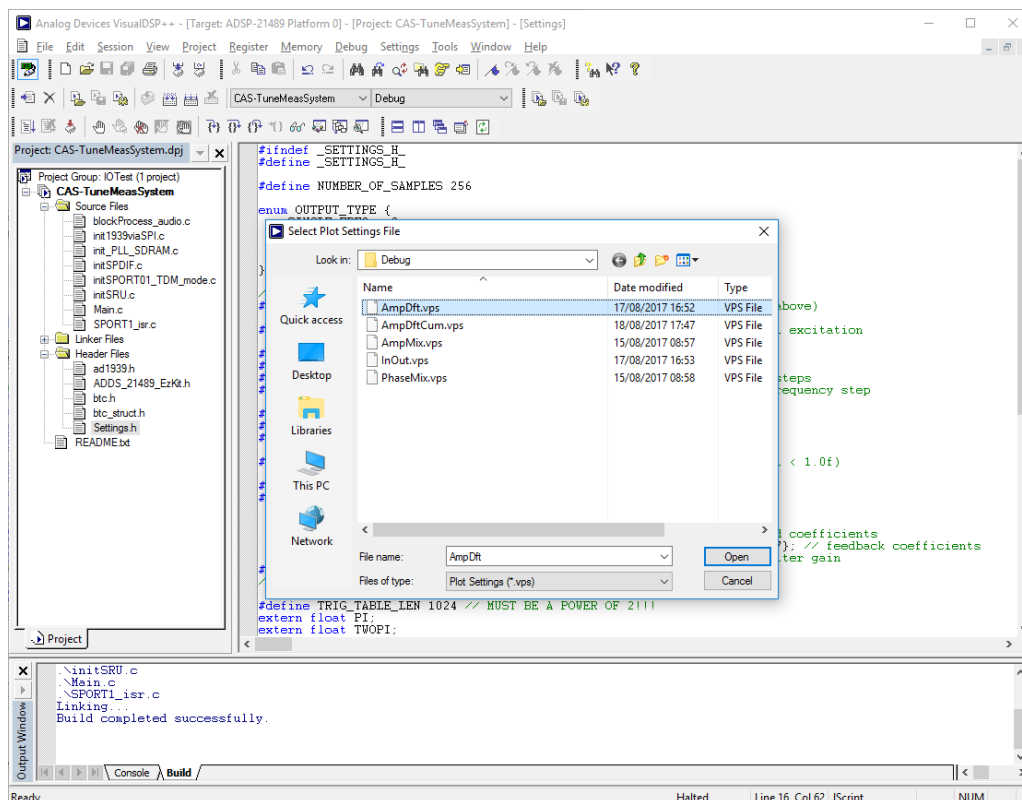
#define EXCITATION_AMPLITUDE 0.99f // excitation amplitude (-1.0f < val < 1.0f)

#define FILTER_ON true // switch filter on/off
#if defined FILTER_ON
    // Band-pass filter [1.2kHz - 1.8kHz]
    #define FILTER_ORDER 2
    static double b[FILTER_ORDER+1] = {1.0, 0.0f, -1.0}; // feedforward coef
    static double a[FILTER_ORDER+1] = {0.0, 1.8888701184, -0.9243904917}; //
    static double filterInputGain = 1.0/26.32372878; // feedforward filter g
#endif
// SETTINGS - END
```

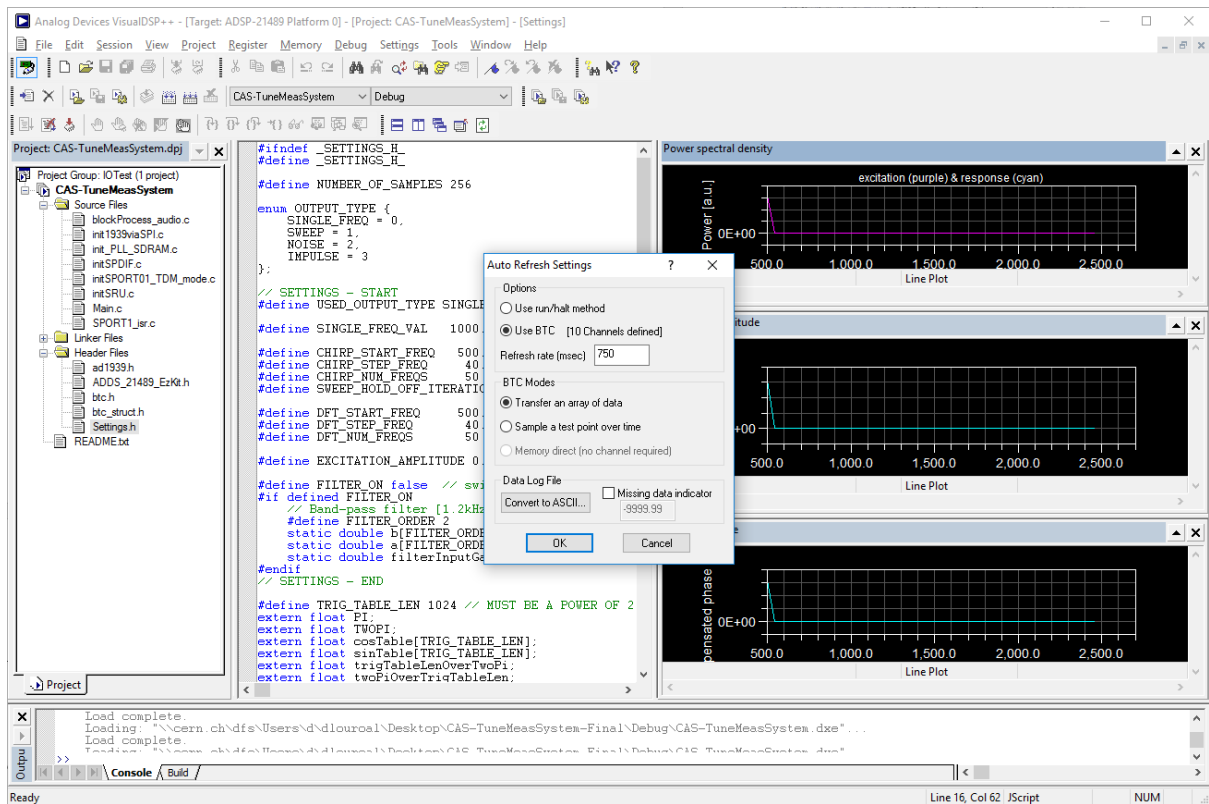
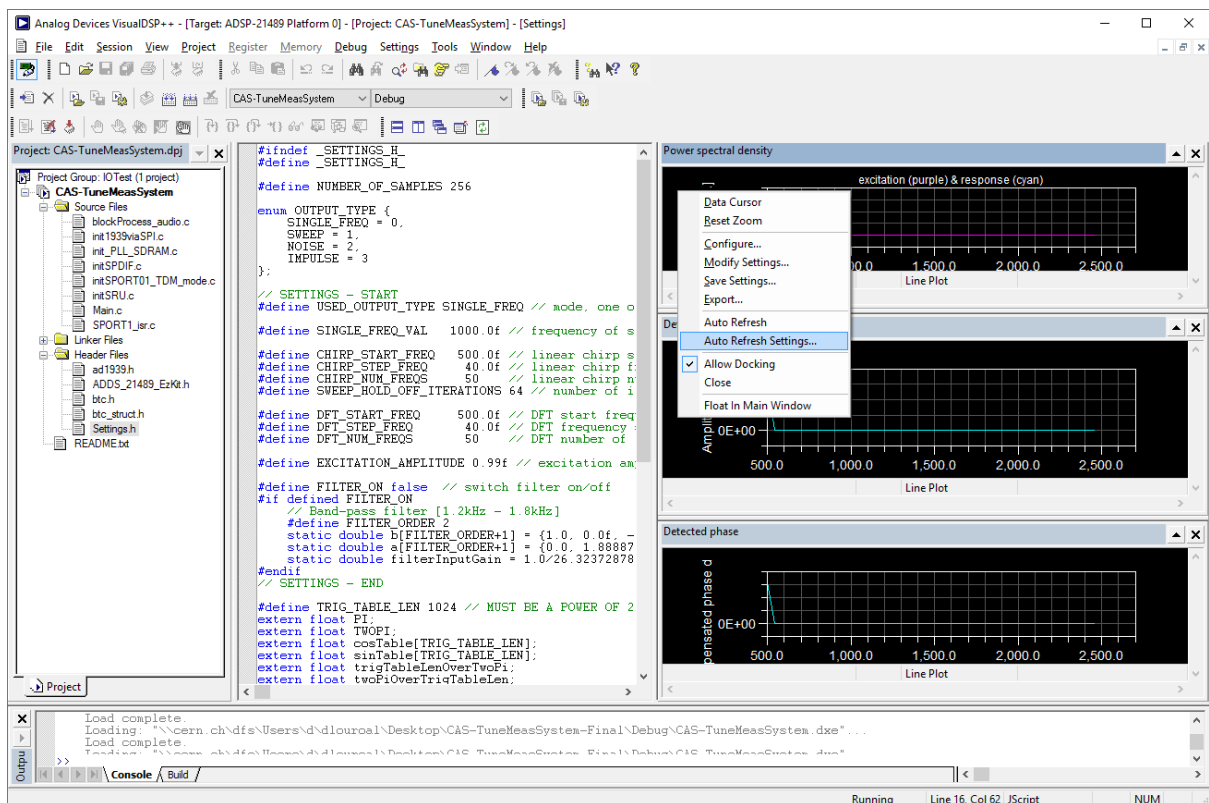
Getting the realtime displays generated by the DSP:



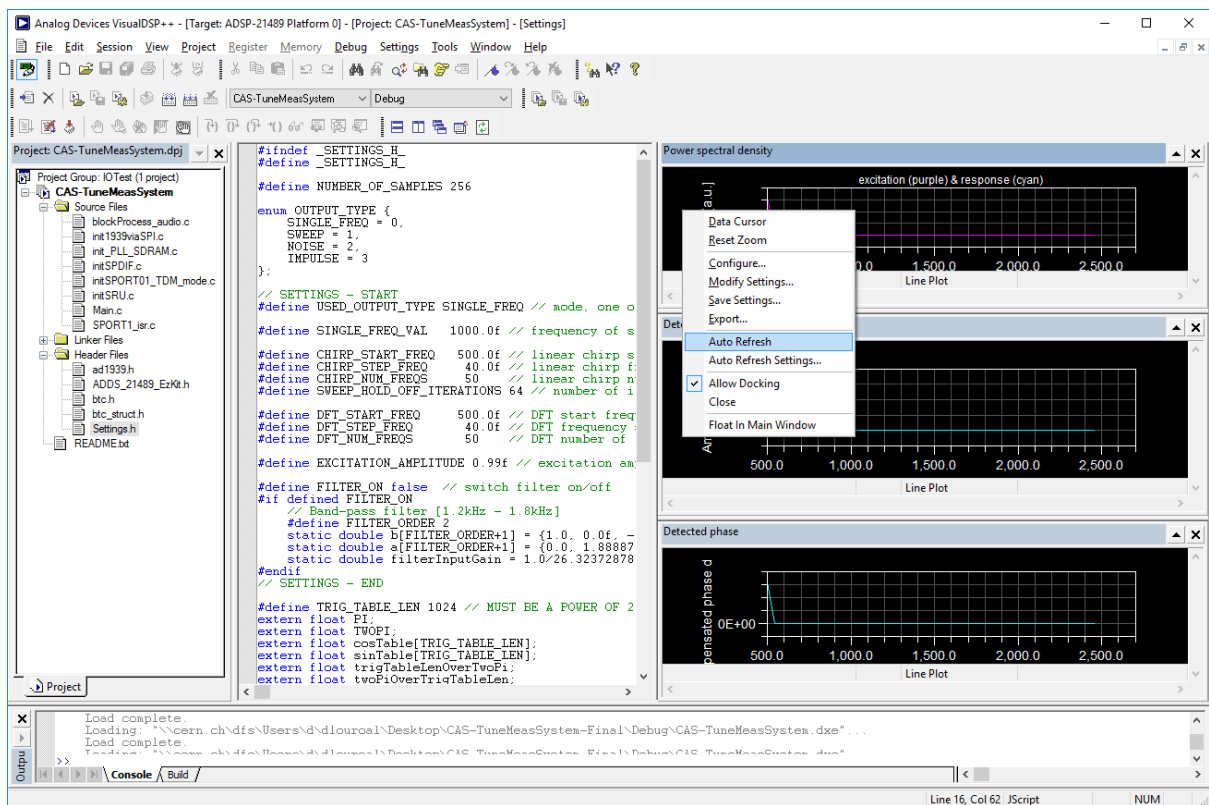
Load AmpDft.vps, AmpMix.vps and PhaseMix.vps:



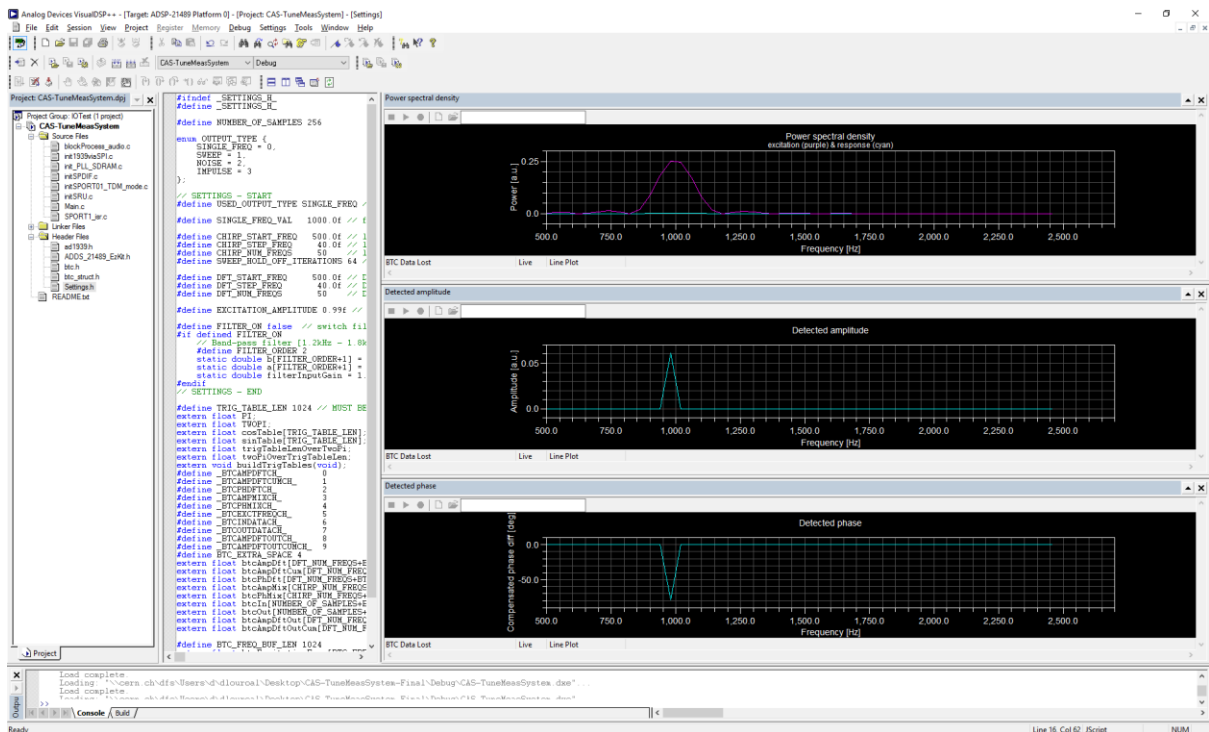
Right-click on any plot window to setup auto-refresh settings.



Right-click on each plot window and enable auto refresh:



Finally you should see:

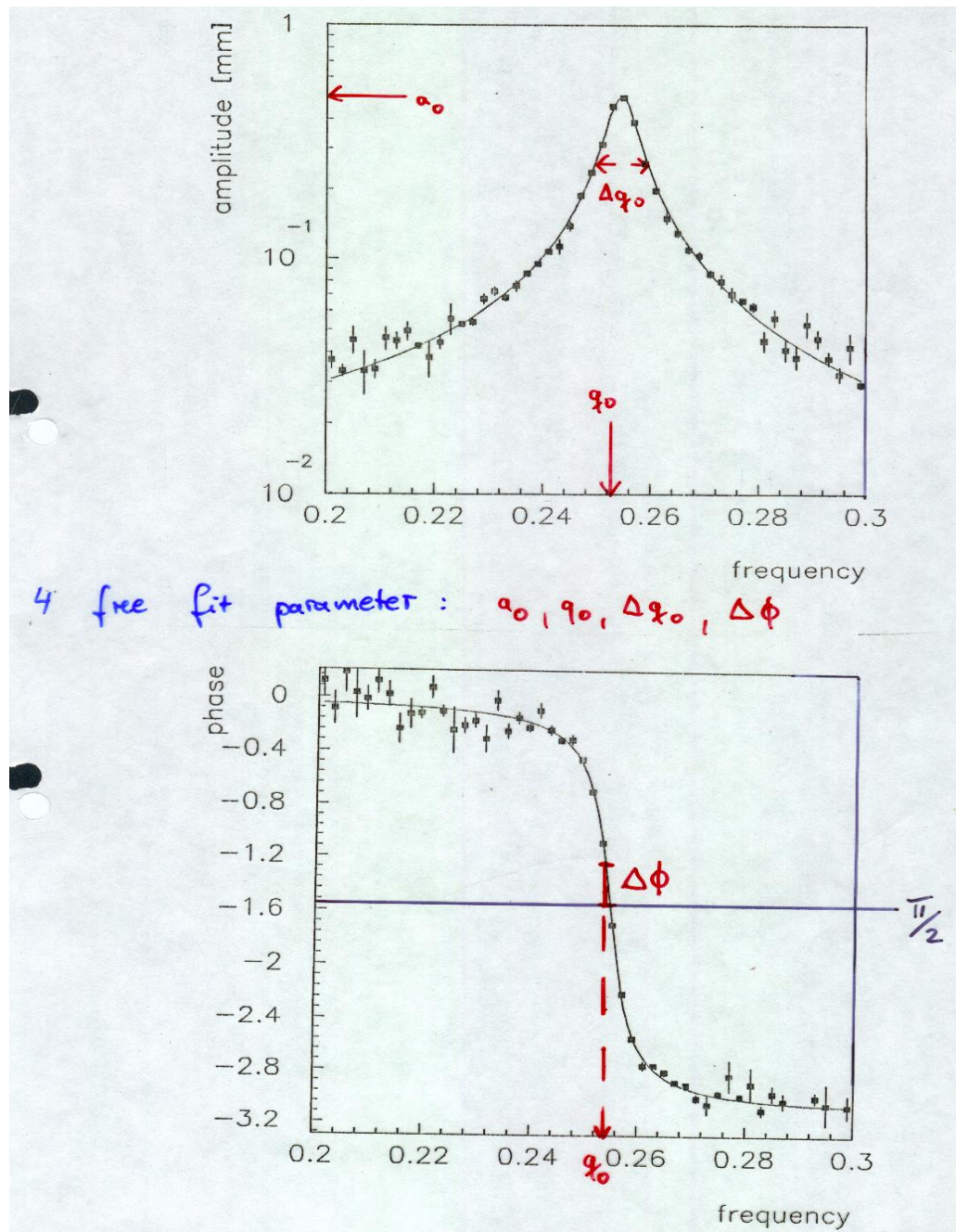


Exercise 1: Measuring by hand the beam transfer function

Use the setup of above, but vary the fixed frequency in at least 7 steps in the range 500 Hz to 2500 Hz. At each frequency measure the amplitude of the beam oscillation and the phase between excitation and beam oscillation directly from the AmpMix and PhaseMix plots.

Draw on a piece of paper or in excel the results as a function of the excitation frequency.

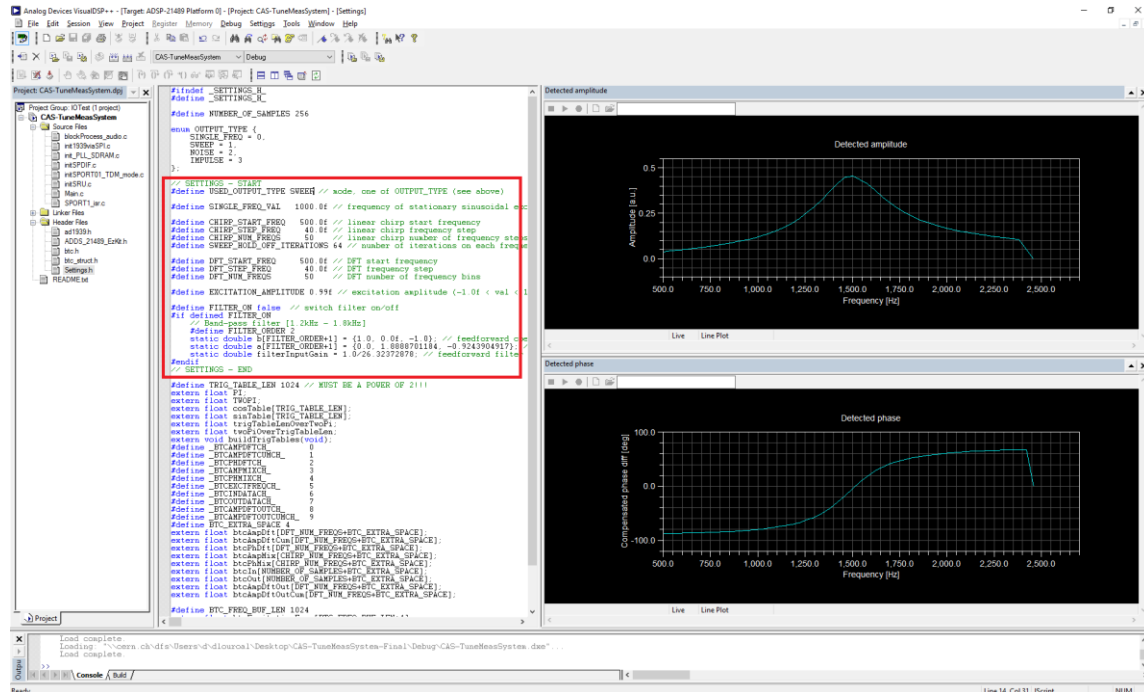
Below: Measurement on beam from about 1991



Exercise 2: Have the DSP do exercise 1

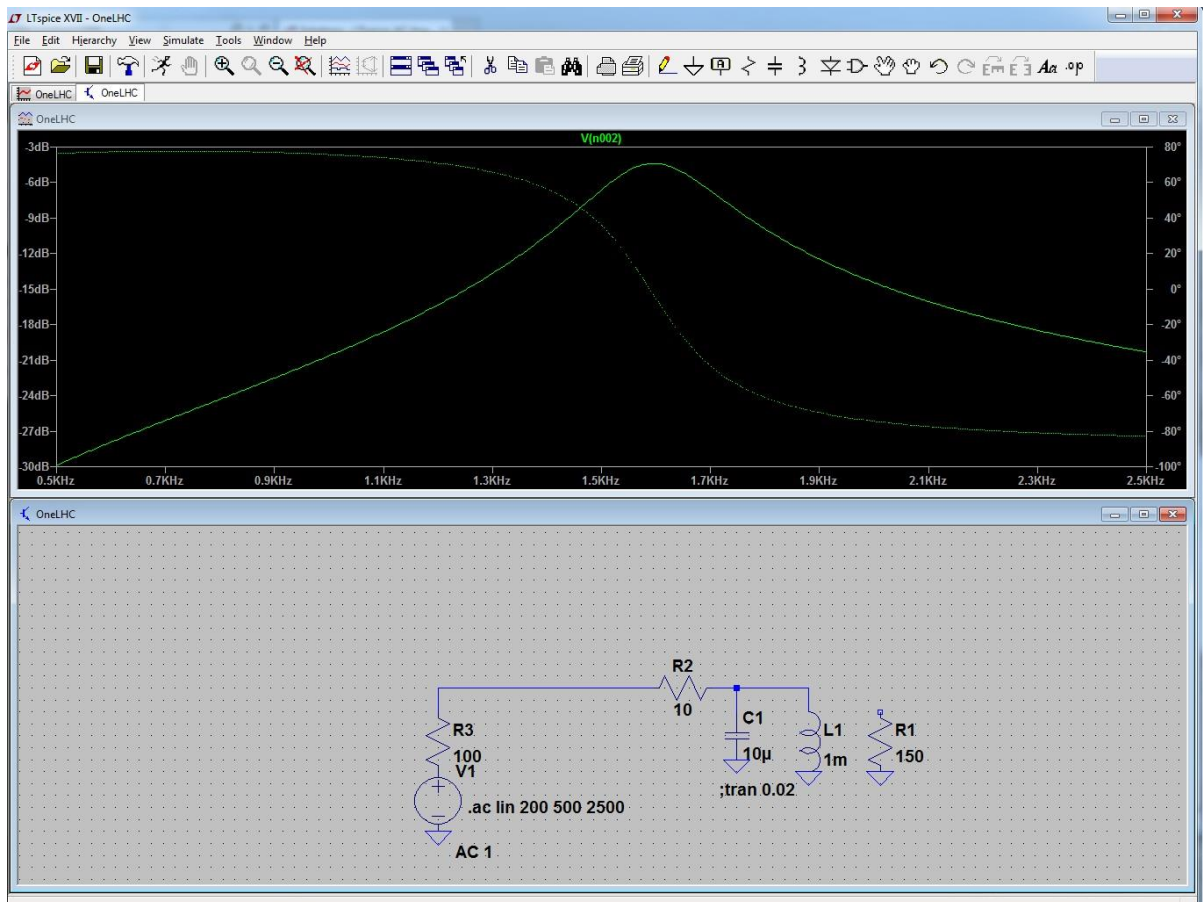
Use again the same setup, but this time put the DSP into the mode “SWEPT FREQUENCY” also often called network analysis. Use as sweep range again 500...2500 Hz.

In real time the DSP will draw the curves which you have measured yourself.



Details:

```
// SETTINGS - START
#define USED_OUTPUT_TYPE SWEEP // mode, one of OUTPUT_TYPE (see above)
#define SINGLE_FREQ_VAL 1000.0f // frequency of stationary sinusoidal exci
#define CHIRP_START_FREQ 500.0f // linear chirp start frequency
#define CHIRP_STEP_FREQ 40.0f // linear chirp frequency step
#define CHIRP_NUM_FREQS 50 // linear chirp number of frequency steps
#define SWEEP_HOLD_OFF_ITERATIONS 64 // number of iterations on each frequen
#define DFT_START_FREQ 500.0f // DFT start frequency
#define DFT_STEP_FREQ 40.0f // DFT frequency step
#define DFT_NUM_FREQS 50 // DFT number of frequency bins
#define EXCITATION_AMPLITUDE 0.99f // excitation amplitude (-1.0f < val < 1.0f)
#define FILTER_ON false // switch filter on/off
// Band-pass filter [1.2kHz - 1.8kHz]
#define FILTER_ORDER 2
static double b[FILTER_ORDER+1] = {1.0, 0.0f, -1.0}; // feedforward coef
static double a[FILTER_ORDER+1] = {0.0, 1.8888701184, -0.9243904917}; //
static double filterInputGain = 1.0/26.32372878; // feedforward filter g
// SETTINGS - END
```



Above: Simulated result from LTspice

Exercise 3: Broad band excitation

We now switch to a different excitation mode of the beam, which we refer to as “noise excitation”. The advantage is that by applying noise, i.e. random excitation voltages, we excite the whole frequency spectrum, so we do can do the above measurements at the same time, with the price of reduced accuracy.

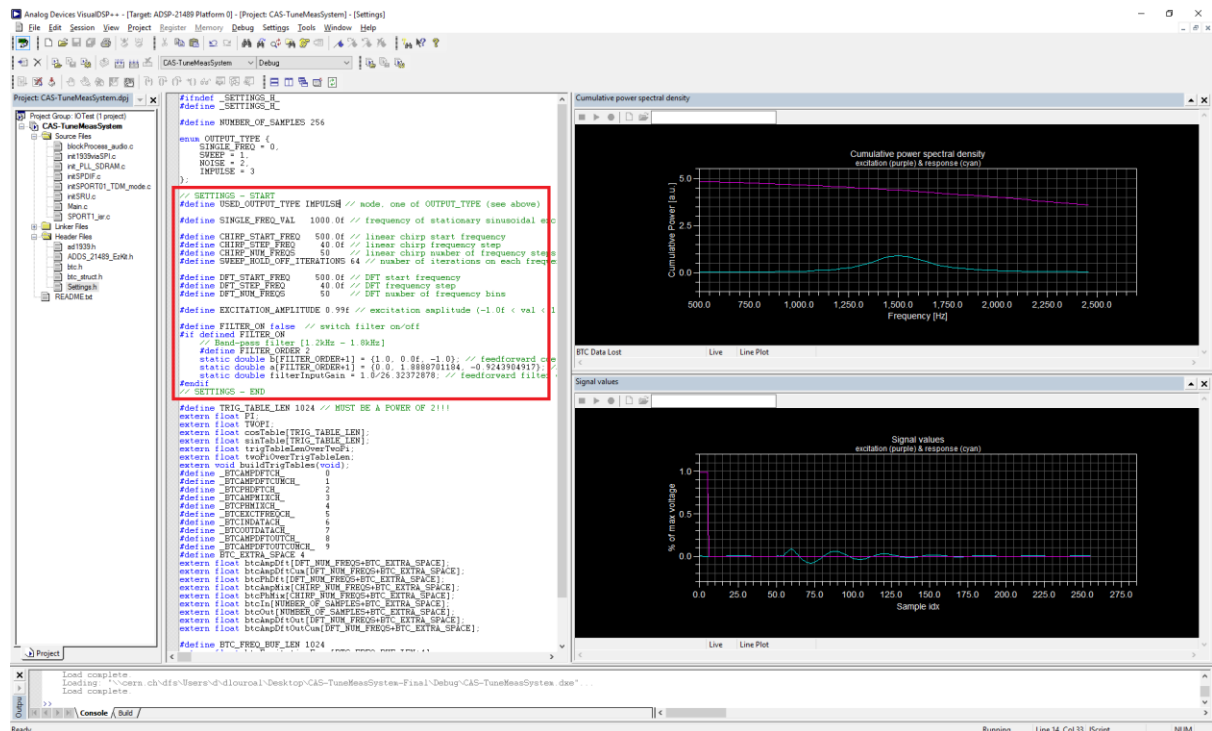
Noise can be applied in various ways, we offer three different options

- single unipolar excitation pulse
- regular random voltage excitation pulses
- regular random excitation pulses after applying a filter, which is centred on the expected resonance frequency of the beam.

The advantages and disadvantages of each excitation mode will become clear during the exercise.

3a) Set the DSP to the mode “SINGLE PULSE EXCITATION”

Observe with the oscilloscope the excitation signal (use this also to trigger the scope) and observe the beam oscillation. Make the appropriate code changes and load AmpDftCum.vps and InOut.vps.



Details of parameter file:

```

// SETTINGS - START
#define USED_OUTPUT_TYPE IMPULSE // mode, one of OUTPUT_TYPE (see above)

#define SINGLE_FREQ_VAL 1000.0f // frequency of stationary sinusoidal excitation

#define CHIRP_START_FREQ 500.0f // linear chirp start frequency
#define CHIRP_STEP_FREQ 40.0f // linear chirp frequency step
#define CHIRP_NUM_FREQS 50 // linear chirp number of frequency steps
#define SWEEP_HOLD_OFF_ITERATIONS 64 // number of iterations on each frequency

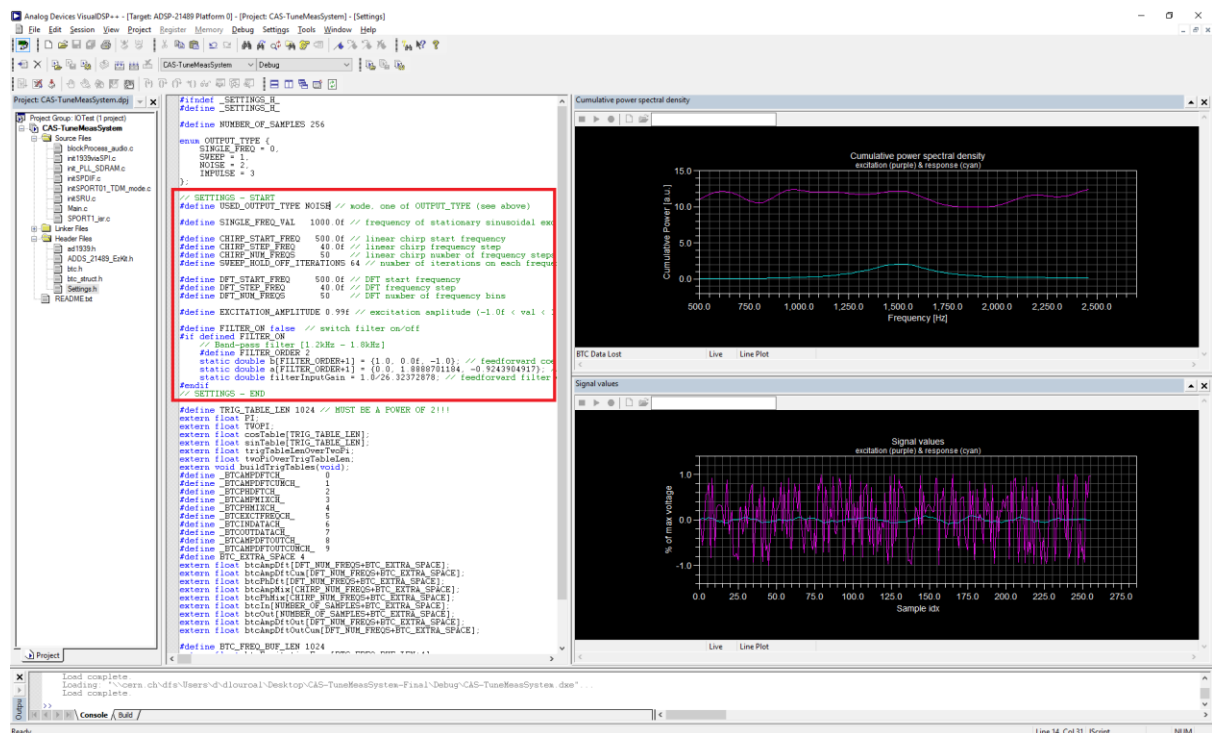
#define DFT_START_FREQ 500.0f // DFT start frequency
#define DFT_STEP_FREQ 40.0f // DFT frequency step
#define DFT_NUM_FREQS 50 // DFT number of frequency bins

#define EXCITATION_AMPLITUDE 0.99f // excitation amplitude (-1.0f < val < 1.0f)

#define FILTER_ON false // switch filter on/off
#if defined FILTER_ON
    // Band-pass filter [1.2kHz - 1.8kHz]
    #define FILTER_ORDER 2
    static double b[FILTER_ORDER+1] = {1.0, 0.0f, -1.0}; // feedforward coefficients
    static double a[FILTER_ORDER+1] = {0.0, 1.8888701184, -0.9243904917}; // feedback coefficients
    static double filterInputGain = 1.0/26.32372878; // feedforward filter gain
#endif
// SETTINGS - END

```


3b) Set the DSP into the mode of “NOISE EXCITATION, FILTER OFF”



```
// SETTINGS - START
#define USED_OUTPUT_TYPE NOISE // mode, one of OUTPUT_TYPE (see above)

#define SINGLE_FREQ_VAL 1000.0f // frequency of stationary sinusoidal exc.

#define CHIRP_START_FREQ 500.0f // linear chirp start frequency
#define CHIRP_STEP_FREQ 40.0f // linear chirp frequency step
#define CHIRP_NUM_FREQS 50 // linear chirp number of frequency steps
#define SWEEP_HOLD_OFF_ITERATIONS 64 // number of iterations on each frequency

#define DFT_START_FREQ 500.0f // DFT start frequency
#define DFT_STEP_FREQ 40.0f // DFT frequency step
#define DFT_NUM_FREQS 50 // DFT number of frequency bins

#define EXCITATION_AMPLITUDE 0.99f // excitation amplitude (-1.0f < val < 1.0f)

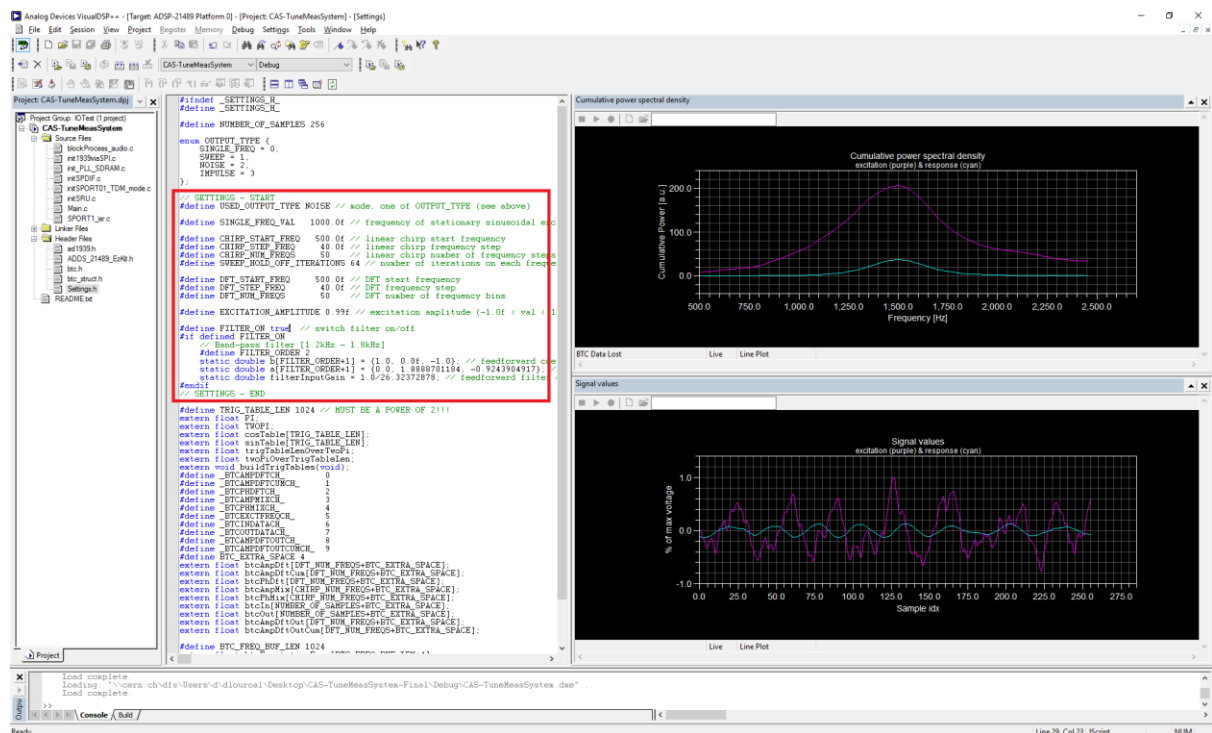
#define FILTER_ON false // switch filter on/off
#if defined FILTER_ON
// Band-pass filter [1.2kHz - 1.8kHz]
#define FILTER_ORDER 2
static double b[FILTER_ORDER+1] = {1.0, 0.0f, -1.0}; // feedforward coefficients
static double a[FILTER_ORDER+1] = {0.0, 1.8888701184, -0.9243904917}; // feedback coefficients
static double filterInputGain = 1.0/26.32372878; // feedforward filter gain
#endif
// SETTINGS - END
```

The DSP will work in an endless loop, each time sending excitation pulses, collecting the data and computing a discrete Fourier transform in the frequency range 500...2500 Hz. The amplitude spectrum of the excitation and the beam are displayed.

In the top two graphs shot by shot, and the bottom two plots as accumulated spectra. Interpret the results.

If wanted you can also observe the time domain data with the oscilloscope, but it is difficult to get a meaningful measurement.

3c) Set the DSP into the mode of “NOISE EXCITATION, FILTER ON”



```
// SETTINGS - START
#define USED_OUTPUT_TYPE NOISE // mode, one of OUTPUT_TYPE (see above)

#define SINGLE_FREQ_VAL 1000.0f // frequency of stationary sinusoidal excitation

#define CHIRP_START_FREQ 500.0f // linear chirp start frequency
#define CHIRP_STEP_FREQ 40.0f // linear chirp frequency step
#define CHIRP_NUM_FREQS 50 // linear chirp number of frequency steps
#define SWEEP_HOLD_OFF_ITERATIONS 64 // number of iterations on each frequency

#define DFT_START_FREQ 500.0f // DFT start frequency
#define DFT_STEP_FREQ 40.0f // DFT frequency step
#define DFT_NUM_FREQS 50 // DFT number of frequency bins

#define EXCITATION_AMPLITUDE 0.99f // excitation amplitude (-1.0f < val < 1.0f)

#define FILTER_ON true // switch filter on/off
#if defined FILTER_ON
    // Band-pass filter [1.2kHz - 1.8kHz]
    #define FILTER_ORDER 2
    static double b[FILTER_ORDER+1] = {1.0, 0.0f, -1.0}; // feedforward coefficients
    static double a[FILTER_ORDER+1] = {0.0, 1.8888701184, -0.9243904917}; // feedback coefficients
    static double filterInputGain = 1.0/26.32372878; // feedforward filter gain
#endif
// SETTINGS - END
```

The DSP will work in an endless loop, each time sending excitation pulses, collecting the data and computing a discrete Fourier transform in the frequency range 500...2500 Hz.

The only difference compared to exercise 4b is that the noise excitation is pre-filtered around the expected resonant frequency (1600 Hz; jumper for extra C “out”).

Interpret the changes to 3b)

Details on the filter:

The filter has been implemented in order to concentrate the excitation energy of the noise to the frequency range of interest.

The following procedure is implemented in the code:

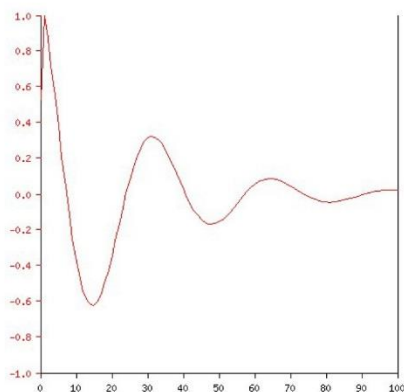
- apply a digital Butterworth bandpass of 1st order
- determine the peak to peak signal level after filtering
- apply a gain to these values, such that the maximum DAC input (-1...+1) is used.

Possible Discussion: Pros and cons of this method? Alternatives?

Filter data: Sampling frequency 48 kHz

Lower corner frequency 1200 Hz, higher corner frequency 1800 Hz

Below the simulated filter amplitude and phase response plus its response to a kick excitation.

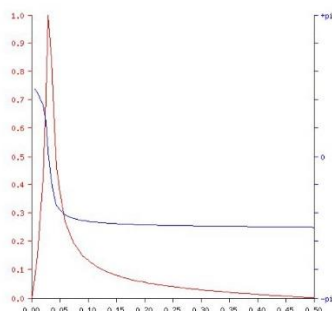


Step response

- x axis: time, in samples (i.e. 48000 represents 1 second)
- y axis (red): filter response (linear, normalized)

Magnitude (red) and phase (blue) vs. frequency

- x axis: frequency, as a fraction of the sampling rate (i.e. 0.5 represents the Nyquist frequency, which is 24000 Hz)
- y axis (red): magnitude (linear, normalized)
- y axis (blue): phase



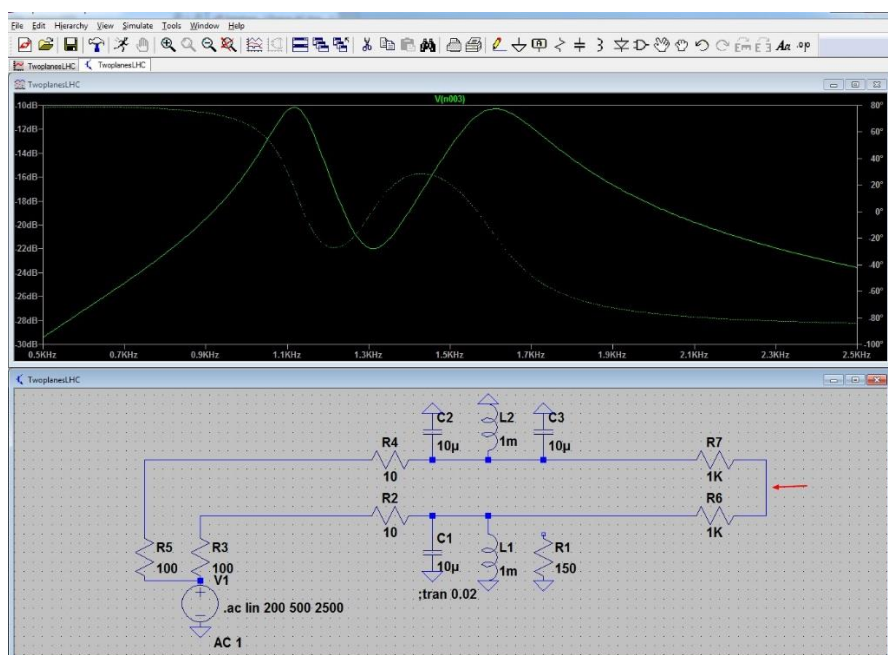
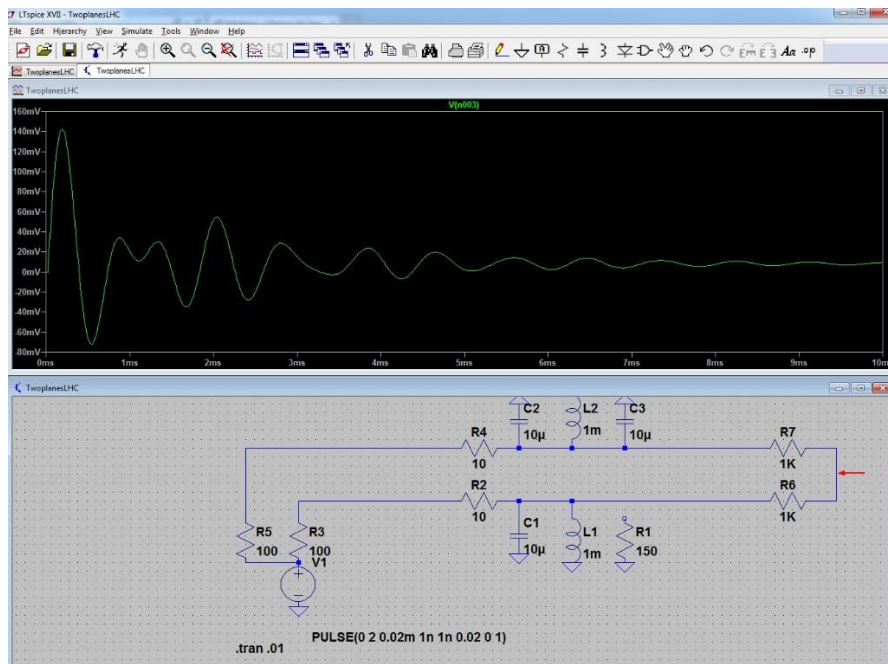
4.) Coupling

Output 1 and 2 always produce the same excitation signal. So you can connect the excitation signal to both planes of the beam in parallel.

Without the jumper inserted to the middle part of the PCB, both planes are “uncoupled”.

Inserting this jumper produces a simple way of simulating (Strong) coupling of both betatron planes.

Please repeat exercise 3a and 3b with coupled beam. Best results are obtained with both resonant frequencies different. So one jumper “IN” and the same jumper on the other board “OUT”. Below the results in simulation.



4. Phase Locked Loop (PLL) tune tracking

All tune diagnostics exercised so far results in a single shot tune measurement. This is for many situations sufficient, but people also want to diagnose operational transitions of an accelerator (like acceleration) or any other manipulation, during which magnetic fields in the accelerator get changed.

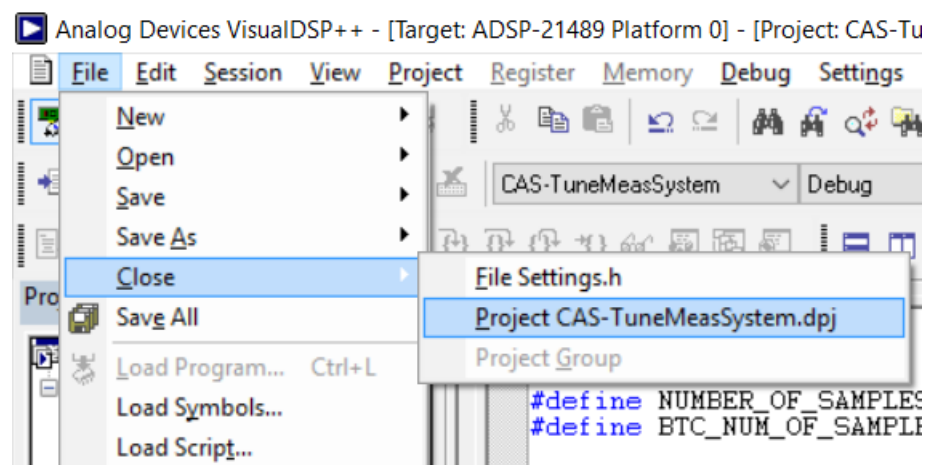
A classical approach for such dynamic tune measurements is continuous noise excitation and consecutive Fourier transforms (FFT) of the measured turn by turn position data.

With so called “peak finding algorithms” the tune is identified in each measurement as the maximum of the FFT amplitude spectrum and hence the evolution of the tune can be followed. Measurement resolution in tune and time resolution between two measurement points have always to be traded against each other.

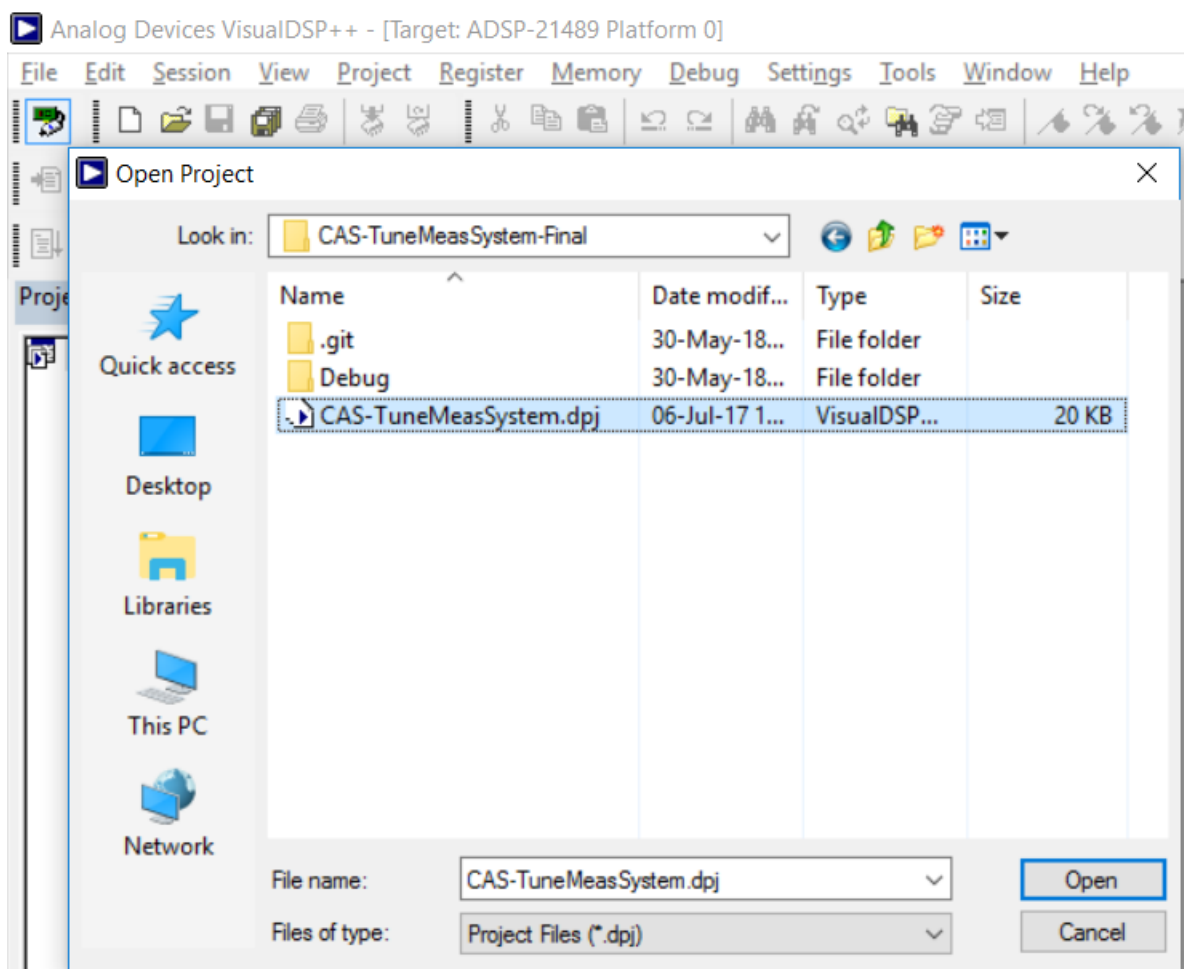
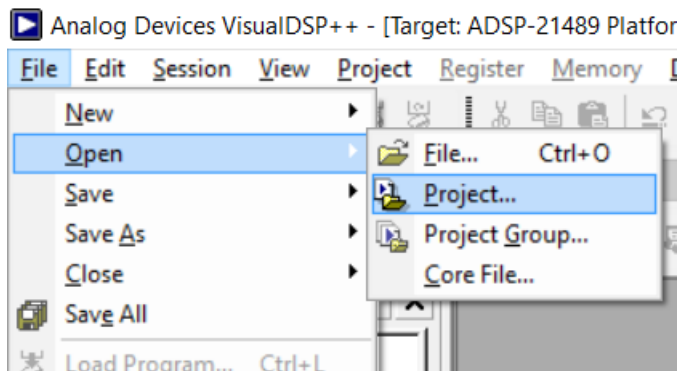
A more elegant way of tracking the evolution of the tune is by having an electronic circuit (analog or digital), which excites continuously the beam to transverse oscillations and changes automatically the frequency of the beam excitation, such that excitation and beam response stay in phase. So the circuit locks itself to a stable excitation phase and it is hence called a “Phase locked loop”.

In contrary to the FFT method indicated above, the PLL uses the phase information as indicator for resonance, the FFT methods look for a maximum in amplitude.

For the experiment with the DSP please close the previous project:



Open new project:



The DSP is programmed to execute a PLL circuit. It generates a continuous sine wave oscillation and it measures permanently the phase between its excitation signal and the stimulated oscillation.

You may have a look at the code in particular in order to see how the phase detection is made. But otherwise the only parameter that you need to care about is the start frequency of the oscillator (not too far from the expected resonance) and a loop gain for the PLL.

Settings overview in the file settings.h:

Analog Devices VisualDSP++ - [Target: ADSP-21489 Platform 0] - [Project: CAS-TuneMeasSystem] - [Settings.h]

```

File Edit Session View Project Register Memory Debug Settings Tools Window Help

CAS-TuneMeasSystem Debug

Project: CAS-TuneMeasSystem...
Project Group (1 project)
CAS-TuneMeasSystem
Source Files
blockProcess_audio.c
init1939viaSPI.c
init_PLL_SDRAM.c
initSPDIF.c
initSPORT01_TDM_moi
initSRU.c
Main.c
SPORT1_isr.c
Linker Files
Header Files
ad1939.h
ADDS_21489_EzKit.h
btc.h
btc_struct.h
Settings.h
README.txt

#ifdef _SETTINGS_H
#define _SETTINGS_H

#define NUMBER_OF_SAMPLES 1 // Number of samples handles per IRQ (
#define BTC_NUM_OF_SAMPLES 200 // Number of samples in BTC channel

// SETTINGS - START
#define EXCITATION_AMPLITUDE 0.99 // excitation amplitude: [-1.0f,
// #define EXCITATION_AMPLITUDE 0.6 excitation amplitude: [-1.0f,1

#define PLL_GAIN 1.0 // PLL gain -> to be used with old beam

#define START_FREQ_HZ 1400.0 // PLL starting frequency in Hz -> to
// SETTINGS - END

#define TRIG_TABLE_LEN 1024 // MUST BE A POWER OF 2!!!
extern long double PI;
extern long double TWOPI;
extern long double cosTable[TRIG_TABLE_LEN];
extern long double sinTable[TRIG_TABLE_LEN];
extern long double trigTableLenOverTwoPi;
extern long double twoPiOverTrigTableLen;
extern void buildTrigTables(void);

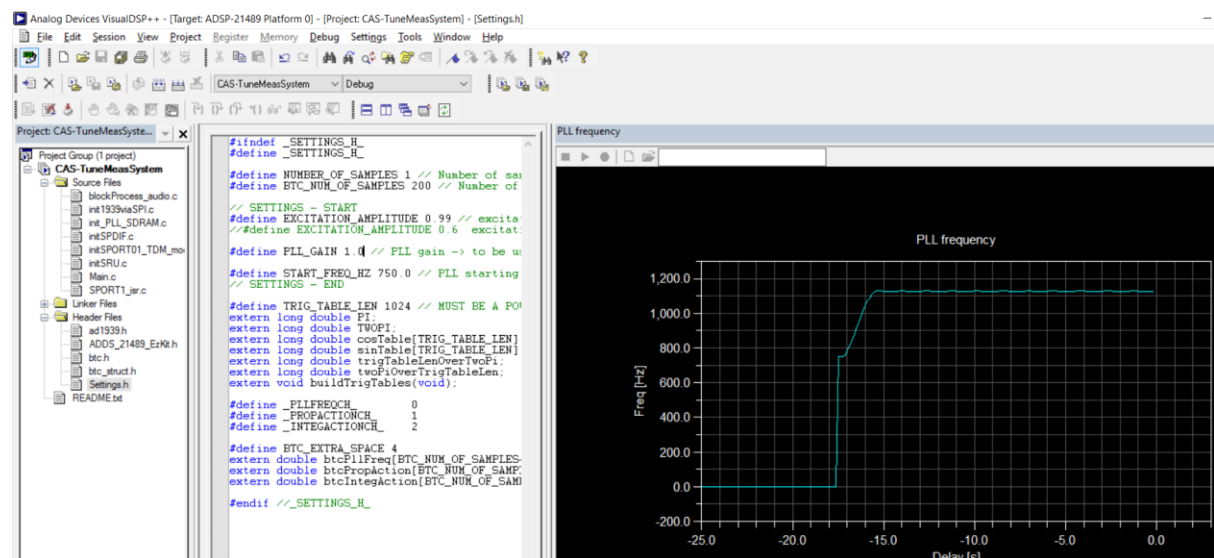
#define _PLLFREQCH_ 0
#define _PROPACCTIONCH_ 1
#define _INTEGACCTIONCH_ 2

#define BTC_EXTRA_SPACE 4
extern double btcPllFreq[BTC_NUM_OF_SAMPLES+BTC_EXTRA_SPACE];
extern double btcPropAction[BTC_NUM_OF_SAMPLES+BTC_EXTRA_SPACE];
extern double btcIntegAction[BTC_NUM_OF_SAMPLES+BTC_EXTRA_SPACE];

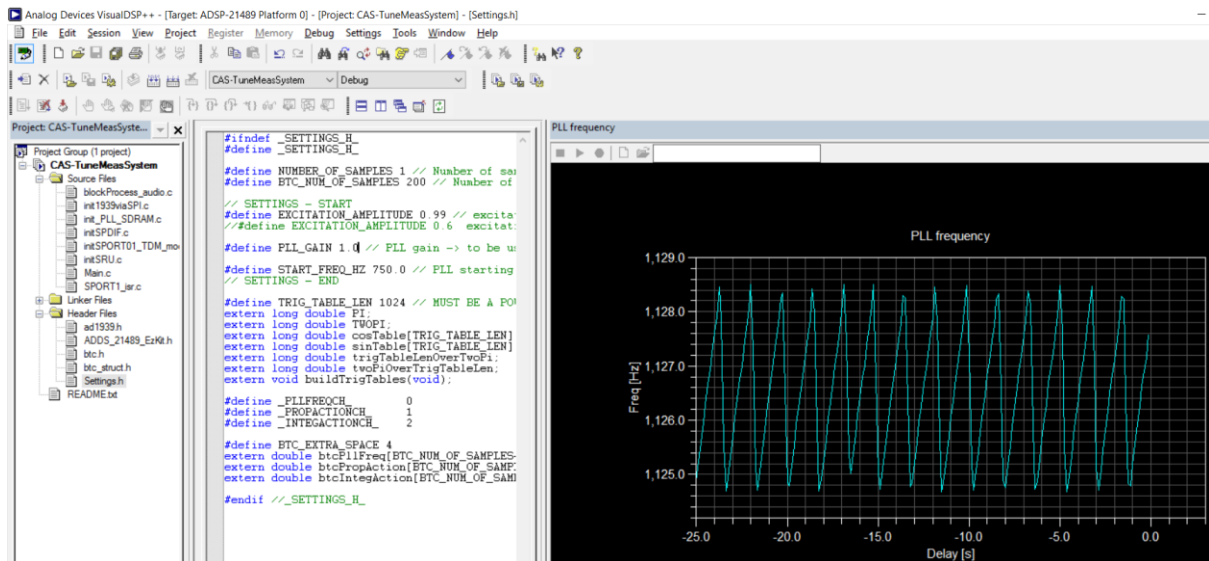
#endif // SETTINGS_H

```

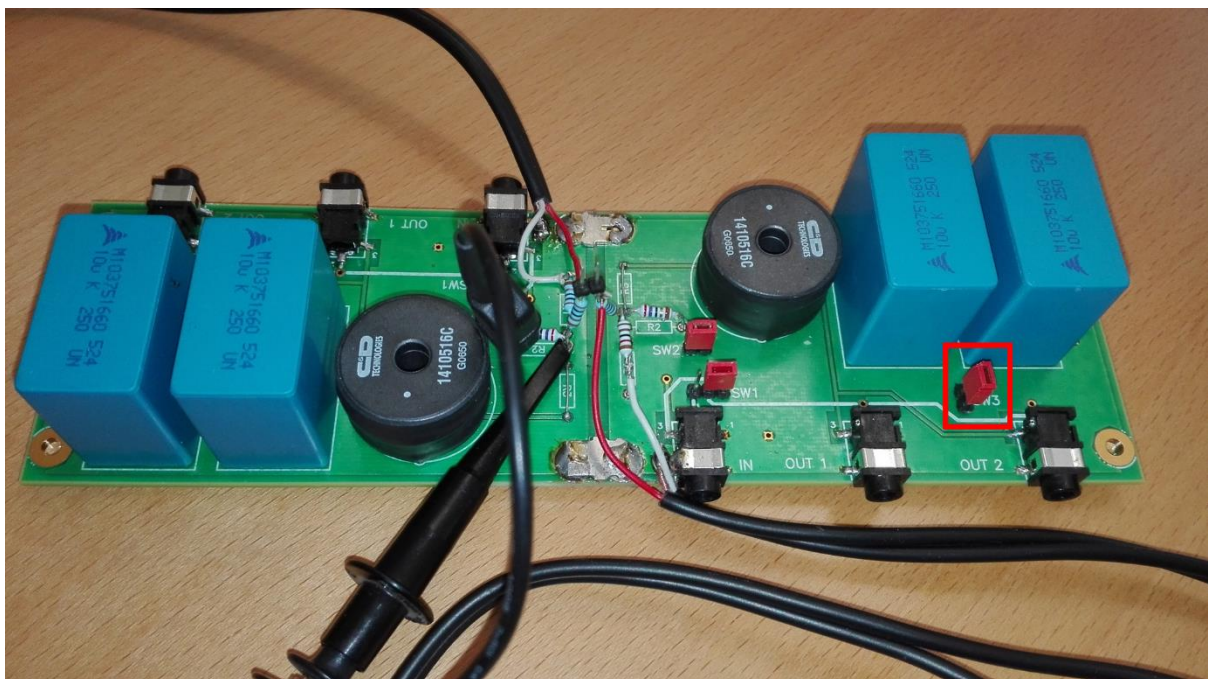
Set starting frequency and PLL gain. Compile, load and start (PLL starts at 750Hz and converges to resonant frequency):



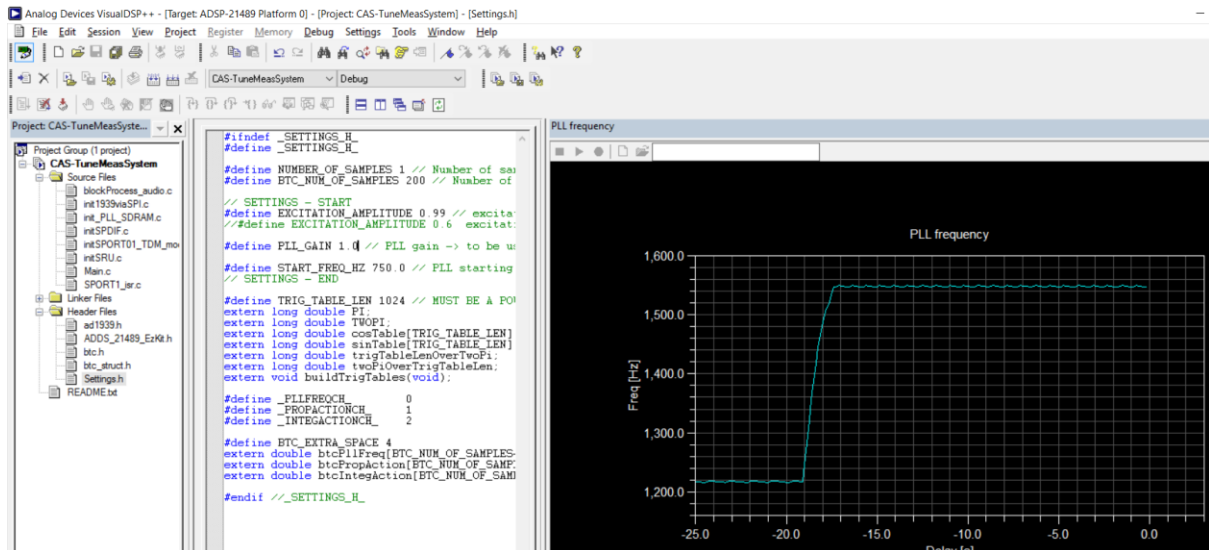
After locking, PLL frequency oscillates with small excursions around the resonant frequency:



Change (increase) resonant frequency by removing jumper SW3:



PLL tracks new resonant frequency:



You can experiment with the start frequency and the loop gain and observe the reactivity of the PLL circuit.

At any time you may look at the signals with the oscilloscope.