Digital Filter Designer

The Digital Filter Designer contains a toolset to specify, analyse, validate digital filters and to generate C and C++ source code of the designed filter.

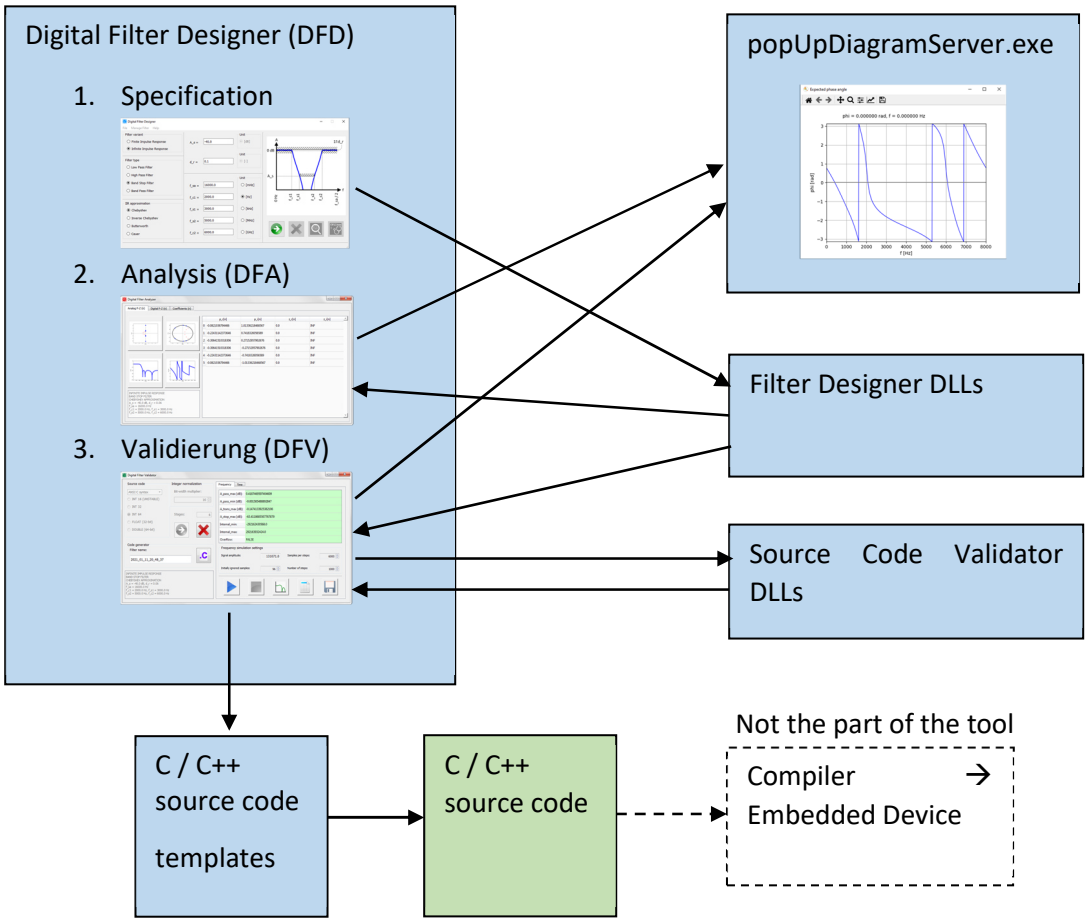


FIGURE 1: BLOCK DIAGRAM OF THE DIGITAL FILTER DESIGNER

The toolset contains 3 graphical user interface elements and an external *popUpDiagramServer.exe* tool for the visualisation of the diagrams. At the same time only one instance of the *dfdGui.exe* can be started.

After starting the dfdGui.exe the main window of Digital Filter Designer (DFD) is shown. The generated filters can be analysed by the Digital Filter Analyser (DFA) and validated by the Digital Filter Validator (DFV) GUIs.

Once a filter is designed, it can be saved with ".DFD " extension, which can be loaded by the application for further use.

To design the filters the DFD is extended by the *FIR_designer.dll* and *IIR_designer.dll* files. These files must be stored in the *_30_dllDriversDesigner* subfolder.

The source code validation occurs by the following external DLLs: *FIR_validator_c.dll, FIR_validator_cpp_98.dll, IIR_validator_c.dll* and *IIR_validator_cpp_98.dll*. These files must be stored under the _31_dllDriversValidator subfolder.

A source code generator with the default source code template files is also the part of the tool, which makes possible to generate C and C++ source code files for further applications.

1. Digital Filter Designer GUI

1.1. Settings Window of the DFD

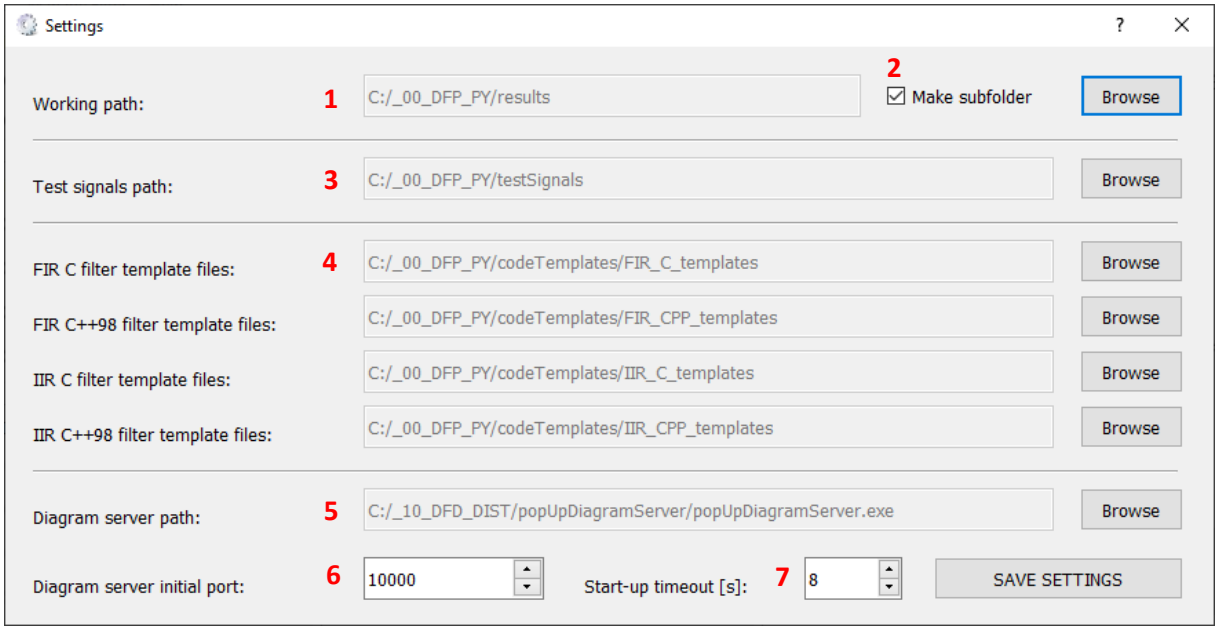


FIGURE 2: SETTINGS WINDOW OF THE DIGITAL FILTER DESIGNER

The settings window of the tool can be reached under the "*File* → *Settings*" point of the menu bar. The following parameters can be configured:

- **[1]** Working path: standard working path of the filter designer. If the checkbox **[2]** "Make subfolder" element is checked, every single working path is going to be extended with a subfolder according to the UTC timestamp of the filter creation time, as: *YYYY_MM_DD_hh_mm_ss*, where: *YYYY* the year value as the filter was created, *MM*: the month value as the filter was created, *DD*: the day value as the filter was created, *hh*: the hour value as the filter was created, mm: the minute value as the filter was created, *ss*: the second value as the filter was created. The path can be set by a file dialog after clicking on the browse button.
- **[3]** Test signals path: for the time-based simulation *.CSV* files can be used with the data points. The default path of these files can be set under this point. The path can be set by a file dialog after clicking on the browse button.
- **[4]** Source code template files: the tool makes possible for the user to apply own version of the pattern files. The path of the pattern files can be set by a file dialog after clicking on the browse button.
- Diagram server settings:
  - **[5]** Diagram server path: to visualise the different diagrams, an external *popUpDiagramServer.exe* tool must be used. The path can be set by a file dialog after clicking on the browse button.

- **[6]** Diagram server initial port: the *dfdGui.exe* and the *popUpDiagramServer.exe* applications are communicating each other by socket communication. The initial port is the applied port value of the first instance of the diagram server. The port value is incremented at the opening of another instance of the diagram server. For a successful application of the diagram server, it must be guaranteed, the applied ports are available for the localhost communication and not blocked by the firewall.
- **[7]** Start-up timeout: at the request of a new diagram instance a subprocess of a *popUpDiagramServer.exe* is started and the *dfdGui.exe* is operating in this case as a client. The value of this parameter is a waiting time in seconds. Within this time, it must be guaranteed, the server subprocess is running and ready to receive the communication from the client-side. In case of "*WinError 10061*" this timeout value must be extended to grant the readiness of the server-side communication.

By clicking onto the "*SAVE SETTINGS*" button, the settings are stored on the hard drive, and going to be loaded automatically after starting the *dfdGui.exe* application.

1.2 Diagram Server

To guarantee a flexible visualisation of the relevant data, the diagram server applications (popUpDiagramServer.exe) must be started in separated subprocesses. After closing any window or changing the equivalent GUI state, all the belonging subprocesses will be closed as well. After requesting any diagram another instance of the subprocess will be started. After clicking on any open diagram button, it may take up to one minute until the visualisation of the diagram.

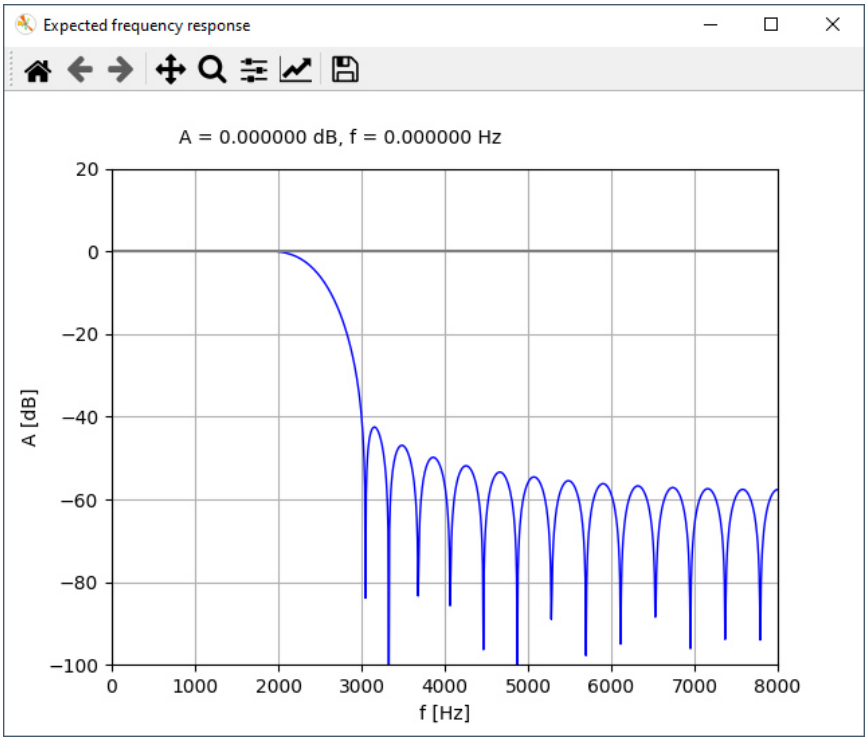


FIGURE 3: AN EXAMPLE OF DATA VISUALISATION BY THE DIAGRAM SERVER

The diagram server offers the following services:

- Zoom in / out opportunity
- Diagram cursor, which can be frozen / unfrozen by a mouse left click
- The diagram view can be saved as a .PNG file

1.3 Filter Specification

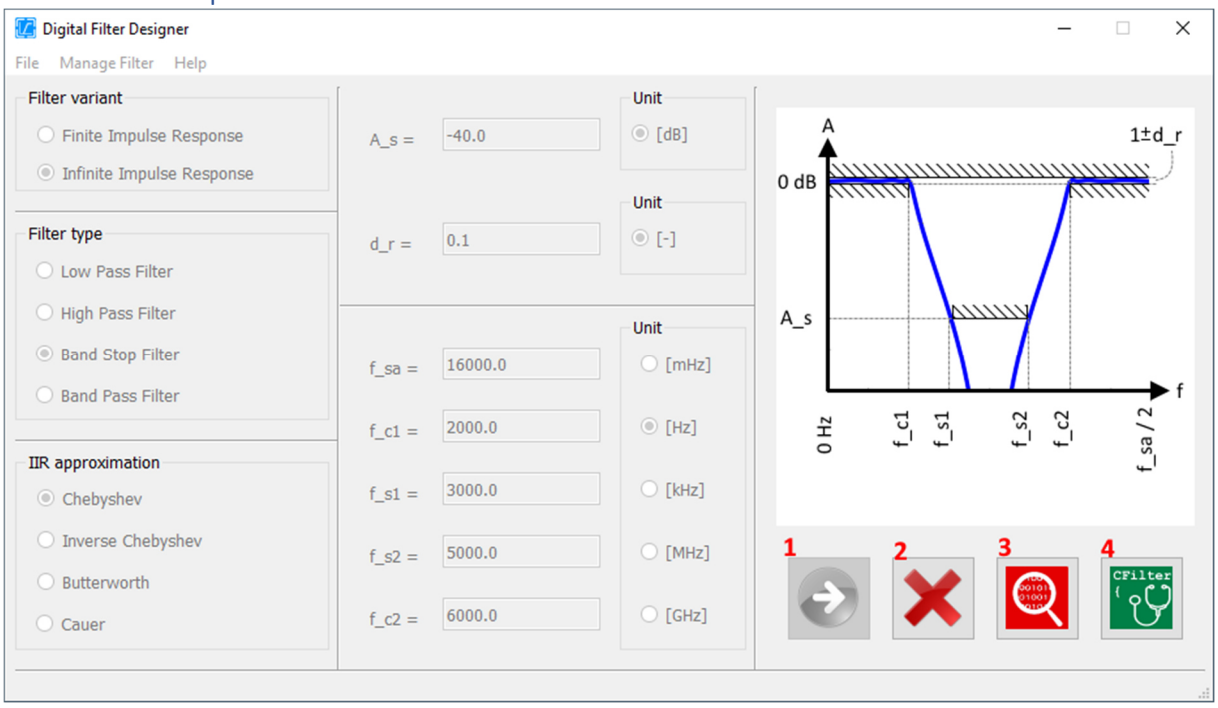


FIGURE 4: MAIN WINDOW OF THE DIGITAL FILTER DESIGNER

After starting the *dfdGui.exe* application the main window is shown, where the parameters of the digital filter can be set. According to the Nyquist sampling theorem, the frequency behaviour in the frequency band between 0 Hz and the half the sampling frequency can be specified.

The goal of the DFD designer algorithm to grant the following parameters:

- In case of FIR filters: the designer cannot handler the “d_r” parameter for FIR filter, but due to the structure of the FIR filters, the pass-band attenuation approximates 0 dB
- In case of IIR filters: “d_r” is the maximal pass-band ripple, where $A_r[dB] = 20 \cdot log_{10}(1 \pm d_r)$
- “A_s” is the minimal attenuation in stop band

After specifying the filter parameters:

- **[1]** Click on the “Create filter” button or press “F5” to create the specified filter
- **[2]** Click on the “Discard filter” button or press “F6” to discard an earlier created filter
- **[3]** Once a filter is created, click on the “Analyse current filter” button or press “F7” to open the DFA GUI
- **[4]** Once a filter is created, click on the “Generate Code and Validate” button or press “F8” to open the DFV GUI

2. Digital Filter Analyser

2.1. Analysis of FIR Filters

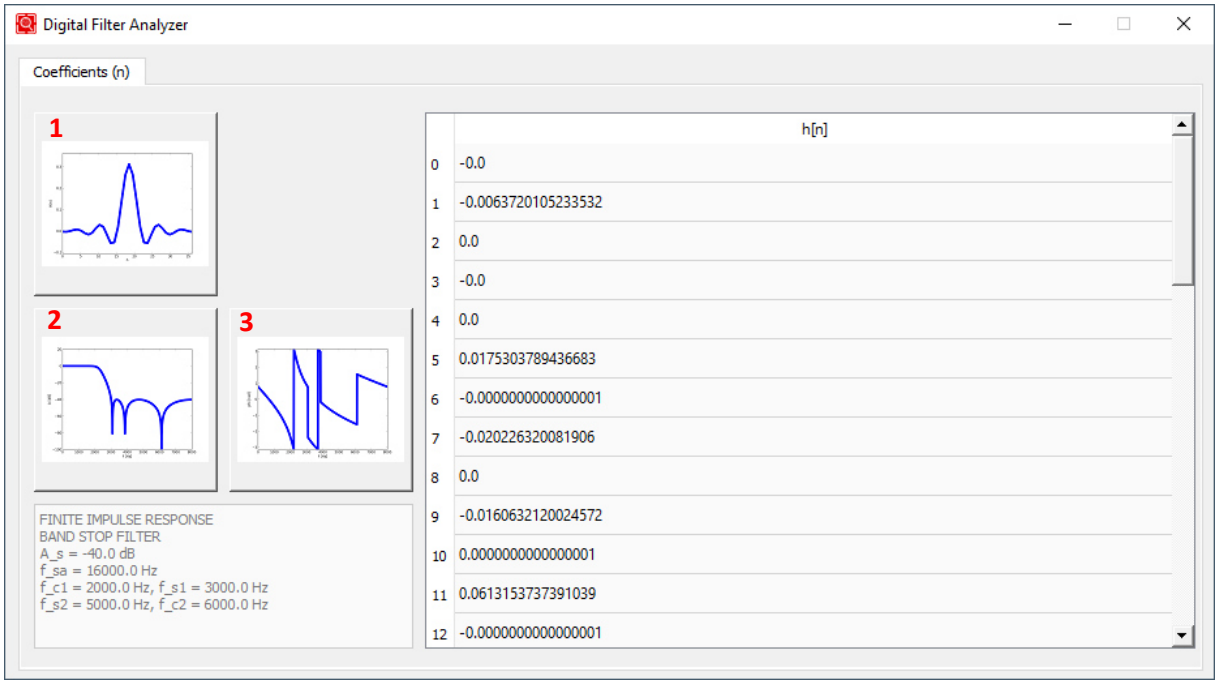


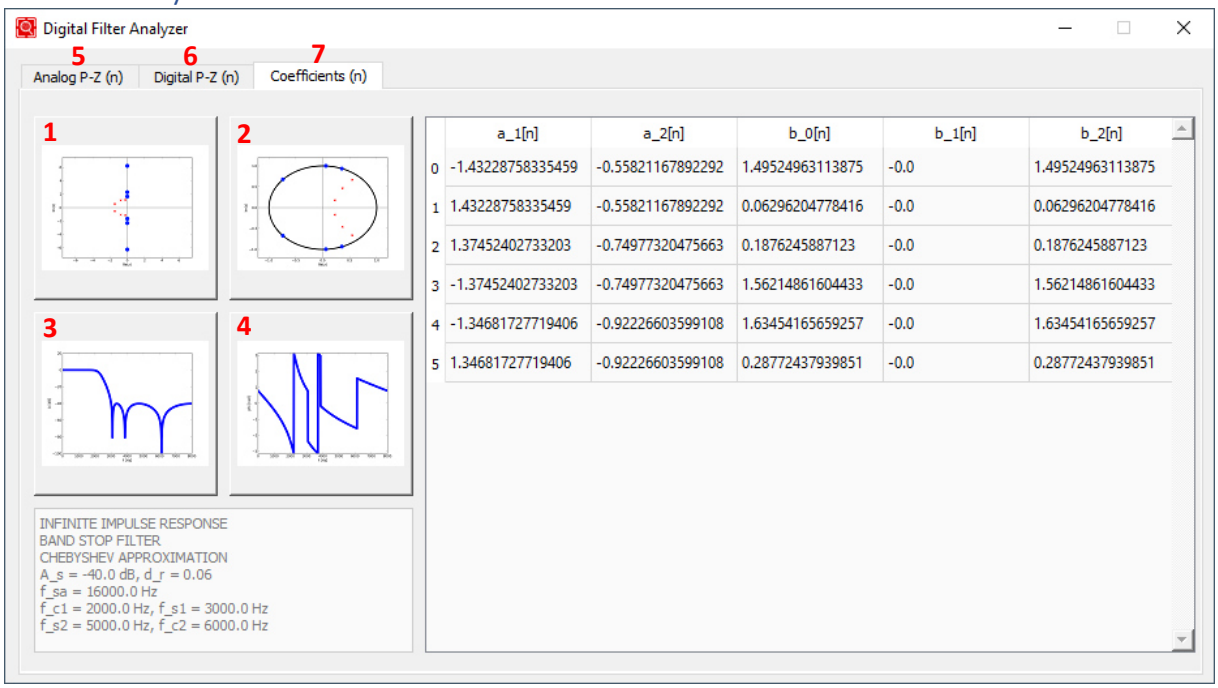
FIGURE 5: ANALYSIS WINDOW OF A FIR FILTER

The analysis window of a FIR filter makes possible to visualize the following diagrams, by clicking on the equivalent buttons (with the notations of Figure 5):

- **[1]** Expected impulse response diagram or press “F5”
- **[2]** Expected frequency response diagram or press “F7”. This diagram is calculated from to the coefficient values. **The behaviour of the real filter cannot be guaranteed according to these calculations!**
- **[3]** Expected phase angle diagram or press “F8”. This diagram is calculated from to the coefficient values. **The behaviour of the real filter cannot be guaranteed according to these calculations!**

A tabular view of the filter coefficients is also shown on the GUI. **These values are rounded values, thus must not be used for further calculations!** The source code generator delivers these coefficients in the requested data type with the best accuracy.

2.2. Analysis of IIR Filters



	a_1[n]	a_2[n]	b_0[n]	b_1[n]	b_2[n]
0	-1.43228758335459	-0.55821167892292	1.49524963113875	-0.0	1.49524963113875
1	1.43228758335459	-0.55821167892292	0.06296204778416	-0.0	0.06296204778416
2	1.37452402733203	-0.74977320475663	0.1876245887123	-0.0	0.1876245887123
3	-1.37452402733203	-0.74977320475663	1.56214861604433	-0.0	1.56214861604433
4	-1.34681727719406	-0.92226603599108	1.63454165659257	-0.0	1.63454165659257
5	1.34681727719406	-0.92226603599108	0.28772437939851	-0.0	0.28772437939851

FIGURE 6: ANALYSIS WINDOW OF AN IIR FILTER

The analysis window of a IIR filter makes possible to visualize the following diagrams, by clicking on the equivalent buttons (with the notations of Figure 6):

- **[1]** Analog pole zero diagram or press “F5”
- **[2]** Digital pole zero diagram or press “F6”
- **[3]** Expected frequency response diagram or press “F7”. This diagram is calculated from to the coefficient values. **The behaviour of the real filter cannot be guaranteed according to these calculations!**
- **[4]** Expected phase angle diagram or press “F8”. This diagram is calculated from to the coefficient values. **The behaviour of the real filter cannot be guaranteed according to these calculations!**

By selecting the equivalent tabs the following tabular views are shown on the GUI:

- **[5]** Analog P-Z (n) to show the analog pole-zero pairs
- **[6]** Digital P-Z (n) to show the digital pole-zero pairs
- **[7]** Coefficients (n) to show the IIR filter coefficients

These values are rounded values, thus must not be used for further calculations! The source code generator delivers these coefficients in the requested data type with the best accuracy.

3. Digital Filter Validator GUI

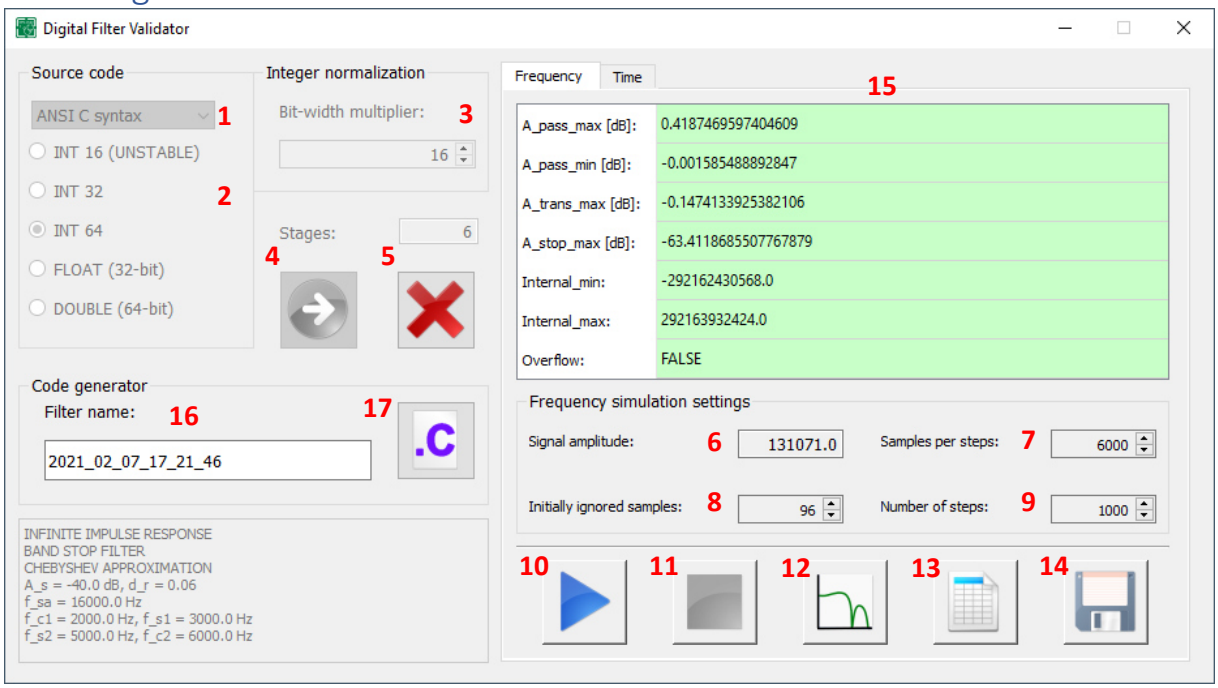


FIGURE 7: VALIDATOR WINDOW AFTER A FREQUENCY-BASED VALIDATION

Prior to the source code generation, the following parameters must be set (with the notations of Figure 7):

- **[1]** Source code language syntax: the language of the generated source code
  - ANSI C syntax
  - C++ 98 syntax
- **[2]** Source code data type:
  - INT 16: 16-bit integer (in most of the cases does not meet the specifications!)
  - INT 32: 32-bit integer
  - INT 64: 64-bit integer
  - FLOAT: 32-bit floating point
  - DOUBLE: 64-bit floating point
- **[3]** Bit-width multiplier (only in case of integer filters):
  The coefficients of the filters are calculated internally with 64-floating point (double) accuracy. For integer-based filters the coefficients must be multiplied according to:
  $COEFF_x = coeff_x \cdot 2.0^n$, where:
  - $COEFF_x$: any integer-based coefficient,
  - $coeff_x$: any internally calculated (double-based) coefficient,
  - n: is the "Bit-width multiplier" parameter.

  The value of the multiplier must be set according to the data type of the filter and the data type of the input values.
  - The less the value of the "Bit-width multiplier" the lower accuracy of the filter can be expected.
  - The higher the value of the "Bit-width multiplier" overflow of the filter operation may occur.
- **[4]** Create source code button or press "F5"
- **[5]** Discard source code button or press "F6"

3.1. Frequency-based Simulation:

As follows the phrase “sample” is being used for a data point in time domain and the phrase “step” for a data point in frequency domain. For a frequency-based simulation the following parameters must be set (with the notations of the Figure 7):

- **[6]** Signal amplitude: amplitude of the input signal (see Figure 8 below). In case of integer-based filters higher amplitude values may cause overflow, the lower values may cause uncertain operation.
- **[7]** Samples per steps: the number of time-based datapoints, used for the calculation of a single frequency step (see Figure 8 and Figure 9 below). The higher this value, the longer simulation time must be expected. The lower this value, the more likely your filter does not meet the specification.
- **[8]** Initially ignored samples: due to transient behaviour – especially in case of IIR filters – the first samples cannot be used – thus must be ignored – for an accurate frequency-based simulation (see Figure 8 and Figure 9 below). The higher its value, the longer time the simulation takes. The lower its value the more likely your filter does not meet the specifications.
- **[9]** Number of steps: number of the data points of the frequency-based simulation between 0 Hz, and the half of sampling frequency (see Figure 10 below). The higher this value, the longer simulation and visualisation time must be expected. The lower this value, the less information is delivered by the simulation.

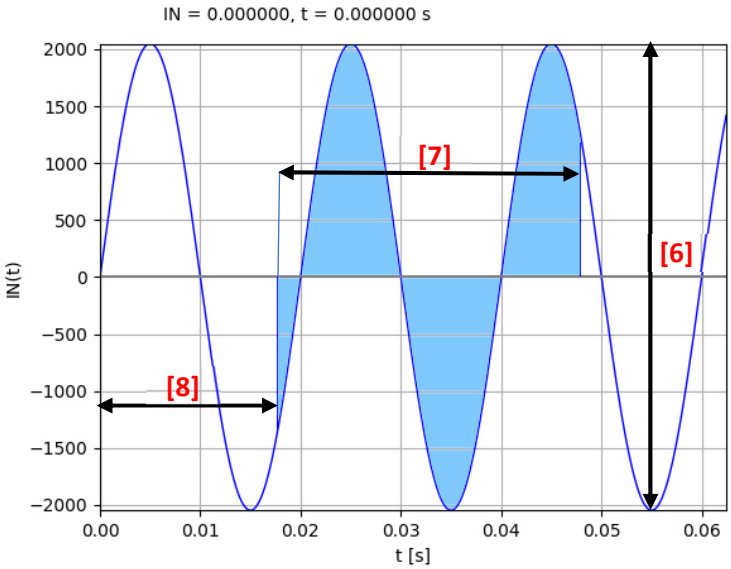


FIGURE 8: INPUT SIGNAL

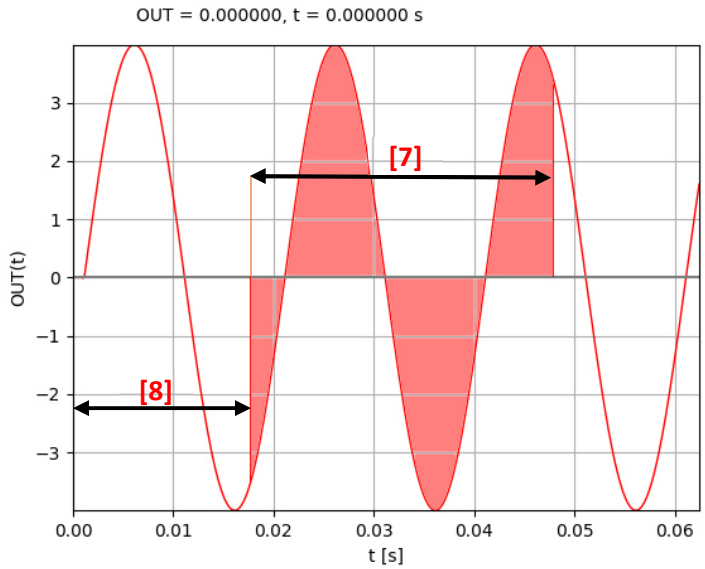


FIGURE 9: OUTPUT SIGNAL

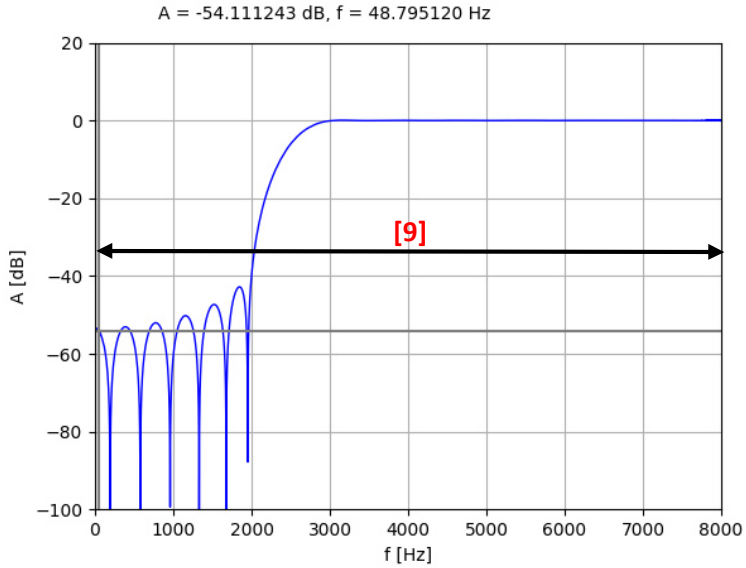


FIGURE 10: FREQUENCY RESPONSE OF A HIGH PASS FILTER

Thus, a data point of the frequency response (see Figure 10) can be calculated, as:

$$A(f) = 20 \cdot log_{10} \frac{\sum_{t=iis}^{(iis+sps)} |OUT(t)|}{\sum_{t=iis}^{(iis+sps)} |IN(t)|}$$

Where:

- f: frequency of the input sinusoidal signal,
- sps: value of "Samples per steps", marked by **[7]** on Figure 8 and Figure 9,
- iis: value of "Initially ignored samples", marked by **[8]** on Figure 8 and Figure 9,
- $\sum_{t=iis}^{(iis+sps)} |IN(t)|$: the absolute value of the blue marked surface on Figure 8,
- $\sum_{t=iis}^{(iis+sps)} |OUT(t)|$: the absolute value of the red marked surface on Figure 9.

Further elements of the validator GUI, with the notations of Figure 7 on Page 7:

- **[10]** Start of a frequency-based simulation run. The simulation run may take up to a few minutes. The higher values of the "Samples per steps", "Initially ignored steps" and "Number of steps" parameters may result longer simulation time.
- **[11]** Immediate stop of a running simulation
- **[12]** Show the simulated frequency response diagram. The opening and data transfer with the diagram server may take a few minutes. The higher value of the "Number of steps" parameter may result longer data transfer time.
- **[13]** Show the results of the frequency-based simulation as a tabular view
- **[14]** Save the results of the frequency-based simulation into a .CSV
- **[15]** Tabular view about the evaluation of the filter simulation results. The filter evaluation criteria are the following:
    - In case of FIR filter: the designer cannot handler the "d_r" parameter for FIR filter, but due to the structure of the FIR filters, the pass-band attenuation approximates 0 dB
    - Maximal pass band attenuation (IIR only): $\left|max\left(A_{pass}\right)\right| \leq |20 \cdot log_{10}(1 + d_r)|$,
    - Minimal pass band attenuation (IIR only): $\left|min\left(A_{pass}\right)\right| \leq |20 \cdot log_{10}(1 - d_r)|$,
    - Transient band validation: $max(A_{transient}) \leq max\left(A_{pass}\right)$,
    - Stop band validation: $max\left(A_{stop}\right) \leq A_s$,
    - Datatype range criteria: the internal limit values must fit to the selected datatype,
    - Overflow validation (not for double data type): if any internal operation results a numerical value which exceeds the limits of the applied data type

The d_r and A_s are the delta-ripple and stop-band attenuation parameters, as specified filter parameters, see Figure 4 on Page 4.

All the fulfilled criteria marked by green background colour, the failed criteria are marked by red. **The delivered results are based solely on a simulation! This simulation does not substitute the necessary testing steps on your target device!**

After the validation of a filter, the source code can be generated. The relevant elements of the validator GUI – with the notations of Figure 7 on Page 7 – are the following:

- **[16]** Filter name: the default name it is the UTC timestamp of the filter creation time, as: *YYYY_MM_DD_hh_mm_ss*, where: *YYYY* the year value as the filter was created, *MM*: the month value as the filter was created, *DD*: the day value as the filter was created, *hh*: the hour value as the filter was created, mm: the minute value as the filter was created, *ss*: the second value as the filter was created. The user has the opportunity to apply an arbitrary filter name, which must contain only English alphabetic characters or numbers (the name is the part of the class name and the file name; thus, it must be a valid substring of them).
- **[17]** Save source code button (Ctrl + S): by pressing this button the source code files are going to be generated under the path, which is set as "Working path" under the settings (see Figure 2 on Page 2). In case of already existing files, the tool requests the user's permission to overwrite.

The name of the source code files is created as follows:

- <FILTER_NAME>_<FILTER_TYPE><DATA_TYPE><SOURCE_CODE_LANGUAGE>, where:
  - FILTER_NAME: element **[16]** of Figure 7 on Page 7
  - FILTER_TYPE: “fir” or “iir”
  - DATA_TYPE: “i16” (INT 16), “i32” (INT32), “i64” (INT64), “f32” (FLOAT), “f64” (DOUBLE)
  - SOURCE_CODE_LANGUAGE: c (ANSI C), cpp98 (C++ 98)

The name of the filter is applied within the source code as well, as in the name of the classes or in the function names:

- C<FILTER_NAME>_<FILTER_TYPE><DATA_TYPE><SOURCE_CODE_LANGUAGE>, where:
  - FILTER_NAME: element **[16]** of Figure 7 on Page 7
  - FILTER_TYPE: “Fir” or “Iir”
  - DATA_TYPE: “I16” (INT 16), “I32” (INT32), “I64” (INT64), “F32” (FLOAT), “F64” (DOUBLE)
  - SOURCE_CODE_LANGUAGE: C (ANSI C), CPP98 (C++ 98)

Further information about the application of the generated source code can be read in chapter “4. Application of the Generated Source Code” on Page 12.

3.2 Time-based Simulation

Next to the frequency-based simulation, the tool makes possible to analyse the behaviour of a digital filter in time domain.

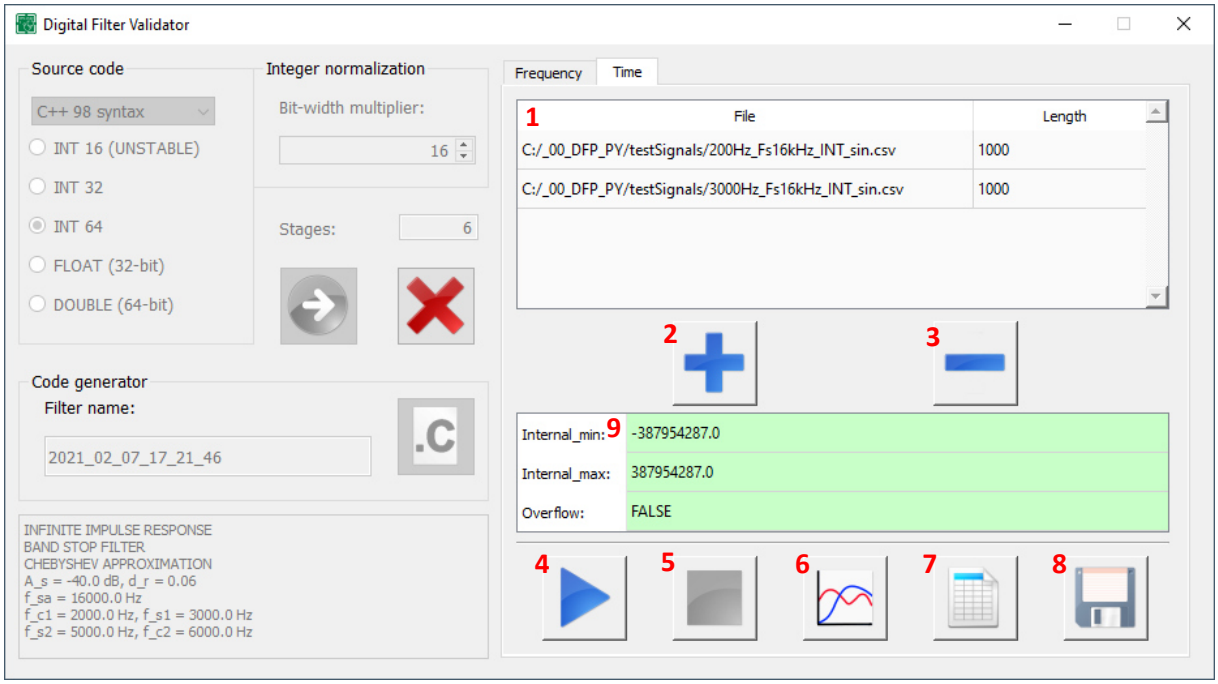


FIGURE 11: VALIDATOR WINDOW AFTER A TIME-BASED VALIDATION

- For the time-based simulations, the input signal or signals are listed in the table view **[1]**. The input signals files must be stored with .CSV extension and the files have to contain the data points in a column vector. The storage format can be either floating point, or integer.
- With the “+” button **[2]** another time signal can be added, thus the input signal for each data point can be calculated as:

$$x(n) = \sum_{i=0}^{i=M} x_i(n)$$

- With the "-" button **[3]** all signals can be deleted from the input signal list
- **[4]** Start of a time-based simulation run
- **[5]** Immediate stop of a running simulation
- **[6]** Show the input and output signal diagram. The opening and data transfer with the diagram server may take a few minutes.
- **[7]** Show the results of the time-based simulation as a tabular view
- **[8]** Save the results of the time-based simulation into a .CSV
- **[9]** Tabular summary of the limit and overflow evaluation

The delivered results are based solely on a simulation! This simulation does not substitute the necessary testing steps on your target device!

4. Application of the Generated Source Code Files

4.1 FIR ANSI C Source Code Generation

For each FIR ANSI C filter 4 files are generated under the path, which is set under **[1]** on Figure 2 on Page 2:

- **fir<XXX>c.c**: general code implementation of a filter, one instance of this file can handle any amount of particular filter instances with the given data type
- **fir<XXX>c.h**: the header of the general implementation file, one instance of this file can handle any amount of particular filter instances with the given data type
- **<FILTER_NAME>_fir<XXX>c.c**: the code implementation of the designed filter
- **<FILTER_NAME>_fir<XXX>c.h**: the header file of the implemented filter

Where XXX is the applied data type, the data types are:

- F64: 64-bit floating point
- F32: 32-bit floating point
- I64: 64-bit signed integer
- I32: 32-bit signed integer
- I16: 16-bit signed integer

```
// Steps, how to use the generated files

// 1. Include the header files
#include "FILTER_NAME_firi64c.h"
#include "firi64c.h"

// 2. Make a filter instance
struct CFirI64CData l_myFilter;

// 3. Initialize the filter
CFILTER_NAMEFirI64C_initFilter(&l_myFilter);

// 4. Reset the filter
CFILTER_NAMEFirI64C_resetFilter(&l_myFilter);

// Do the filtering continuously (step 5), call l_myFilter.resetFilter() (step 4) if the filter must be decoupled from its former state

// 5. Call cyclic the filtering function with all the samples, after this step the result can be used
l_outputVal = CFILTER_NAMEFirI64C_doFiltering(&l_myFilter, l_inputVal);

// 6. Destroy the filter instance at the end of the usage, do not use the filter again, once it is destroyed
CFILTER_NAMEFirI64C_destroyFilter(&l_myFilter);
```

FIGURE 12: USAGE OF THE GENERATED FILES IN CASE OF A FIR ANSI C I64 FILTER

4.2 FIR C++ 98 Source Code Generation

For each FIR CPP 98 filter 3 files are generated under the path, which is set under **[1]** on Figure 2 on Page 2:

- **fircpp98.hpp**: general implementation of a FIR filter, one instance of this file can handle any amount of particular filter instances
- **<FILTER_NAME>_fir<XXX>cpp98.cpp**: the code implementation of the designed FIR filter
- **<FILTER_NAME>_fir<XXX>cpp98.hpp**: the header file of the implemented FIR filter

Where XXX is the applied data type, the data types are:

- F64: 64-bit floating point
- F32: 32-bit floating point
- I64: 64-bit signed integer
- I32: 32-bit signed integer
- I16: 16-bit signed integer

```
// Steps, how to use the generated files

// 1. Include the header file
#include "FILTER_NAME_firi64cpp98.hpp"

// 2. Make a filter instance
CFILTER_NAMEFirI64Cpp98 l_myFilter;

// 3. Reset the filter
l_myFilter.resetFilter();

// Do the filtering continuously (step 4), call l_myFilter.resetFilter() (step 3) if the filter must be decoupled from its former state

// 4. Call cyclic the forward filtering function with all the samples, after this step the result can be used
l_realOutputVal = l_myFilter.doFiltering(l_inputVal);
```

FIGURE 13: USAGE OF THE GENERATED FILES IN CASE OF A FIR CPP98 I64 FILTER

4.3 IIR ANSI C Source Code Generation

For each IIR ANSI C filter 4 files are generated under the path, which is set under **[1]** on Figure 2 on Page 2:

- **iir<XXX>c.c**: general code implementation of a filter, one instance of this file can handle any amount of particular filter instances with the given data type
- **iir<XXX>c.h**: the header of the general implementation file, one instance of this file can handle any amount of particular filter instances with the given data type
- **<FILTER_NAME>_iir<XXX>c.c**: the code implementation of the designed filter
- **<FILTER_NAME>_iir<XXX>c.h**: the header file of the implemented filter

Where XXX is the applied data type, the data types are:

- F64: 64-bit floating point
- F32: 32-bit floating point
- I64: 64-bit signed integer
- I32: 32-bit signed integer
- I16: 16-bit signed integer

```
// Steps, how to use the generated files

// 1. Include the header files
#include "FILTER_NAME_iiri64c.h"
#include "iiri64c.h"

// 2. Make a filter instance
struct CFILTER_NAMEIirI64C l_myFilter;

// 3. Initialize the filter
CFILTER_NAMEIirI64C_initFilter(&l_myFilter);

// 4. Reset the filter
CFILTER_NAMEIirI64C_resetFilter(&l_myFilter);

// Do the filtering continuously (step 5 and 6), call l_myFilter.resetFilter() (step 4) if the filter must be decoupled from its former state

// 5. Call cyclic the filtering function with all the samples, after this step the result can be used
l_outputVal = CFILTER_NAMEIirI64C_doFiltering(&l_myFilter, l_inputVal);

// 6. Call cyclic the rewards filtering function with all the samples, this function can be executed parallely in a different program thread
CFILTER_NAMEIirI64C_doRwdFiltering(&l_myFilter);

// 7. Destroy the filter instance at the end of the usage, do not use the filter again, once it is destroyed
CFILTER_NAMEIirI64C_destroyFilter(&l_myFilter);
```

FIGURE 14: USAGE OF THE GENERATED FILES IN CASE OF AN IIR ANSI C I64 FILTER

4.4 IIR C++ 98 Source Code Generation

For each IIR CPP 98 filter 3 files are generated under the path, which is set under **[1]** on Figure 2 on Page 2:

- **iircpp98.hpp**: general implementation of an IIR filter, one instance of this file can handle any amount of particular filter instances
- **<FILTER_NAME>_iir<XXX>cpp98.cpp**: the code implementation of the designed IIR filter
- **<FILTER_NAME>_iir<XXX>cpp98.hpp**: the header file of the implemented IIR filter

Where XXX is the applied data type, the data types are:

- F64: 64-bit floating point
- F32: 32-bit floating point
- I64: 64-bit signed integer
- I32: 32-bit signed integer
- I16: 16-bit signed integer

```
// Steps, how to use the generated files

// 1. Include the header file
#include "FILTER_NAME_iiri64cpp98.hpp"

// 2. Make a filter instance
CFILTER_NAMEIirI64Cpp98 l_myFilter;

// 3. Reset the filter
l_myFilter.resetFilter();

// Do the filtering continuously (step 4 and 5), call l_myFilter.resetFilter() (step 3) if the filter must be decoupled from its former state

// 4. Call cyclic the forward filtering function with all the samples, after this step the result can be used
l_realOutputVal = l_myFilter.doFiltering(l_inputVal);

// 5. Call cyclic the rewards filtering function with all the samples, this function can be executed parallely in a different program thread
l_myFilter.doRwdFiltering();
```

FIGURE 15: USAGE OF THE GENERATED FILES IN CASE OF AN IIR CPP98 I64 FILTER

4.5 Application of own template files

The Digital Filter Designer offers the user the opportunity to define own implementation of the template files.

The necessary steps to work with the own template files:

- Make the same files structure of the pattern files under an arbitrary path. Use the file names of the original pattern files!
- Set the path of the own template files under **[4]** on Figure 2 on Page 2
- Each source code template file must return a source code string "res_str" with the content of the particular source / header file. At the end of the processing, the “res_str” output string must contain a valid source / header file
- All the information can be gained from the tool is listed on Figure 16 below

```
CEnum2Val.get(CFilterVariantEnumShort, f_filterRequirement.m_filterVariant_i32) # filter variant: "FIR", "IIR"
CEnum2Val.get(CFilterVariantEnumLong, f_filterRequirement.m_filterVariant_i32) # filter variant: "FINITE IMPULSE RESPONSE", "INFINITE IMPULSE RESPONSE"

CEnum2Val.get(CFilterTypeEnumShort, f_filterRequirement.m_filterType_i32) # filter type: "LPF", "HPF", "BSF", "BPF"
CEnum2Val.get(CFilterTypeEnumLong, f_filterRequirement.m_filterType_i32) # filter type: "LOW PASS FILTER", "HIGH PASS FILTER", "BAND STOP FILTER", "BAND PASS FILTER"

CEnum2Val.get(CFilterApproximationEnumShort, f_filterRequirement.m_iirApproximation_i32) # filter type: "CH", "ICH", "BW", "CA"
CEnum2Val.get(CFilterApproximationEnumLong, f_filterRequirement.m_iirApproximation_i32) # filter type: "CHEBYSHEV", "INVERSE CHEBYSHEV", "BUTTERWORTH", "CAUER"

f_filterRequirement.m_checksum_str # requirement checksum

f_filterRequirement.getSourceCodeCommentStr() # Summary if the filter requirements as C/C++ comment, no "//" is necessary at the beginning of the line

f_filterResults.m_filterValid_b # Boolean value, if the filter has been validated successfully

f_filterResults.m_timeOfCreation_i # UNIX time of the creation of the filter

f_filterResults.m_numOfStages_i # number of filter stages

f_filterResults.m_dllVersion_str # DLL version of the filter designer DLL

f_filterResults.m_dllUser_str # Name of the licensed user of the filter designer DLL

f_filterResults.m_dllLicenseInfo_str # License information of the filter designer DLL

CEnum2Val.get(CSourceDataTypeShort, f_filterResults.m_dataType_i) # "I16", "I32", "I64", "F32", "F64"

CEnum2Val.get(CSourceDataTypeSpecifier, f_filterResults.m_dataType_i) # "short", "long", "long long", "float", "double"

f_filterResults.getCTypeCoeffString(i) # the i-th coefficient of a FIR filter, formated as a string, according to the data type settings of the filter

f_filterResults.getCTypeCoeffA1String(i) # the i-th A1 coefficient of a IIR filter, formated as a string, according to the data type settings of the filter
f_filterResults.getCTypeCoeffA2String(i) # the i-th A2 coefficient of a IIR filter, formated as a string, according to the data type settings of the filter
f_filterResults.getCTypeCoeffB0String(i) # the i-th B0 coefficient of a IIR filter, formated as a string, according to the data type settings of the filter
f_filterResults.getCTypeCoeffB1String(i) # the i-th B1 coefficient of a IIR filter, formated as a string, according to the data type settings of the filter
f_filterResults.getCTypeCoeffB2String(i) # the i-th B2 coefficient of a IIR filter, formated as a string, according to the data type settings of the filter

f_filterResults.m_checksum_str # Value of the result checksum

f_filterName_str # arbitrary name of the filter, it has to compatible character to be a valid file and C/C++ struct name

# An example: to initialize an integer variable "m_numOfStages" and assign the value of the number of stages to it, use the following line:
res_str += ("int m_numOfStages = %s;\n" % f_filterResults.m_numOfStages_i)
```

FIGURE 16: LIST OF ALL THE INFORMATION CAN BE GAINED FROM THE FILTER DESIGNER

5. Troubleshooting

- Failure message after starting a second instance of the tool:
  - Only one instance of the tool can be opened at the same time
- "WinError 10061" after the opening of diagrams:
  - This error indicates, the tool cannot communicate with the popUpDiagramServer.exe
  - Prove, if socked communication is possible with the local host via the port **[6]** on Figure 2 on page 2, and via the next ports for the further diagram instances. Port might be blocked by the applied firewall settings of your PC
  - Set the start-up timeout parameter (**[7]** on Figure 2 on Page 2) to a higher value
- "The diagram server file: <PATH> does not exist!" failure message after the opening of diagrams:
  - Prove, if the popUpDiagramServer.exe file exists under the configured path **[5]** on Figure 2 on page 2
- Diagrams are loaded slowly:
  - Since the diagrams are visualised by an external process, the data transfer may take a little time. Please wait…
  - In case of the digital filter validator the number of data points can be reduced, thus the visualisation process can be accelerated
- Unplausible data visualisation:
  - If not all processes of the popUpDiagramServer.exe are closed from a previous run of the client (dfdGui.exe), the client may communicate with a wrong instance of the server
  - To avoid it, close all instances popUpDiagramServer.exe in the Windows Task Manager
- Invalid filter in Digital Filter Validator, although the Digital Filter Analyser shown a valid filter:
  - The results shown in Digital Filter Analyser (see Chapter 2 on Page 5) are estimated results, based on calculations with the coefficient values, with double data types. The results of Digital Filter Validator (see Chapter 3 on Page 7) are simulated on the source code with the given data type, thus a difference between the two results is possible.
  - Modification of the simulation parameters might improve the quality of the simulation. In case of invalid filter try to modify the simulation parameters as it written in Chapter 3.1 on Page 8.
  - Modify the applied data type to a more accurate one: the applied data type has a strong effect on the accuracy of the filter. Especially the application of integer data types may cause unstable filter operation.
- Filter cannot be designed:
  - The filters are calculated by complex algorithms. There is no guarantee, all filter expectations can be fulfilled. You cannot expect a proper filter operation in case of too strict or even irreal filter specification (eg. very high attenuation in case of small difference between the stop-band and the cut-off frequency values).