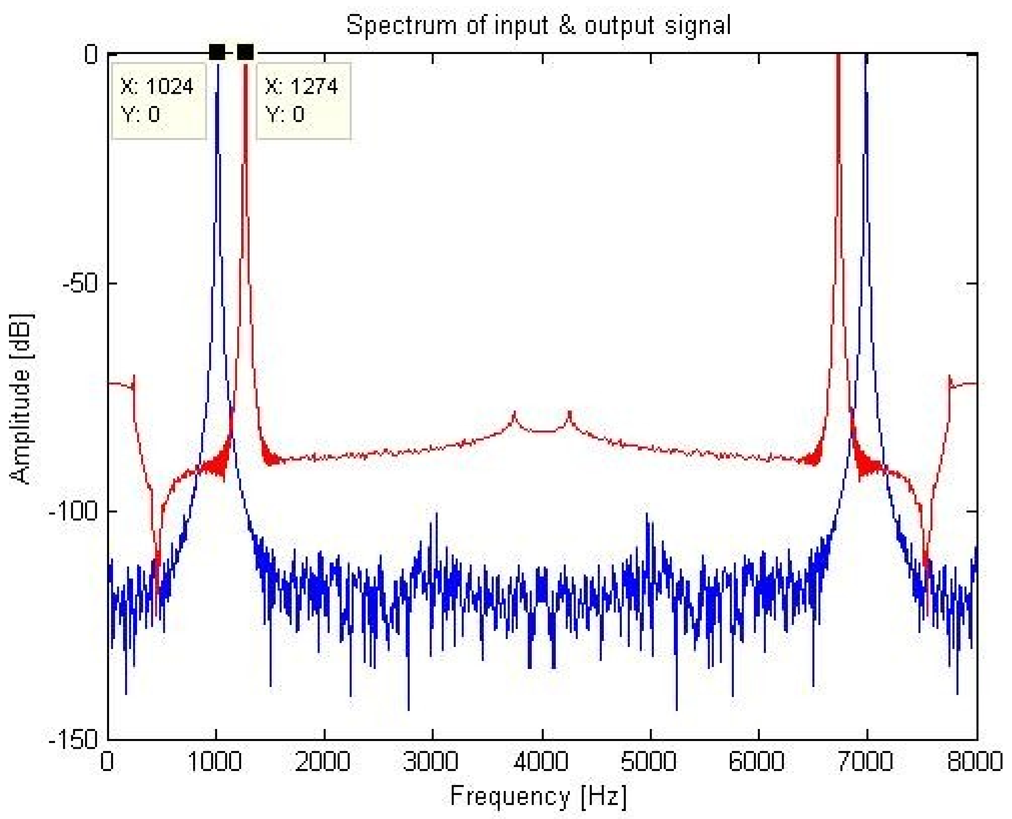
Example: Frequency shifter

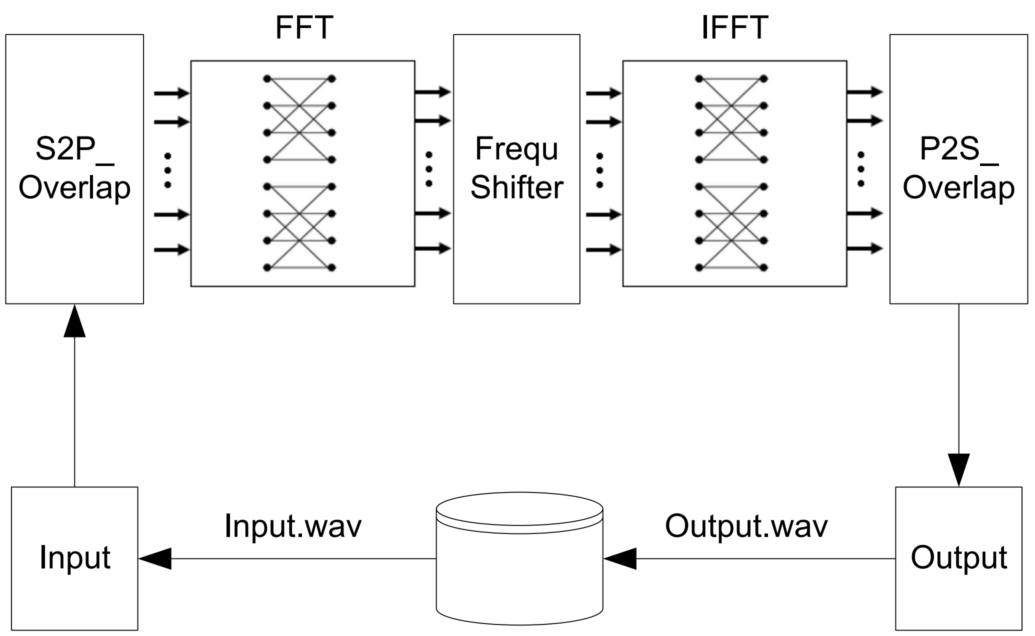
The frequency shift example reads an input Wave file, shifts the contained frequencies and writes the results to another Wave file. The shift operation is performed in the frequency domain. To convert the signal to the frequency domain and to convert it back after shifting, the library element FFT/IFFT is used.

Applied to a simple Sine Wave, the frequency of the sine signal is changed according to the Shift parameter. Unlike by simply increasing the playback speed, the length of the audio signal is not changed. The following figure shows the spectrum of an input signal and the spectrum of the shifted signal.



The FFT length has been set to 1024 points and the Shift factor is set to 32. With a sampling frequency of 8 kHz, the distance between two frequency points is 8000/1024 = 7.8125 Hz. The signal is shifted by 32 frequency points, which results in a frequency shift of 250 Hz. The input Sine(blue) with a frequency of 1024 Hz has been shifted to 1274 Hz (red). Apparently, the overall noise is increased. Since this example should serve as a simple application of the ANDRES library components no additional investigations have been made to improve the quality of the of the output signal.

The following figure shows the module structure of the frequency shifter system. The input file is read by the Input module. The S2P_Overlap module converts the signal to a parallel sample vector and applies a Hamming window. The resulting sample vector is then transformed to the frequency domain by the FFT/IFFT module. In the frequency domain, the actual frequency shift is performed and the result is converted back by another FFT/IFFT module, configured as inverse FFT. By the overlap and add method, the calculated frames are set together to build the output signal, which is written to the specified output file.



The provided project allows the following command line parameters:

Int Shift	sets the shifting distance in the FrequShifter module
char* pInputFile	sets the path to the input file
char* pOutputFile	sets the path to the output file

1. Input

Reads specified input file and writes single samples to the connected channel.

Module definition: Input(sc_module_name n, char* pFile)
Module interfaces: sca_tdf::sca_out<double> out;

Name	Type	Default value	Description
nm	sc_module_name	-	
pFile	char*	-	path to input file

2. S2P_Overlap

Serial to parallel converter with adjustable overlap. Imaginary part is constantly set to zero. Rate of in port is set to (N-Overlap). Each parallel data vector is multiplied with a Hamming window to prepare the vector for the FFT. This module stops the simulation, if all samples of the input file have been read.

Module definition: template <int N, int Overlap>
S2P(sc_module_name n)
Module interface: sca_tdf::sca_in<double> in;

sca_tdf::sca_out<double> out_real[N];
sca_tdf::sca_out<double> out_imag[N];

Name	Type	Default value	Description
nm	sc_module_name	-	

3. FrequShifter

As input an spectrum with N frequency points is expected. The parameter Shift specifies the number of points the spectrum is shifted. Empty points are filled with zeros. A negative Shift value moves the to a lower frequency range.

Module definition: template <int N>
FrequShifter(sc_module_name n, int Shift)

Module interface: sca_tdf::sca_in<double> in_real[N];
sca_tdf::sca_in<double> in_imag[N];
sca_tdf::sca_out<double> out_real[N];
sca_tdf::sca_out<double> out_imag[N];

Name	Type	Default value	Description
nm	sc_module_name	-	
Shift	int	-	specifies shift distance. Has to be in the interval [-N/2;N/2]

4. P2S_Overlap

Performs the overlap and add method on the vectors provided by the IFFT unit. The rate of the output is set to (N-Overlap).

Module definition: template <int N, int Overlap>
P2S_Overlap(sc_module_name n)

Module interface: sca_tdf::sca_in<double> in_real[N];
sca_tdf::sca_in<double> in_imag[N];
sca_tdf::sca_out<double> out;

Name	Type	Default value	Description
nm	sc_module_name	-	

5. Output

Writes single samples to the specified file. If it does not exist, a new file is generated. At the end of the simulation, when the object is destroyed, the Wave file is finished (several length fields are written).

Module definition: Output(sc_module_name n, char* pFile)
Module interface: sca_tdf::sca_in<double> in;

Name	Type	Default value	Description
nm	sc_module_name	-	
pFile	char*	-	path to ouput file