Delft University of Technology
Faculty of Electrical Engineering,
Mathematics and Computer Science
Circuits & Systems Group

# (L)WDF Toolbox for MATLAB

## Reference Guide

**Version 1.0**

Ing. H.J. Lincklaen Arriëns
February 2006

(L)WDF Toolbox for MATLAB *Reference Guide*

## Table of Contents

## Alphabetical Listing of Functions

| | |
|---|---|
| cc2pars | Convert Cauer parameters $\rho$ and $\theta$ into toolbox parameters. |
| evalHs | Evaluate $\lvert H(s) \rvert$ for $s$ = j*w |
| evalHz | Evaluate $\lvert H(z) \rvert$ for $z$ = exp(j*2pi*fFs) |
| fs2fz | Bilinear frequency translation. |
| fz2fs | Inverse bilinear frequency translation. |
| Hs2Hz | Conversion of transfer function from *s*- to *z*-domain. |
| Hs2LWDF | Calculate the coefficients of a Lattice Wave Digital Filter (LWDF). |
| Hs_bpVlach | Vlach type band-pass filter design, with a free choice of zero-frequencies in the stop-bands and additional Unit Elements (UEs). |
| Hs_butter | Returns *H(s)* and its roots for a Butterworth low-pass filter. |
| Hs_cauer | Cauer low-pass filter design. |
| Hs_cauer_birec | Designs a discrete-time Bireciprocal Cauer low/high-pass filter. |
| Hs_cheby | Chebyshev low-pass filter design. |
| Hs_invcheby | Inverse Chebyshev low-pass filter design. |
| Hs_Vlach | Vlach/Sharpe type low-pass filter design, with a free choice of the number (limited by the filter order) and the frequencies of stop-band zeros, and the possibility to add Unit Elements. |
| Hz2Hs | Conversion of transfer function from *z*- to *s*-domain. |
| ladder2Magn | Reconstruct the magnitude plot for a given ladder filter. |
| ladder2WDF | Translate a ladder filter into a Wave Digital Filter (WDF) structure. |
| ladderSynthesis | Compute ladder element values given the input reactance function. |
| LWDF2Hz | Calculate the transfer function *H(z)* given an LWDF. |
| nladder2bp | Transform normalized low-pass ladder circuit into band-pass ladder. |
| nladder2bs | Transform normalized low-pass ladder circuit into band-stop ladder. |
| nladder2hp | Transform normalized low-pass ladder circuit into high-pass ladder. |
| nladder2lp | Normalized low-pass ladder circuit to denormalized low-pass ladder. |
| nlp2bp | Normalized low-pass to band-pass transformation. |
| nlp2bs | Normalized low-pass to band-stop transformation. |
| nlp2hp | Normalized low-pass to high-pass transformation. |
| nlp2lp | Normalized low-pass to low-pass transformation. |
| nlpf | Design of normalized low-pass filters in the continuous-time domain. |
| nlp_ladder | Designs a ladder type normalized low-pass filter. |
| plotHs | Magnitude and phase plots for transfer function(s) in the *s*-domain. |
| plotHz | Magnitude and phase plots for transfer function(s) in the *z*-domain. |
| rho2ripple | Reflection coefficient to ripple conversion. |
| ripple2rho | Ripple to reflection coefficient conversion. |
| showLadder | Print the values and plot the schematics of a ladder filter. |
| showLWDF | Display the coefficients and the structure of an LWDF. |
| showWDF | Show info and structure of Wave Digital Filter. |

## Design Graphical User Interfaces

| | |
|---|---|
| `bpVlach_GUI` | GUI for designing Vlach type band-pass Lattice Wave Digital Filters. |
| `wdf_GUI` | GUI for designing ladder networks and (Lattice) Wave Digital Filters. |

## Additional functions, linking to scheduling functions

| | |
|---|---|
| `LWDF_insRegs` | Insert pipeline registers between the slices of an LWDF. |
| `LWDF2cir` | Writes the LWDF structure as a .cir description for scheduling. |
| `WDF2cir` | Writes the WDF structure as a .cir description for scheduling. |

## Some Example m-files

| | |
|---|---|
| `examples.m` | general filter plot examples. |
| `example_Gazsi_ex5.m` | "*Cauer parameter (elliptical) bireciprocal low-pass filter*" from Gazsi |
| `examples_vlach.m` | several Vlach filter possibilities. |
| `example_2WDFs.m` | band-pass transformation and (L)WDF realizations. |
| `example_LCres.m` | LWDF realization of a simple first order band-pass filter. |

# cc2pars

**Purpose** Convert Cauer parameters $\rho$ and $\theta$ into toolbox parameters.

**Syntax**

```
[N,rp,rs,ftype,Wn,normtd] = cc2pars(filterOrder,rho,theta,ftype)
```

**Description**

`cc2pars` is meant to convert a Cauer filter classification based on parameters $\rho$ and $\theta$ into the parameters used throughout this toolbox.

In literature, particularly tables and catalogs, Cauer filters are often classified in a format like '`Cnn t rh ma`' , e.g. `C06 B 25 47`, in which

`nn` = filterOrder,
`t` = type 'A', 'B' or 'C' ('A' sometimes omitted),
`rh` = $\rho$ = reflection coefficient as a percentage,
`ma` = $\theta$ = modular angle in degrees,

while the Cauer functions in this toolbox need the parameters `filterOrder` (`N`), `passBandRipple_dB` (`rp`), `stopBandRipple_dB` (`rs`), `skwirMode` (`fType`), `cutOffFrequency` (`Wn`) and `freqNormMode` (`normtd`).

**See Also**

| | |
|---|---|
| `Hs_cauer` | Cauer low-pass filter design. |
| `nlp_ladder` | Design a ladder type normalized low-pass filter. |
| `rho2ripple` | Reflection coefficient to ripple conversion. |
| `ripple2rho` | Ripple to reflection coefficient conversion. |

## evalHs

**Purpose**

Evaluate $|H(s)|$ for $s$ = j*w

**Syntax**

```
Hw = evalHs(Hs,w)
```

**Description**

`Hw = evalHs(Hs,w)` calculates the value(s) of Hs for the given w, where w can be a vector. Usually Hs will be a complex value.

**See Also**

| | |
|---|---|
| `evalHz` | Evaluate $|H(z)|$ for $z$ = exp(j*2pi*fFs) |
| `plotHs` | Magnitude and phase plots for transfer function(s) in the s-domain. |

## evalHz

**Purpose**

Evaluate $|H(z)|$ for $z$ = exp(j*2pi*fFs)

**Syntax**

```
HfFs = evalHz(Hz,fFs)
```

**Description**

`HfFs = evalHz(Hz,fFs)` calculates Hz for the given `fFs`, where `fFs` is the frequency relative to the sample frequency ( 0 to 0.5 ) and may be a vector.

**Warning**: When a fairly large number of Unit Elements are being used, the accuracy of the output data for normalized frequency values near 0.5 may deteriorate.

**See Also**

| | |
|---|---|
| `evalHs` | Evaluate $|H(s)|$ for $s$ = j*w |
| `plotHz` | Magnitude and phase plots for transfer function(s) in the z-domain. |

## fs2fz

**Purpose** Bilinear frequency translation (from s-domain to z-domain).

**Syntax**

```
fz = fs2fz(fs)
fz = fs2fz(fs,sampleFreqFraction)
```

**Description**

`fz = fs2fz(fs)` converts the time-continuous frequency(vector) `fs` to its corresponding frequency(vector) `fz` in the time-discrete domain according to the bilinear transformation rules.
`fs = 1.0` corresponds with `fz = 0.25`

`fz = fs2fz(fs,sampleFreqFraction)`
As above, but now `sampleFracFraction` is the normalized time-discrete frequency (with Sample frequency = 1) to be used as the reference, which means that `fs = 1.0` will translate to `fz = sampleFracFraction`.

**See Also**

| | |
|---|---|
| `fz2fs` | Inverse bilinear frequency translation. |

## fz2fs

**Purpose** Inverse bilinear frequency translation (from z-domain to s-domain).

**Syntax**

```
fs = fz2fs(fz)
fs = fz2fs(fz,sampleFreqFraction)
```

**Description**

`fs = fz2fs(fz)` converts the time- discrete frequency(vector) `fz` to its corresponding frequency(vector) `fs` in the time- continuous domain according to the inverse bilinear transformation rule.
`fz = 0.25` corresponds with `fs = 1.0`

`fs = fz2fs(fz,sampleFreqFraction)`, as above, but now `sampleFracFraction` is the normalized time-discrete frequency (with Sample frequency = 1) to be used as the reference: `fz = sampleFracFraction` will translate to `fs = 1.0`.

**See Also**

| | |
|---|---|
| `fs2fz` | Bilinear frequency translation. |

# Hs2Hz

**Purpose** Conversion of transfer function from s- to z-domain.

**Syntax** `Hz = Hs2Hz(Hs)`

**Description** `Hz = Hs2Hz(Hs)` converts the continuous-time transfer function(s) H(s) to its corresponding discrete-time transfer function(s) H(z) using the bilinear transformation, such, that the frequency 1.0 of H(s) translates into the normalized discrete frequency 0.25 of H(z).

Thus, $s$ will be replaced with $s = \frac{z-1}{z+1}$.

`Hs` should be entered as the structure described in e.g. `Hs_butter`, resulting in the structure `Hz`:

| | | |
|---|---|---|
| `Hz.poly_fz` | - | the coefficients of the numerator function |
| `Hz.poly_gz` | - | the coefficients of the denominator function |
| `Hz.ident` | - | a string, describing the filter |
| `Hz.roots_fz` | - | the roots of the numerator |
| `Hz.roots_gz` | - | the roots of the denominator |

In the above, `Hs.poly_fs` and `Hs.poly_gs` are vectors of coefficients in descending powers of `s` (N,N-1,...,2,1,0), while `Hz.poly_fz` and `Hz.poly_gz` are vectors of coefficients in either descending positive powers of `z` (N,N-1,...,2,1,0), or ascending negative powers of `z` (0,-1,-2,...,-(N-1),-N).
If more than one polynomial function is to be transformed, `Hs` has to be entered as a vector e.g. `[Hs1 Hs2]`. Then `Hz` will become `[Hz1 Hz2]`.
In case Unit Elements are involved, `Hs.poly_fs` has to be given as the cell array { poly_fs without UEs; number of UEs }, while `Hz.poly_fz` will be returned as { poly_fz without UEs; number of UEs }.

**Examples**

**See Also**

| | |
|---|---|
| `Hs_butter` | Returns H(s) and its roots for a Butterworth low-pass filter. |
| `Hz2Hs` | Conversion of transfer function from z- to s-domain. |

# Hs2LWDF

**Purpose** Compute the coefficients for a Lattice Wave Digital Filter (LWDF).

**Syntax**

```
LWDF = Hs2LWDF(Hs)
LWDF = Hs2LWDF(Hs,figNo)
LWDF = Hs2LWDF(Hs,figNo,showMessages)
LWDF = Hs2LWDF(Hs,figNo,showMessages,plotOptionsString)
[LWDF,Hz,Messages] = Hs2LWDF(...)
```

**Description**

`LWDF = Hs2LWDF(Hs)` creates a structure LWDF given a transfer function `Hs`.
The structure `Hs` should be 1x1 struct array with fields organized as:

- `Hs.poly_fs` - the coefficients of the numerator function
- `Hs.poly_gs` - the coefficients of the denominator function
- `Hs.ident` - a string, describing the filter
- `Hs.roots_fs` - the roots of the numerator
- `Hs.roots_gs` - the roots of the denominator

where `poly_fs` and `poly_gs` are vectors of coefficients in descending powers of `s`.
Odd polynomials `Hs.poly_gs` result in low/high pass LWDFs, even polynomials in band-pass/stop LWDFs.
The structure `LWDF` will contain the fields

- `LWDF.wdaCodes` and
- `LWDF.gamma`

`LWDF.wdaCodes` is an array of 2 strings, which describe which adaptors are used and at what positions.
The following adaptor/delay combinations are recognized

- 't' - a single delay element
- 's' - one 2-port and one delay element
- 'S' - one 2-port with two cascaded delay elements
- 'd' - two 2-ports with two delay elements
- 'D' - two 2-ports with two times two cascaded delay elements
- 'x' - only an interconnection in this slot

`LWDF.gamma` gives the coefficient values for the 2-ports.
The block diagram shown (default) can be used to see which adaptor corresponds to which coefficient.

`[LWDF,Hz] = Hs2LWDF(Hs)` additionally returns the discrete-time magnitude transfer function `Hz` that can be reconstructed from the LWDF structure.

`[...] = Hs2LWDF(Hs,figNo)` can be used to control the block diagram plot.
Use `figNo = 0` if no output is wanted. When no `figNo` is specified, `figure(1)` will be used for plotting, otherwise `figure(figNo)`.

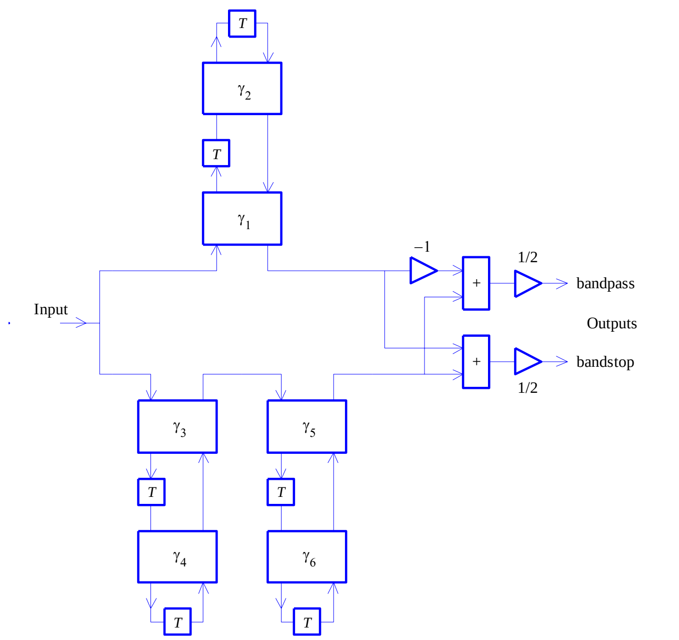
`[LWDF,Hz,Messages] = Hs2LWDF(Hs,figNo,showMessages)` can be used to control the printing of error messages in the workspace window.
`showMessages 1` acts as 'normal': errors are signalled, while
`showMessages 0` suppresses output and returns the error messages in the output string Messages.

```
[LWDF,Hz,Messages] = Hs2LWDF(Hs,figNo,showMessages,plotOptionsString)
```

To enable the output of an additional `Hz` plot, `plotOptions` can be entered (as a string), which are passed to `PlotHz`. See `PlotHz` for an extensive description of the options.

## Examples

```
% a 6th order band-pass filter (Butterworth approximation method)
[LWDF,Hz] = Hs2LWDF(nlp2bp(hs_butter(3),fz2fs(0.15),0.1),1,1,'1,2');
```




```
% an 11th order bireciprocal cauer filter
% with >=55 dB stop-band attenuation
hs = hs_cauer_birec(11,55);
Hs2LWDF(hs(1),1,1,'1,2');     % Note the hs(1) since hs_cauer_birec
                              % returns a 1x2 struct array
```

## See Also

`showLWDF` Display the coefficients and the structure of an LWDF.
`LWDF2Hz` Calculate the transfer function H(z) given an LWDF.

# Hs_bpVlach

**Purpose**

Vlach type band-pass filter design, with a free choice of zeros frequencies in the stop-bands (limited by filterOrder) and additional Unit Elements.

**Syntax**

```
Hs = Hs_bpVlach(filterOrder,passbandRipple_dB,cutOffFrequencies,
                                                    stopbandZeros)
Hs = Hs_bpVlach(filterOrder,passbandRipple_dB,cutOffFrequencies,
                                       stopbandZeros,nUnitElements)
Hs = Hs_bpVlach(filterOrder,passbandRipple_dB,cutOffFrequencies,
                          stopbandZeros,nUnitElements,freqNormMode)
```

**Description**

```
Hs = Hs_bpVlach(filterOrder,passbandRipple_dB,cutOffFrequencies,
                                                    stopbandZeros)
```

returns a structure `Hs` describing the continuous-time Vlach approximation of an ideal band-pass filter, given the specified parameters.
The structure `Hs` is organized as follows:

- `Hs.poly_fs` - the coefficients of the numerator function
- `Hs.poly_gs` - the coefficients of the denominator function
- `Hs.ident` - a string, describing the filter
- `Hs.roots_fs` - the roots of the numerator
- `Hs.roots_gs` - the roots of the denominator

where `poly_fs` and `poly_gs` are vectors of coefficients in descending powers of s.
`cutOffFrequencies` is expected to be a two element vector, defining resp. the lower and the upper cut-off frequency.
With `stopbandZeros`, transmission zeros outside the pass-band can be defined. Here, every non-zero frequency value is treated as two conjugated imaginary transmission zeros. Zero values each mean a single transmission zero at zero frequency. The total number of transmission zeros should be less than the (EVEN) `cutOffFrequency` (`stopbandZeros` can be an empty vector).

```
Hs = Hs_bpVlach(filterOrder,passbandRipple_dB,cutOffFrequencies,
                                       stopbandZeros,nUnitElements)
```

adds `nUnitElements` Unit Elements to the design. Each Unit Element contributes to the transfer function, by increasing the order of the approximation of the pass band and increasing the attenuation in the stop band by up to 7.7 dB.
With Unit Elements present, `poly_fs` cannot be written as a common polynomial any more, so `poly_fs` and `roots_fs` are extended to cell arrays:

- `Hs.poly_fs` ---> { poly_fs without UEs; number of UEs }.
- `Hs.roots_fs` ---> { roots_fs without UEs; number of UEs }.

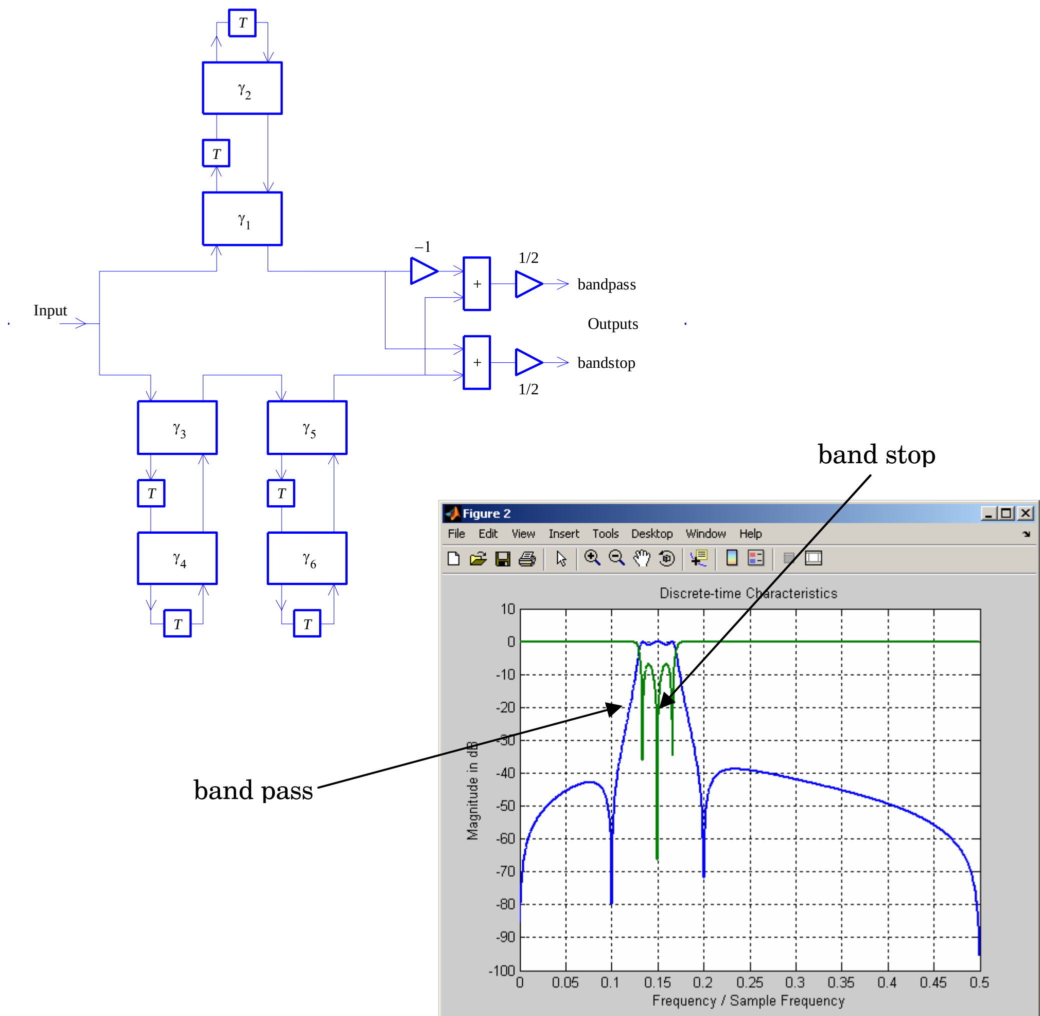
The sum of `filterOrder` and `nUnitElements` should be EVEN, and less than the total number of transmission zeros.

```
Hs = Hs_bpVlach(filterOrder,passbandRipple_dB,cutOffFrequencies,
                          stopbandZeros,nUnitElements,freqNormMode)
```

with `freqNormMode` 0 returns the same output as in the previous descriptions.
For `freqNormMode` 1, the cutoff frequencies are defined to be at the –3 dB magnitude level.

## Example

```
Hs = Hs_bpVlach(6,1,fz2fs([0.13 0.17]),fz2fs([0 0.1 0.2]),0,0);
plotHz( LWDF2Hz(Hs2LWDF(Hs)) ,1,2 );
```




## See Also

| | |
|---|---|
| `Hs_butter` | Returns `H(s)` and its roots for a Butterworth low-pass filter. |
| `Hs_cauer` | Cauer low-pass filter design. |
| `Hs_cauer_birec` | Designs a discrete-time Bireciprocal Cauer low/high-pass filter. |
| `Hs_cheby` | Chebyshev low-pass filter design. |
| `Hs_invcheby` | Inverse Chebyshev low-pass filter design. |
| `Hs_Vlach` | Vlach/Sharpe type low-pass filter design, with a free choice of the number (limited by the filter order) and the frequencies of zeros in the stop-band and additional Unit Elements. |
| `nlp2bp` | Normalized low-pass to band-pass transformation. |
| `nladder2bp` | Transform normalized low-pass ladder circuit into band-pass ladder circuit. |
| `fs2fz` | Bilinear frequency translation. |

# Hs_butter

**Purpose**

Returns H(s) and its roots for a Butterworth low-pass filter.

**Syntax**

```
Hs = Hs_butter(filterOrder)
Hs = Hs_butter(filterOrder,cutOffFrequency)
```

**Description**

`Hs = Hs_butter(filterOrder)` returns a structure `Hs` describing the continuous-time transfer function of a normalized (cutoff frequency = 1) Butterworth approximation of the ideal low-pass filter.
The structure `Hs` is organized as follows:

| | | |
|---|---|---|
| `Hs.poly_fs` | - | the coefficients of the numerator function |
| `Hs.poly_gs` | - | the coefficients of the denominator function |
| `Hs.ident` | - | a string, describing the filter |
| `Hs.roots_fs` | - | the roots of the numerator |
| `Hs.roots_gs` | - | the roots of the denominator |

where `poly_fs` and `poly_gs` are vectors of coefficients in descending powers of s (for the Butterworth filter, `poly_fs` = 1.0 ).
The length of the vector `poly_gs` equals the `filterOrder` +1.
By default the cutoff frequency is normalized to equal 1.0 at that point of the transition slope where the magnitude level equals –3dB.

`Hs = Hs_butter(filterOrder,cutOffFrequency)` returns the output parameters for the specified denormalized `cutOffFrequency`.

**Examples**

**See Also**

| | |
|---|---|
| `Hs_cauer` | Cauer low-pass filter design. |
| `Hs_cauer_birec` | Designs a discrete-time Bireciprocal Cauer low/high-pass filter. |
| `Hs_cheby` | Chebyshev low-pass filter design. |
| `Hs_invcheby` | Inverse Chebyshev low-pass filter design. |
| `Hs_Vlach` | Vlach/Sharpe type low-pass filter design, with a free choice of the number (limited by the filter order) and the frequencies of zeros in the stop-band and additional Unit Elements. |

# Hs_cauer

**Purpose** Cauer low-pass filter design.

**Syntax**

```
Hs = Hs_cauer(filterOrder,passbandRipple_dB,stopbandRipple_dB)
Hs = Hs_cauer(filterOrder,passbandRipple_dB,stopbandRipple_dB,
                                                        skwirMode)
Hs = Hs_cauer(filterOrder,passbandRipple_dB,stopbandRipple,
                                        skwirMode,cutOffFrequency)
Hs = Hs_cauer(filterOrder,passbandRipple_dB,stopbandRipple_dB,
                          skwirMode,cutOffFrequency,freqNormMode)
[Hs,wp] = Hs_cauer(...)
```

**Description**

`Hs = Hs_cauer(filterOrder,passbandRipple_dB,stopbandRipple_dB)` returns a structure `Hs` describing the continuous-time transfer function of a normalized (cutoff frequency = 1) Cauer approximation of the ideal low-pass filter.
The structure `Hs` is organized as follows:

- `Hs.poly_fs` - the coefficients of the numerator function
- `Hs.poly_gs` - the coefficients of the denominator function
- `Hs.ident` - a string, describing the filter
- `Hs.roots_fs` - the roots of the numerator
- `Hs.roots_gs` - the roots of the denominator

where `poly_fs` and `poly_gs` are vectors of coefficients in descending powers of s.
The length of the vector `poly_gs` equals the `filterOrder` +1, while that of `poly_fs` depends on the chosen filter type (see below).

`[Hs,wp] = Hs_cauer(filterOrder,passbandRipple_dB,stopbandRipple_dB)` additionally returns the frequencies of the peaks in the pass-band.

```
[...] = Hs_cauer(filterOrder,passbandRipple_dB,stopbandRipple_dB,
                                                        skwirMode)
```

Here, the parameter `skwirMode` indicates the filter type, as described by Skwirzynski in 1965: An odd order filter is always of type 'A' (also the default value here).
Even order Cauer filters of type 'A' can't be realized with lumped elements, so in that case the transfer function has to be 'transformed' using a Skwirzynski transformation. Allowable entries then are 'B' or 'C'.
Other implementation methods sometimes can cope with even ordered type 'A' designs.
The length of `poly_fs` equals `filterOrder` for odd, type 'A' filters; `filterOrder` +1 for even, type 'A' filters, or `filterOrder` –1 for type 'B' or 'C' filters.

```
Hs = Hs_cauer(filterOrder,passbandRipple_dB,stopbandRipple_dB,
                                        skwirMode,cutOffFrequency)
```

returns the output parameters for the specified denormalized `cutOffFrequency`.

```
Hs = Hs_cauer(filterOrder,passbandRipple_dB,stopbandRipple_dB,
                              skwirMode,cutOffFrequency,freqNormMode)
```

By default the cutoff frequency is normalized to equal 1.0 at that point of the transition slope where the design is 'symmetric' with respect to the pass-band and stop-band ripple. `freqNormMode` –1 gives the same output.
For `freqNormMode` 0, the cutoff frequency is defined to be at that point where the magnitude of the transition slope equals the minima of the pass-band ripple.
`freqNormMode` 1 defines the cutoff frequency to be at the –3 dB magnitude level.

## Examples

## See Also

| | |
|---|---|
| `Hs_butter` | Returns `H(s)` and its roots for a Butterworth low-pass filter. |
| `Hs_cauer_birec` | Designs a discrete-time Bireciprocal Cauer low/high-pass filter. |
| `Hs_cheby` | Chebyshev low-pass filter design. |
| `Hs_invcheby` | Inverse Chebyshev low-pass filter design. |
| `Hs_Vlach` | Vlach/Sharpe type low-pass filter design, with a free choice of the number (limited by the filter order) and the frequencies of zeros in the stop-band and additional Unit Elements. |

# Hs_cauer_birec

**Purpose** Design a discrete-time Bireciprocal Cauer low/high-pass filter.

**Syntax**

```
Hs = Hs_cauer_birec(filterOrder,stopbandRipple_dB)
[Hs,passbandRipple_dB,wp,Ws,ws] = Hs_cauer_birec(filterOrder,
                                                  stopbandRipple_dB)
```

**Description**

`Hs = Hs_cauer_birec(filterOrder,stopbandRipple_dB)` returns a structure `Hs` describing the continuous-time transfer function of a normalized (cutoff frequency = 1) Cauer approximation of the ideal low-pass filter, that, when translated into the discrete-time domain and implemented as a Wave Digital Filter results in a bireciprocal design, e.g. 'symmetrical' low-pass and high-pass transfer characteristics.
A bireciprocal design requires that `filterOrder` is odd, while the pass-band ripple will be derived from the specified `stopbandRipple_dB`.
The structure `Hs` is organized as follows:

| | |
|---|---|
| `Hs.poly_fs` | - the coefficients of the numerator function |
| `Hs.poly_gs` | - the coefficients of the denominator function |
| `Hs.ident` | - a string, describing the filter |
| `Hs.roots_fs` | - the roots of the numerator |
| `Hs.roots_gs` | - the roots of the denominator |

where `poly_fs` and `poly_gs` are vectors of coefficients in descending powers of s. Special emphasis is given to the fact that the roots of `Hs.poly_gs` have to be on the unit circle.

```
[Hs,passbandRipple_dB,wp,Ws,ws] = Hs_cauer_birec(filterOrder,
                                                  stopbandRipple_dB)
```

returns a number of additional parameters, such as the `passbandRipple_dB`, that is derived from the specified `stopbandRipple_dB`,
the positions of those frequencies (relative to the Sampling Frequency) in the pass-band where the magnitude equals 1.0 (`wp`),
the frequency on the transitionband where the magnitude equals that of the peaks of the stop-band ripple (`Ws`), and the frequencies of the stop-band zeros (`ws`).

**Examples**

**See Also**

| | |
|---|---|
| `Hs_cauer` | Cauer low-pass filter design. |
| `Hs2LWDF` | Calculate the coefficients of a Lattice Wave Digital Filter. |

# Hs_cheby

**Purpose** Chebyshev low-pass filter design.

**Syntax**

```
Hs = Hs_cheby(filterOrder,passbandRipple_dB)
Hs = Hs_cheby(filterOrder,passbandRipple_dB,cutOffFrequency)
Hs = Hs_cheby(filterOrder,passbandRipple_dB,cutOffFrequency,
freqNormMode)
```

**Description**

`Hs = Hs_cheby(filterOrder,passbandRipple_dB)` returns a structure `Hs` describing the continuous-time transfer function of a normalized (cutoff frequency = 1) Chebyshev approximation of the ideal low-pass filter.
The structure `Hs` is organized as follows:

- `Hs.poly_fs` - the coefficients of the numerator function
- `Hs.poly_gs` - the coefficients of the denominator function
- `Hs.ident` - a string, describing the filter
- `Hs.roots_fs` - the roots of the numerator
- `Hs.roots_gs` - the roots of the denominator

where `poly_fs` and `poly_gs` are vectors of coefficients in descending powers of s (for the Chebyshev approximation, `poly_fs` = 1.0 ).
The length of the vector `poly_gs` equals the `filterOrder` +1.
By default the cutoff frequency is normalized to equal 1.0 at that point of the transition slope where the magnitude equals that of the minima in the pass-band ripple.

`Hs = Hs_cheby(filterOrder,passbandRipple_dB,cutOffFrequency)` returns the output parameters for the specified denormalized `cutOffFrequency`.

```
Hs = Hs_cheby(filterOrder,passbandRipple_dB,cutOffFrequency,
                                                      freqNormMode)
```

with `freqNormMode` 0 returns the same output as in the previous description.
For `freqNormMode` 1, the cutoff frequency is defined to be at the –3dB magnitude level.

**Examples**

**See Also**

| | |
|---|---|
| `Hs_butter` | Returns `H(s)` and its roots for a Butterworth low-pass filter. |
| `Hs_cauer` | Cauer low-pass filter design. |
| `Hs_cauer_birec` | Designs a discrete-time Bireciprocal Cauer low/high-pass filter. |
| `Hs_invcheby` | Inverse Chebyshev low-pass filter design. |
| `Hs_Vlach` | Vlach/Sharpe type low-pass filter design, with a free choice of the number (limited by the filter order) and the frequencies of zeros in the stop-band and additional Unit Elements. |

# Hs_invcheby

**Purpose** Inverse Chebyshev low-pass filter design.

**Syntax**

```
Hs = Hs_invcheby(filterOrder,stopbandRipple_dB)
Hs = Hs_invcheby(filterOrder,stopbandRipple_dB,cutOffFrequency)
Hs = Hs_invcheby(filterOrder,stopbandRipple_dB,cutOffFrequency,
                                                        freqNormMode)
```

**Description**

`Hs = Hs_invcheby(filterOrder,stopbandRipple_dB)` returns a structure `Hs` describing the continuous-time transfer function of a normalized (cutoff frequency = 1) Inverse Chebyshev approximation of the ideal low-pass filter.
The structure `Hs` is organized as follows:

| | |
|---|---|
| `Hs.poly_fs` | - the coefficients of the numerator function |
| `Hs.poly_gs` | - the coefficients of the denominator function |
| `Hs.ident` | - a string, describing the filter |
| `Hs.roots_fs` | - the roots of the numerator |
| `Hs.roots_gs` | - the roots of the denominator |

where `poly_fs` and `poly_gs` are vectors of coefficients in descending powers of s (for the Chebyshev approximation, `poly_fs` = 1.0 ).
The length of the vector `poly_gs` equals the `filterOrder` +1, while the length of `poly_gs` equals `filterOrder` for odd filterOrders, and `filterOrder` +1 for even orders.
By default the cutoff frequency is normalized to equal 1.0 at that point of the transition slope where the magnitude equals that of the minima in the pass-band ripple.

`Hs = Hs_invcheby(filterOrder,stopbandRipple_dB,cutOffFrequency)` returns the output parameters for the specified denormalized `cutOffFrequency`.

```
Hs = Hs_invcheby(filterOrder,stopbandRipple_dB,cutOffFrequency,
                                                        freqNormMode)
```

with `freqNormMode` 0 returns the same output as in the previous description.
For `freqNormMode` 1, the cutoff frequency is defined to be at the –3dB magnitude level.

**Examples**

**See Also**

| | |
|---|---|
| `Hs_butter` | Returns `H(s)` and its roots for a Butterworth low-pass filter. |
| `Hs_cauer` | Cauer low-pass filter design. |
| `Hs_cauer_birec` | Designs a discrete-time Bireciprocal Cauer low/high-pass filter. |
| `Hs_cheby` | Chebyshev low-pass filter design. |
| `Hs_Vlach` | Vlach/Sharpe type low-pass filter design, with a free choice of the number (limited by the filter order) and the frequencies of zeros in the stop-band and additional Unit Elements. |

# Hs_Vlach

**Purpose**

Vlach/Sharpe type low-pass filter design, with a free choice of the number (limited by the filter order) and the frequencies of zeros in the stop-band and additional Unit Elements.

**Syntax**

```
Hs = Hs_Vlach(filterOrder,passbandRipple_dB)
Hs = Hs_Vlach(filterOrder,passbandRipple_dB,cutOffFrequency)
Hs = Hs_Vlach(filterOrder,passbandRipple_dB,cutOffFrequency,
                                                        stopbandZeros)
Hs = Hs_Vlach(filterOrder,passbandRipple_dB,cutOffFrequency,
                                          stopbandZeros,nUnitElements)
Hs = Hs_Vlach(filterOrder,passbandRipple_dB,cutOffFrequency,
                             stopbandZeros,nUnitElements,freqNormMode)
[Hs,wp] = Hs_Vlach(...)
```

**Description**

`Hs = Hs_Vlach(filterOrder,passbandRipple_dB)` returns a structure `Hs` describing the continuous-time normalized (cutoff frequency = 1) Vlach approximation of the ideal low-pass filter, given the specified parameters.
Note that without stop-band zeros and/or Unit Elements, the Vlach approximation completely equals the Chebyshev approximation.
The structure `Hs` is organized as follows:

| | |
|---|---|
| `Hs.poly_fs` | - the coefficients of the numerator function |
| `Hs.poly_gs` | - the coefficients of the denominator function |
| `Hs.ident` | - a string, describing the filter |
| `Hs.roots_fs` | - the roots of the numerator |
| `Hs.roots_gs` | - the roots of the denominator |

where `poly_fs` and `poly_gs` are vectors of coefficients in descending powers of s.

`[Hs,wp] = Hs_cauer(filterOrder,passbandRipple_dB)` additionally returns the frequencies of the peaks in the pass-band.

`[...] = Hs_Vlach(filterOrder,passbandRipple_dB,cutOffFrequency)` returns output parameters for the specified denormalized `cutOffFrequency`.

```
[...] = Hs_Vlach(filterOrder,passbandRipple_dB,cutOffFrequency,
                                                        stopbandZeros)
```

Here, `stopbandZeros` is a scalar or a vector giving frequencies of transmission zeros in the stop-band, in which

– every `stopbandZeros` frequency is internally treated as two imaginary conjugate frequency points,
– transmission zeros in infinity should be left out.

```
[...] = Hs_Vlach(filterOrder,passbandRipple_dB,cutOffFrequency,
                                          stopbandZeros,nUnitElements)
```

adds `nUnitElements` Unit Elements to the design. Each Unit Element contributes to the transfer function, by increasing the order of the approximation of the pass-band and increasing the attenuation in the stop-band by up to 7.7 dB.
With Unit Elements present, `poly_fs` cannot be written as a common polynomial any more, so `poly_fs` and `roots_fs` are extended to cell arrays:

`Hs.poly_fs` ---> { poly_fs without UEs; number of UEs }.

`Hs.roots_fs` ---> { roots_fs without UEs; number of UEs }.

```
[...] = Hs_Vlach(filterOrder,passbandRipple_dB, cutOffFrequency,
                             stopbandZeros,nUnitElements,freqNormMode)
```

with `freqNormMode` 0 returns the same output as in the previous descriptions. For `freqNormMode` 1, the cutoff frequencie is defined to be at the –3 dB magnitude level.

## Examples

## See Also

| | |
|---|---|
| `Hs_butter` | Returns `H(s)` and its roots for a Butterworth low-pass filter. |
| `Hs_cauer` | Cauer low-pass filter design. |
| `Hs_cauer_birec` | Designs a discrete-time Bireciprocal Cauer low/high-pass filter. |
| `Hs_cheby` | Chebyshev low-pass filter design. |
| `Hs_invcheby` | Inverse Chebyshev low-pass filter design. |
| `Hs_bpVlach` | Vlach type band-pass filter design, with a free choice of zeros frequencies in the stop-bands (limited by filterOrder) and additional Unit Elements. |

# Hz2Hs

**Purpose** Conversion of transfer function from z- to s-domain.

**Syntax**

```
Hs = Hz2Hs(Hz)
```

**Description**

`Hz = Hs2Hz(Hs)` converts the discrete-time transfer function(s) H(z) to its corresponding continuous-time transfer function(s) H(s) using the inverse bilinear transformation, such, that the discrete frequency value 0.25 of H(z) translates into the normalized frequency 1.0 of H(s).

Thus, replace $z$ with $z = \frac{s-1}{s+1}$.

`Hz` should be entered as as a structure according to

- `Hz.poly_fz` - the coefficients of the numerator function
- `Hz.poly_gz` - the coefficients of the denominator function
- `Hz.ident` - a string, describing the filter
- `Hz.roots_fz` - the roots of the numerator
- `Hz.roots_gz` - the roots of the denominator

where `Hz.poly_fz` and `Hz.poly_gz` are vectors of coefficients in either descending positive powers of `z` (N,N-1,...,2,1,0),
or ascending negative powers of `z` (0,-1,-2,...,-(N-1),-N).
The conversion will return a structure `Hs` (see p.e. `Hs_Butter.m`).
If more than one polynomial function is to be transformed, `Hz` has to be entered as a vector e.g. `[Hz1 Hz2]`, resulting in `[Hs1 Hs2]`.
In case Unit Elements are involved,
`Hz.poly_fz` has to be given as the cell array
{ poly_fz without UEs; number of UEs }, while `Hs.poly_fs` will be returned as
{ poly_fs without UEs; number of UEs }.

**Examples**

**See Also**

| | |
|---|---|
| `Hs_butter` | Returns H(s) and its roots for a Butterworth low-pass filter. |
| `Hs2Hz` | Conversion of transfer function from s- to z-domain. |

# ladder2Magn

**Purpose** Reconstruct the magnitude plot for a given ladder filter.

**Syntax**

```
magn_dB = Ladder2Magn(Ladder)
magn_dB = Ladder2Magn(Ladder,freqInfo)
magn_dB = Ladder2Magn(Ladder,freqInfo,figNo)
[magn_dB,freq] = Ladder2Magn(...)
```

**Description**

`magn_dB = Ladder2Magn(Ladder)` computes the magnitude transfer function from the ladder structure described in Ladder, where `Ladder` should contain the fields

`Ladder.elements` - a string describing the ladder
`Ladder.values` - a (set of) column vector(s) with the element values

The `elements`-string may consist of the following elements:

'`r`' for the source resistance, '`R`' for the load resistance,
'`l`' or '`L`' for an inductor,
'`c`' or '`C`' for a capacitor,
'`s`' or '`S`' for a serial resonator LC-circuit and
'`p`' or '`P`' for a parallel resonance LC-circuit.

The lowercase notation is used to identify elements in series arms, while the uppercase is used for elements in shunt arms.
Moreover, also Unit Elements can be present, denoted by a '`U`'.
The values-vector contains the values of the elements in the same sequence as given in the elements-string. Each resonator circuit needs two values, where always the first one denotes the inductor value and the second one the capacitor value.
**Note:** if multiple columns are present in `Ladder.values`, the FIRST ONE ONLY is selected to compute the magnitude transfer function.

`[magn_dB,freq] = Ladder2Magn(Ladder,freqInfo)` also returns a vector `freq` representing the frequency points for which the transfer function has been evaluated. Default is 1000 points in the range 1e-10 to 5.

`[...] = Ladder2Magn(Ladder,freqInfo)` uses the parameters `freqInfo` to judge what to do: if `freqInfo` is a scalar, it just specifies the number of frequency point to be used, given the same frequency range as above.
If `freqInfo` is a vector, this data is interpreted as the frequencies to be used for the evaluation.

`[...] = Ladder2Magn(Ladder,freqInfo,figNo)`
If `figNo` is 0, no plot will be made. If `figNo` is 1 or is left out, a plot will be made in figure(1), otherwise figure(`figNo`) will be drawn.

**Note from the author**

This function is restricted to work with only positive element values in the ladder circuit. This is not strictly necessary for the calculations in `ladder2Magn` or seen from the WDF point of view.
(Some?) negative element ladders will –if passed to `ladder2WDF`– result in feasible WDF structures.
For this to check, you have to comment out the test in lines 48..52 (or only line 51) in `ladder2Magn.m` (`ladder2Magn` is called from `nlp_ladder` and will normally block the calculations).
The drawback is that at this moment I'm not certain what the impact will be on e.g. the accuracy and/or the stability of the WDF and where and when limitations will occur.

try e.g.:

```
>> nlpLadder = nlp_ladder('invcheby',7,45,1);
>> nlpLadder.values(:,1)
>> WDF = ladder2wdF(nlpLadder,'2p_sym');
```

**See Also**

`nlp_ladder` Designs a ladder type normalized low-pass filter.

`nladder2lp, nladder2hp, nladder2bp, nladder2bs`
Transform normalized low-pass ladder circuit into resp. low-pass, high-pass, band-pass or band-stop ladder.

# ladder2WDF

**Purpose** Translate a ladder filter into a Wave Digital Filter structure.

**Syntax**

```
WDF = ladder2WDF(Ladder)
WDF = ladder2WDF(Ladder,wdfType,impulseResponseLength)
WDF = ladder2WDF(Ladder,wdfType,impulseResponseLength)
WDF = ladder2WDF(Ladder,wdfType,impulseResponseLength,figNo)
[WDF,fwdB,revB,allB] = ladder2WDF(...)
```

**Description**

`WDF = ladder2WDF(Ladder)` translates the LC-ladder network Ladder into an equivalent time-discrete Wave Digital Filter WDF containing only three-port adaptors.
The structure `Ladder` should contain the fields

`Ladder.elements` - a string describing the ladder
`Ladder.values` - a (set of) column vector(s) with the element values

The `elements`-string may consist of the following elements:

'`r`' for the source resistance, '`R`' for the load resistance,
'`l`' or '`L`' for an inductor,
'`c`' or '`C`' for a capacitor,
'`s`' or '`S`' for a serial resonator LC-circuit and
'`p`' or '`P`' for a parallel resonance LC-circuit.

The lowercase notation is used to identify elements in series arms, while the uppercase is used for elements in shunt arms.
Moreover, also Unit Elements can be present, denoted by a '`U`'.
The values-vector contains the values of the elements in the same sequence as given in the elements-string. Each resonator circuit needs two values, where always the first one denotes the inductor value and the second one the capacitor value.

**Note:** if multiple columns are present in `Ladder.values`, the FIRST ONE ONLY is selected to compute the magnitude transfer function.

The returned `WDF` will be a structure containing the fields

`WDF.wdaStruct`
`WDF.wdaNo`
`WDF.mulFacs`

In here, `WDF.wdaStruct`, describes the WDF block diagram:
A WDF block diagram is represented with 2 strings, one describing the adaptors in the signal path (bottom row), the second one (top row) describing the elements or adaptors connected to the serial or parallel ports of the first mentioned adaptors.
So, the bottom row can only consist of the following codes

'`s`' - for a reflection free 3-port serial adaptor,
'`p`' - a reflection free 3-port parallel adaptor,
'`S`' - a 3-port serial adaptor with two coefficients,
'`P`' - a 3-port parallel adaptor with two coefficients,
'`m`' - an output inverter or scaling factor, if needed.

For all these adaptors, port 1 is the input, port 3 the (reflection free) output, and

port 2 the interface to the top row elements.
Each element in the top row string is connected to port 2 of the adaptor in the same position in the bottom row string. Possible codes are:

- '+' - a single delay element (translation of a capacitance),
- '-' - a delay element in series with an inverter (inductance),
- 's' - a reflection free serial adaptor (series LC resonator),
- 'p' - a reflection free parallel adaptor (parallel LC resonator),
- 'x' - for an empty slot.

With the 's' and 'p' adaptors, port 1 is connected to a single delay element (translation of the capacitance), port 2 to a delay element in series with an inverter (the inductance), while the reflection free port 3 is connected to port 2 of the corresponding bottom row adaptor.
`WDF.wdaNo` defines the numbering of the individual adaptors,
`WDF.mulFacs` lists the multiplication coefficients of the adaptors, starting from adaptor one. The very last adaptor, which is not reflection free, needs two coefficients, while, if the bottom row string ends with an 'm', the last value will be the scaling coefficient.
Finally, the coefficients will be listed together with –unless deliberately suppressed– a block diagram of the WDF-structure. Also the magnitude transfer functions for forward and reverse outputs are recalculated from the WDF-structure and the scattering matrices as has been found. Both transfer function are showed in the top window of a two-figures plot. The bottom window shows the peak levels of the magnitude transfer functions of each B-output port as bar diagrams.

`[WDF,fwdB] = ladder2WDF(Ladder)` additionally returns the impulse response of the forward output.

`[WDF,fwdB,revB] = ladder2WDF(Ladder)` also returns the impulse response of the reflection or reverse output.

`[WDF,fwdB,revB,allB] = ladder2WDF(Ladder)` also returns all B-outputs in a 3 column by 'numbers of adaptors' matrix form.

`[...] = ladder2WDF(Ladder,wdfType)` can be used to choose among different filter structures.
`wdfType` can be:

- `'3p'` : use only three-port adaptors (default, see above).
- `'2p'` : use two-port adaptors for resonators in the top row.
  Top row codes are extended with
  - `'S'` - a 2-port translation of a serial LC resonance circuit,
  - `'P'` - a 2-port translation of a parallel LC resonance circuit.
- `'3p_sym'`: in case of an odd order of bottom-row adaptors and if the ladders shows a topological symmetry, a symmetric WDF structure using only three-ports will be constructed.
  For all these adaptors except the middle one, port 1 is the input, port 3 the output (to be reflection free, adapted to port 1 of the adaptor to its right if left from the middle, or to port 1 of the adaptor at its left if right from the middle).
  The adaptor in the middle has port 1 connected to port 3 of its left neighbor and port 3 connected to port 3 of its right

neighbor.
For all adaptors, port 2 is connected to the top row elements.

`'2p_sym'`: as '3p-sym', except for two-port adaptors for resonators in the top row.

`[...] = ladder2WDF(Ladder,wdfType,impulseResponseLength)` allows the user to specify the length of the impulse reponse that is used for calculating the frequency transfer function. Default value is 512, but this value may be too low for narrowband band-pass/stop filters.

`[...] = ladder2WDF(Ladder,wdfType,impulseResponseLength,figNo)` can be used to control the output plot. Use `figNo = 0` if no output is wanted. When no `figNo` is specified, figure(1) will be used for plotting, otherwise figure(`figNo`).

## Examples

## See Also

| | |
|---|---|
| `nlp_ladder` | Designs a ladder type normalized low-pass filter. |
| `showWDF` | Show info and structure of Wave Digital Filter. |

# ladderSynthesis

**Purpose** Compute ladder element values given the input reactance function.

**Syntax**

```
elValues = LadderSynthesis(InputYZ,Topology)
elValues = LadderSynthesis(InputYZ,Topology,stopbandZeroFrequencies)
[elValues,result] = LadderSynthesis(...)
```

**Description**

`elValues = LadderSynthesis(InputYZ,Topology)` calculates the element values for the given InputYZ and Topology, which has to describe a low-pass ladder filter. The input parameters are both structures, which should contain the fields `InputYZ.num`, `InputYZ.den` and `Topology.elTypeStr`, `Topology.ZorYStr`. `InputYZ.num` and `InputYZ.den` are polynomial descriptions of respectively the numerator and the denominator of the input reactance function, which can be treated as either an impedance or an admittance function, depending on the first character of `Topology.ZorYStr`. The polynomial descriptions list the coefficients of the polynomials in descending powers of `s`.
`Topology.elTypeStr` and `Topology.ZorYStr` describe the element types of the ladder circuit and their interconnections. Recognized element types of `Topology.elTypeStr` are:

- '`x`' - an inductor or a capacitor,
- '`W`' - a series or parallel resonance LC-circuit,
- '`U`' - a Unit Element.

the last element should be an '`R`' for the load resistance.
The source resistance is always expected to be 1.0 Ohm.
For each '`W`'-type element (if present), a resonance frequency should be specified in the 3rd input argument (see below).
`Topology.ZorYStr` is a string of '`Z`' and '`Y`' characters, with the same length as `Topology.elTypeStr`.
The returned `elValues` give the calculated values for the elements, preceded by a 1.0 for the source resistance. '`W`' types result in two elements, which are listed in the sequence: [ ...; inductance; capacitance; ... ]
If the circuit turns out to be not realizable, all `NaN`s are returned.

`[elValues,result] = LadderSynthesis(InputYZ,Topology)` also returns a `result`-flag, which is 1 in case the synthesis executed without errors, or a 0 in case the circuit turns out to be not realizable.

`[...] = LadderSynthesis(InputYZ,Topology,stopbandZeroFrequencies)` uses a third input parameter to specify the resonance freqencies of the '`W`'-type elements in `Topology.elTypeStr` (these are always located in the stop-band).

**Notes**

The choice which element actually results is determined by both `Topology.elTypeStr` and `Topology.ZorYStr`, e.g.

- '`x`', combined with '`Z`' ---> an inductor in a series arm,
- '`x`', combined with '`Y`' ---> a capacitor in a shunt arm,
- '`W`', combined with '`Z`' ---> a parallel resonator in a series arm,
- '`W`', combined with '`Y`' ---> a series resonator in a shunt arm.

## Examples

## See Also

| | |
|---|---|
| `nlp_ladder` | Designs a ladder type normalized low-pass filter. |

# LWDF2Hz

**Purpose**

Calculate the transfer function H(z) given an LWDF.

**Syntax**

```
Hz = LWDF2Hz(LWDF)
```

**Description**

`Hz = LWDF2Hz(LWDF)` derives the discrete-time transfer function H(z) that belongs to the given Lattice Wave Digital Filter (LWDF).
Here `LWDF` is a structure that should contain the fields

```
LWDF.wdaCodes
LWDF.gamma
```

`LWDF.wdaCodes` is an [2x?] character array, which is described in '`Hs2LWDF.m`', as well as in '`showLWDF.m`'.
`LWDF.gamma` contains the coefficients belonging to the `wdaCodes.`
The returned structure `Hz` contains both the description of the regular output, `Hz(1)`, as well as that of the reflected output, `Hz(2)`.
Both `Hz(1)` and `Hz(2)` contain

`Hz(*).poly_fz` - the coefficients of the numerator function
`Hz(*).poly_gz` - the coefficients of the denominator function
`Hz(*).ident` - a string, describing the filter
`Hz(*).roots_fz` - the roots of the numerator
`Hz(*).roots_gz` - the roots of the denominator

( with `* 1` or `2` ) where `Hz(*).poly_fz` and `Hz(*).poly_gz` are vectors of coefficients in either descending powers of positive `z` (N,N-1,...,2,1,0), or ascending powers of negative `z` (0,-1,-2,...,-(N-1),-N).

If the coefficients indicate that we deal with a bireciprocal low/high-pass filter, this property is mentioned in the `ident` field.

**Examples**

**See Also**

| | |
|---|---|
| `Hs2LWDF` | Calculate the coefficients of a Lattice Wave Digital Filter. |
| `showLWDF` | Display the coefficients and the structure of an LWDF. |

# nladder2bp

## Purpose

Transform normalized low-pass ladder circuit into band-pass ladder.

## Syntax

```
BpLadder = nladder2bp(NlpLadder,centerFrequency,BandWidth)
```

## Description

`BpLadder = nladder2bp(NlpLadder,centerFrequency,BandWidth)` transforms the elements of normalized low-pass ladder circuits to obtain band-pass ladder circuits with `BandWidth` around `centerFrequency`.
`NlpLadder`, as well as `BpLadder` are MATLAB structures:

| | | |
|---|---|---|
| `xxLadder.elements` | - | a string describing the ladder |
| `xxLadder.values` | - | a (set of) column vector(s) with the element values |

The low-pass elements-string may consist of the following elements:

- `'r'` for the source resistance, `'R'` for the load resistance,
- `'l'` for an inductor in a series arm,
- `'C'` for a capacitor in a shunt arm,
- `'p'` for a parallel resonator LC-circuit in a series arm,
- `'S'` for a serial resonator LC-circuit in a shunt arm.

**NOTE**: Unit Elements, denoted by a `'U'`, are NOT ALLOWED here.

The band-pass elements-string adds the elements:

- `'P'` for a parallel resonator LC-circuit in a shunt arm,
- `'s'` for a serial resonator LC-circuit in a series arm.

The values-vector contains the values of the elements in the same sequence as given in the elements-string. Each resonator circuit needs two values, where always the first one denotes the inductor value and the second one the capacitor value.
In case two or more resonator circuits are present in the `NlpLadder`, say representing frequencies `f1` and `f2`, `xxLadder.values` may contain several columns, representing the various permutations of the frequencies (here column1: `f1-f2` and column2: `f2-f1`).

## Examples

## See Also

| | |
|---|---|
| `ladderSynthesis` | Compute ladder element values for the input reactance function. |
| `nladder2lp, nladder2hp, nladder2bs` | Transform normalized low-pass ladder circuit into resp. low-pass, high-pass or band-stop ladder. |
| `nlp_ladder` | Design a ladder type normalized low-pass filter. |
| `showLadder` | Print the values and plot the schematics of a ladder filter. |

# nladder2bs

**Purpose** Transform normalized low-pass ladder circuit into band-stop ladder.

**Syntax**

```
BsLadder = nladder2bs(NlpLadder,centerFrequency,BandWidth)
```

**Description**

`BsLadder = nladder2bs(NlpLadder,centerFrequency,BandWidth)` transforms the elements of normalized low-pass ladder circuits to obtain band-stop ladder circuits with `BandWidth` around `centerFrequency`.
`NlpLadder`, as well as `BsLadder` are MATLAB structures:

`xxLadder.elements` - a string describing the ladder
`xxLadder.values` - a (set of) column vector(s) with the element values

The low-pass elements-string may consist of the following elements:

'`r`' for the source resistance, '`R`' for the load resistance,
'`l`' for an inductor in a series arm,
'`C`' for a capacitor in a shunt arm,
'`p`' for a parallel resonator LC-circuit in a series arm,
'`S`' for a serial resonator LC-circuit in a shunt arm.

**NOTE**: Unit Elements, denoted by a '`U`', are NOT ALLOWED here.

The band-stop elements-string will contain only '`r`', '`R`', '`p`'s and '`S`'s.

The values-vector contains the values of the elements in the same sequence as given in the elements-string. Each resonator circuit needs two values, where always the first one denotes the inductor value and the second one the capacitor value.
In case two or more resonator circuits are present in the `NlpLadder`, say representing frequencies `f1` and `f2`, `xxLadder.values` may contain several columns, representing the various permutations of the frequencies (here column1: `f1-f2` and column2: `f2-f1`).

**Examples**

**See Also**

| | |
|---|---|
| `ladderSynthesis` | Compute ladder element values for the input reactance function. |
| `nladder2lp, nladder2hp, nladder2bp` | Transform normalized low-pass ladder circuit into resp. low-pass, high-pass or band-pass ladder. |
| `nlp_ladder` | Design a ladder type normalized low-pass filter. |
| `showLadder` | Print the values and plot the schematics of a ladder filter. |

# nladder2hp

## Purpose

Transform normalized low-pass ladder circuit into high-pass ladder.

## Syntax

```
HpLadder = nladder2hp(NlpLadder,cutOffFrequency)
```

## Description

`HpLadder = nladder2hp(NlpLadder,cutOffFrequency)` transforms the elements of normalized low-pass ladder circuits to obtain high-pass ladder circuits with cutoff frequency `cutOffFrequency`.
`NlpLadder`, as well as `HpLadder` are MATLAB structures:

| | | |
|---|---|---|
| `xxLadder.elements` | - | a string describing the ladder |
| `xxLadder.values` | - | a (set of) column vector(s) with the element values |

The low-pass elements-string may consist of the following elements:

- `'r'` for the source resistance, `'R'` for the load resistance,
- `'l'` for an inductor in a series arm,
- `'C'` for a capacitor in a shunt arm,
- `'p'` for a parallel resonator LC-circuit in a series arm,
- `'S'` for a serial resonator LC-circuit in a shunt arm.

**NOTE**: Unit Elements, denoted by a `'U'`, are NOT ALLOWED here.

The high-pass elements-string adds the elements:

- `'L'` for an inductor in a shunt arm,
- `'c'` for a capacitor in a series arm.

The values-vector contains the values of the elements in the same sequence as given in the elements-string. Each resonator circuit needs two values, where always the first one denotes the inductor value and the second one the capacitor value.
In case two or more resonator circuits are present in the `NlpLadder`, say representing frequencies `f1` and `f2`, `xxLadder.values` may contain several columns, representing the various permutations of the frequencies (here column1: `f1-f2` and column2: `f2-f1`).

## Examples

## See Also

| | |
|---|---|
| `ladderSynthesis` | Compute ladder element values for the input reactance function. |
| `nladder2lp, nladder2bp, nladder2bs` | Transform normalized low-pass ladder circuit into resp. low-pass, band-pass or band-stop ladder. |
| `nlp_ladder` | Design a ladder type normalized low-pass filter. |
| `showLadder` | Print the values and plot the schematics of a ladder filter. |

# nladder2lp

## Purpose

Transform normalized low-pass ladder circuit into denormalized low-pass ladder.

## Syntax

```
LpLadder = nladder2lp(NlpLadder,cutOffFrequency)
```

## Description

`LpLadder = nladder2lp(NlpLadder,cutOffFrequency)` recalculates the element values of normalized low-pass ladder circuits to obtain ladder circuits with cutoff frequency `cutOffFrequency`.
`NlpLadder`, as well as `LpLadder` are MATLAB structures:

`xxLadder.elements` - a string describing the ladder
`xxLadder.values` - a (set of) column vector(s) with the element values

The low-pass elements-string may consist of the following elements:

'`r`' for the source resistance, '`R`' for the load resistance,
'`l`' for an inductor in a series arm,
'`C`' for a capacitor in a shunt arm,
'`p`' for a parallel resonator LC-circuit in a series arm,
'`S`' for a serial resonator LC-circuit in a shunt arm.

**NOTE**: Unit Elements, denoted by a '`U`', are NOT ALLOWED here.

The values-vector contains the values of the elements in the same sequence as given in the elements-string. Each resonator circuit needs two values, where always the first one denotes the inductor value and the second one the capacitor value.
In case two or more resonator circuits are present in the `NlpLadder`, say representing frequencies `f1` and `f2`, `xxLadder.values` may contain several columns, representing the various permutations of the frequencies
(here column1: `f1-f2` and column2: `f2-f1`).

## Examples

## See Also

`ladderSynthesis` Compute ladder element values for the input reactance function.
`nladder2hp, nladder2bp, nladder2bs` Transform normalized low-pass ladder circuit into resp. high-pass, band-pass or band-stop ladder.
`nlp_ladder` Design a ladder type normalized low-pass filter.
`showLadder` Print the values and plot the schematics of a ladder filter.

# nlp2bp

**Purpose** Normalized low-pass to band-pass transformation.

**Syntax**

```
Hsbp = nlp2bp(Hs,centerFrequency,bandWidth)
```

**Description**

`Hsbp = nlp2bp(Hs,centerFrequency,bandWidth)` transforms the description of the (normalized) low-pass transfer function `Hs` to the band-pass transfer function `Hsbp` with `centerFrequency` and `bandWidth`.
If the band-pass filter should have cut off frequencies at `f1` and `f2`, then `bandWidth = f2-f1` and `centerFrequency = sqrt(f1*f2)`.

The transformation formula used is $S_{nlp} \Rightarrow \frac{1}{BW}\left(\frac{s^2 + f_c{}^2}{s}\right)$, with

$f_c =$ `centerFrequency` and $BW =$ `bandwidth`.

**Examples**

**See Also**

`nlpf` Design of normalized low-pass filters in the continuous-time domain.
`nlp2lp, nlp2hp, nlp2bs` Normalized low-pass to resp. low-pass, high-pass and band-stop transformation.

# nlp2bs

**Purpose** Normalized low-pass to band-stop transformation.

**Syntax**

```
Hsbs = nlp2bs(Hs,centerFrequency,bandWidth)
```

**Description**

`Hsbs = nlp2bs(Hs,centerFrequency,bandWidth)` transforms the description of the (normalized) low-pass transfer function `Hs` to the band-stop transfer function `Hsbs` with `centerFrequency` and `bandWidth`.
If the band-stop filter should have cut off frequencies at `f1` and `f2`, then `bandWidth = f2-f1` and `centerFrequency = sqrt(f1*f2)`.

The transformation formula used is $S_{nlp} \Rightarrow BW\left(\frac{s}{s^2 + f_c^{\ 2}}\right)$, with

$f_c$ = `centerFrequency` and $BW$ = `bandwidth`.

**Examples**

**See Also**

`nlpf` Design of normalized low-pass filters in the continuous-time domain.
`nlp2lp, nlp2hp, nlp2bp` Normalized low-pass to resp. low-pass, high-pass and band-pass transformation.

# nlp2hp

**Purpose** Normalized low-pass to high-pass transformation.

**Syntax**

```
Hshp = nlp2hp(Hs, cutOffFrequency)
```

**Description**

`Hshp = nlp2hp(Hs, cutOffFrequency)` transforms the description of the (normalized) low-pass transfer function `Hs` to the high-pass transfer function `Hshp` with cutoff frequency `cutOffFrequency`.

The transformation formula used is $S_{nlp} \Rightarrow \left(\frac{f_c}{s}\right)$, with $f_c$ = `cutOffFrequency`.

**Examples**

**See Also**

`nlpf` Design of normalized low-pass filters in the continuous-time domain.
`nlp2lp, nlp2bp, nlp2bs` Normalized low-pass to resp. low-pass, band-pass and band-stop transformation.

# nlp2lp

**Purpose** Normalized low-pass to low-pass transformation.

**Syntax**

```
Hslp = nlp2lp(Hs, cutOffFrequency)
```

**Description** `Hslp = nlp2lp(Hs, cutOffFrequency)` transforms the description of the (normalized) low-pass transfer function `Hs` to the high-pass transfer function `Hslp` with cutoff frequency `cutOffFrequency`.

The transformation formula used is $S_{nlp} \Rightarrow \left(\frac{s}{f_c}\right)$, with $f_c$ = `cutOffFrequency`.

**Examples**

**See Also**

`nlpf` Design of normalized low-pass filters in the continuous-time domain.

`nlp2hp, nlp2bp, nlp2bs` Normalized low-pass to resp. high-pass, band-pass and band-stop transformation.

# nlpf

**Purpose** Design of normalized low-pass filters in the continuous-time domain.

**Syntax**

```
Hs = nlpf('butter',filterOrder)
Hs = nlpf('cheby',filterOrder,passbandRipple_dB)
Hs = nlpf('cheby',filterOrder,passbandRipple_dB,freqNormMode)
Hs = nlpf('invcheby',filterOrder,stopbandRipple_dB)
Hs = nlpf('invcheby',filterOrder,stopbandRipple_dB,freqNormMode)
Hs = nlpf('cauer',filterOrder,passbandRipple_dB,stopbandRipple_dB,
                                                         skwirNorm)
Hs = nlpf('cauer',filterOrder,passbandRipple_dB,stopbandRipple_dB,
                                            skwirNorm,freqNormMode)
Hs = nlpf('vlach',filterOrder,passbandRipple_dB)
Hs = nlpf('vlach',filterOrder,passbandRipple_dB,stopbandZeros)
Hs = nlpf('vlach',filterOrder,passbandRipple_dB,stopbandZeros,
                                                     nUnitElements)
Hs = nlpf('vlach',filterOrder,passbandRipple_dB,stopbandZeros,
                                        nUnitElements,freqNormMode)
```

**Description** `Hs = NLPF(....)` returns a structure `Hs` describing the continuous-time transfer function of a normalized (cutoff frequency = 1) approximation of the ideal low-pass filter.

The structure `Hs` is organized as follows:

| | |
|---|---|
| `Hs.poly_fs` | - the coefficients of the numerator function |
| `Hs.poly_gs` | - the coefficients of the denominator function |
| `Hs.ident` | - a string, describing the filter |
| `Hs.roots_fs` | - the roots of the numerator |
| `Hs.roots_gs` | - the roots of the denominator |

where `poly_fs` and `poly_gs` are vectors of coefficients in descending powers of s.
The syntax of the function is

`Hs = nlpf(ApproxMethod,filterOrder, ...var number of parameters... )`

`ApproxMethod` can be one of the strings: '`butter`', '`cheby`', '`invcheby`', '`cauer`' or '`vlach`', the number of additional parameters needed being dependant on the chosen approximation method.

Without going into detail, the set of possible commands is listed under **Syntax**. Details can be found in the descriptions of `Hs_butter`, `Hs_cheby`, etc.

**See Also**

| | |
|---|---|
| `Hs_butter` | Butterworth low-pass filter design. |
| `Hs_cauer` | Cauer low-pass filter design. |
| `Hs_cheby` | Chebyshev low-pass filter design. |
| `Hs_invcheby` | Inverse Chebyshev low-pass filter design. |
| `Hs_Vlach` | Vlach/Sharpe type low-pass filter design. |
| `nlp2lp, nlp2hp, nlp2bp, nlp2bs` | Normalized low-pass to resp. high-pass, band-pass and band-stop transformation. |

# nlp_ladder

**Purpose** Designs a ladder type normalized low-pass filter.

**Syntax**

```
NlpLadder = nlp_ladder('butter',filterOrder)
NlpLadder = nlp_ladder('butter',filterOrder,ZorY)
NlpLadder = nlp_ladder('cheby',filterOrder,passbandRipple_dB)
NlpLadder = nlp_ladder('cheby',filterOrder,passbandRipple_dB,
                                                       freqNormMode)
NlpLadder = nlp_ladder('cheby',filterOrder,passbandRipple_dB,
                                                  freqNormMode,ZorY)
NlpLadder = nlp_ladder('invcheby',filterOrder,stopbandRipple_dB)
NlpLadder = nlp_ladder('invcheby',filterOrder,stopbandRipple_dB,
                                                       freqNormMode)
NlpLadder = nlp_ladder('invcheby',filterOrder,stopbandRipple_dB,
                                                  freqNormMode,ZorY)
NlpLadder = nlp_ladder('cauer',filterOrder,passbandRipple_dB,
                                   stopbandRipple_dB,skwirNorm)
NlpLadder = nlp_ladder('cauer',filterOrder,passbandRipple_dB,
                      stopbandRipple_dB,skwirNorm,freqNormMode)
NlpLadder = nlp_ladder('cauer',filterOrder,passbandRipple_dB,
                 stopbandRipple_dB,skwirNorm,freqNormMode,ZorY)
NlpLadder = nlp_ladder('vlach',filterOrder,passbandRipple_dB,
                                                      stopbandZeros)
NlpLadder = nlp_ladder('vlach',filterOrder,passbandRipple_dB,
                             stopbandZeros,stopbandZerosVector)
NlpLadder = nlp_ladder('vlach',filterOrder,passbandRipple_dB,
          stopbandZeros,stopbandZerosVector,unitElementsVector)
NlpLadder = nlp_ladder('vlach',filterOrder,passbandRipple_dB,
        stopbandZeros,stopbandZerosVector,unitElementsVector,
                                                       freqNormMode)
NlpLadder = nlp_ladder('vlach',filterOrder,passbandRipple_dB,
        stopbandZeros,dstopbandZerosVector,unitElementsVector,
                                                  freqNormMode,XorY)
```

**Description** `NlpLadder = nlp_ladder(...)` returns a MATLAB structure `NlpLadder` that describes the topology of a ladder type low-pass filter. Printed information in the command window and some plots are also provided.
`NlpLadder` will contain the fields:

- `NlpLadder.elements` - a string describing the ladder
- `NlpLadder.values` - a (set of) column vector(s) with the element values

The elements-string may consist of the following elements:

- 'r' for the source resistance, 'R' for the load resistance,
- 'l' for an inductor in a series arm,
- 'C' for a capacitor in a shunt arm,
- 'p' for a parallel resonator LC-circuit in a series arm,
- 'S' for a serial resonator LC-circuit in a shunt arm.
- 'U' for Unit Elements.

The values-vector contains the values of the elements in the same sequence as given in the elements-string. Each resonator circuit needs two values, where

always the first one denotes the inductor value and the second one the capacitor value.
The source resistance Rsource ('r') is choosen to be always 1 Ohm.
In case two or more resonator circuits are present in the `NlpLadder`, say representing frequencies `f1` and `f2`, `NlpLadder.values` may contain several columns, representing the various permutations of the frequencies (here column1: `f1-f2` and column2: `f2-f1`).
At the end, functions `plotHs` and `ladder2Magn` are called automatically to compare the theoretical magnitude transfer function with the one reconstructed from the ladder structure as has been found.

The syntax of the function is

```
NlpLadder = nlp_ladder(ApproxMethod,filterOrder,
                                           ...additional params... )
```

where `ApproxMethod` can be one of the strings: '`butter`', '`cheby`', '`invcheby`', '`cauer`' or '`vlach`', the number of additional parameters needed being dependant on the chosen approximation method.
Without going into much detail, the set of possible commands is listed under **Syntax**. Details can be found in the descriptions of `Hs_butter, Hs_cheby`, etc.

## Notes

The variable '`ZorY`' is used to specify whether the ladder circuit should start with a series arm (`XorY = 'Z'`) or with a shunt arm (`ZorY = 'Y'`).
If left out, the ladder will start with a shunt arm.
The first element can be a Unit Elements, if needed.

Concerning the '`Vlach`'-ladders:
`stopbandZeroVector` can be used, when there are less `stopbandZeros` than the filterOrder allows for, to specify the locations of the resonators using 0's (e.g. no resonator here) and 1's (e.g. resonator here). If not specified, resonators are by default filled in from output to input.
Furthermore, given a number of zeros and a number of locations, all possible permutations of the resonators will be calculated.

The number of and the locations of the Unit Elements can be specified in the `unitElementsVector` with 0's and 1's. Unit Elements are filled in from input to output.

## Examples

```
NlpLadder = nlp_ladder('vlach',9,1,[1.2 1.5])
NlpLadder = nlp_ladder('vlach',9,1,[1.2 1.5], [ 0 1 1 0 ] )
NlpLadder = nlp_ladder('vlach',3,1,1.5,[1], [ 1 1 ] )
NlpLadder = nlp_ladder('vlach',3,1,1.5,[1], [ 0 1 1 ] )
NlpLadder = nlp_ladder('vlach',0,1,[],[], [ 1 1 1 1 1 ] )
```

## See Also

`nlpf` Design of normalized low-pass filters in the continuous-time domain.
`ladderSynthesis` Compute ladder element values given the input reactance function.
`plotHs` Magnitude and phase plots for transfer function(s) in the s-domain.
`ladder2Magn` Reconstruct the magnitude plot for a given ladder filter.

# plotHs

**Purpose** Magnitude and phase plots for transfer function(s) in the s-domain.

**Syntax**

```
plotHs(Hs)
plotHs(Hs,axisMode)
plotHs(Hs,axisMode,figNo)
plotHs(Hs,axisMode,figNo,freqInterval)
plotHs(Hs,axisMode,figNo,freqInterval,phasePlotMode)
plotHs(Hs,axisMode,figNo,freqInterval,phasePlotMode,nPoints)
plotHs(Hs,axisMode,figNo,freqInterval,phasePlotMode,nPoints,titleString)
plotHs(Hs,axisMode,figNo,freqInterval,phasePlotMode,nPoints,
                                               titleString,legendString)
```

**Description** `plotHs(Hs)` plots the magnitude of the transfer function `Hs` with linear axes.
`Hs` has to be entered as a structure, with the following fields:

| | |
|---|---|
| `Hs.poly_fs` | - the coefficients of the numerator function |
| `Hs.poly_gs` | - the coefficients of the denominator function |
| `Hs.ident` | - a string, describing the filter |
| `Hs.roots_fs` | - the roots of the numerator |
| `Hs.roots_gs` | - the roots of the denominator |

where `poly_fs` and `poly_gs` are vectors of coefficients in descending powers of s.
More than one transfer function can be plotted, by writing the `Hs`'s as a vector e.g. `[Hs1 Hs2]`. The color scheme is dictated by MATLAB.
In case Unit Elements (UEs) are involved in the description of `Hs`, `poly_fs` and `roots_fs` are extended to cell arrays:

`Hs.poly_fs` ---> { `poly_fs` without UEs; number of UEs }.
`Hs.roots_fs` ---> { `roots_fs` without UEs; number of UEs }.

`plotHs(Hs,axisMode)` enables plotting with different scales, viz.

`axisMode` 0 (default mode) uses linear frequency- and magnitude-axes,
`axisMode` 1 uses a linear frequency axis and a magnitude axis in dB,
`axisMode` 2 plots in a logarithmic frequency scale (base 10) with a magnitude scale in dBs.

`plotHs(Hs,axisMode,figNo)` also specifies which figure window to use.

`plotHs(Hs,axisMode,figNo,freqInterval)` gives the user control over the frequency range to be plotted. Default `freqInterval` values are [0 5] for linear, [0.01 100] for logarithmic plots. A `freqInterval` [] signals to use the default values.

`plotHs(Hs,axisMode,figNo,freqInterval,phasePlotMode)` should be used to also plot the phase transfer characteristics, where

`phasePlotMode` 0 (default value) means no phase plot,
`phasePlotMode` 1 plots the phase function in a seperate figure,
`phasePlotMode` 2 plots the phase function below the magnitude transfer function in the same figure.

`plotHs(Hs,axisMode,figNo,freqInterval,phasePlotMode,nPoints)` gives control over the number of points to be calculated for the plot (default 1000).

`plotHs(Hs,axisMode,figNo,freqInterval,phasePlotMode,nPoints,titleString)` When no `titleString` is specified, 'Continuous-time Characteristics' is shown above the plot. `titleString` replaces the word 'Characteristics' with its own text.

```
plotHs(Hs,axisMode,figNo,freqInterval,phasePlotMode,nPoints,
                                          titleString,legendString)
```
can be used to distinguish combined plots. The legend strings should be entered as a column vector of strings (same lengths!).

**Examples**

```
Hs1 = nlpf('butter',5);
plotHs(Hs1)

Hs2 = nlpf('vlach',5,1,[2.0 3.0],1,1);
plotHs( [Hs1;Hs2], 2,1,[],2,5000,'Transfer Functions', ...
                                     ['Butter, N = 5';'Vlach, N5+1UE'])
```

**See Also**

`plotHz` Magnitude and phase plots for transfer function(s) in the z-domain.

# plotHz

**Purpose** Magnitude and phase plots for transfer function(s) in the z-domain.

**Syntax**

```
plotHz(Hz)
plotHz(Hz,axisMode)
plotHz(Hz,axisMode,figNo)
plotHz(Hz,axisMode,figNo,phasePlotMode)
plotHz(Hz,axisMode,figNo,phasePlotMode,nPoints)
plotHz(Hz,axisMode,figNo,phasePlotMode,nPoints,titleString)
plotHz(Hz,axisMode,figNo,phasePlotMode,nPoints,titleString,legendString)
```

**Description**

`plotHz(Hz)` plots the fundamental part of the repetitive magnitude transfer function `Hz` with a linear magnitude scale and a normalized frequency scale, e.g. the actual frequency relative to the sample frequency ( 0 to 0.5 ).
(Note that this corresponds to a range from 0 to pi, for a frequency expressed in radians/sample).
`Hz` has to be entered as a structure, with the following fields:

- `Hz.poly_fz` - the coefficients of the numerator function
- `Hz.poly_gz` - the coefficients of the denominator function
- `Hz.ident` - a string, describing the filter
- `Hz.roots_fz` - the roots of the numerator
- `Hz.roots_gz` - the roots of the denominator

where `poly_fz` and `poly_gz` are vectors of coefficients in either descending positive powers of `z` (N,N-1,...,2,1,0), or ascending negative powers of `z` (0,-1,-2,...,-(N-1),-N).
More than one transfer function can be plotted, by writing the `Hz`'s as a vector e.g. `[Hz1 Hz2]`. The color scheme is dictated by MATLAB.
In case Unit Elements (UEs) are involved in the description of `Hz`, `poly_fz` and `roots_fz` are extended to cell arrays:

- `Hz.poly_fz` ---> { `poly_fz` without UEs; number of UEs }.
- `Hz.roots_fz` ---> { `roots_fz` without UEs; number of UEs }.

`plotHz(Hz,axisMode)` enables plotting with different scales, viz.

- `axisMode` 0 (default mode) uses linear frequency- and magnitude-axes,
- `axisMode` 1 uses a linear frequency axis and a magnitude axis in dB.

`plotHz(Hz,axisMode,figNo)` also specifies which figure window to use.

`plotHz(Hz,axisMode,figNo,phasePlotMode)` should be used to also plot the phase transfer characteristics, where

- `phasePlotMode` 0 (default value) means no phase plot,
- `phasePlotMode` 1 plots the phase function in a seperate figure,
- `phasePlotMode` 2 plots the phase function below the magnitude transfer function in the same figure.

`plotHz(Hz,axisMode,figNo,phasePlotMode,nPoints)` gives control over the number of points to be calculated for the plot (default 1000).

`plotHz(Hz,axisMode,figNo,phasePlotMode,nPoints,titleString)`
When no `titleString` is specified, 'DIscrete-time Characteristics' is shown above the plot. `titleString` replaces the word 'Characteristics' with its own text.

`plotHz(Hz,axisMode,figNo,phasePlotMode,nPoints,titleString,legendString)`
can be used to distinguish combined plots. The legend strings should be entered as a column vector of strings (same lengths!).

## Warning

When a fairly large number of Unit Elements are being used, the accuracy of the output data for normalized frequency values near a frequency value of 0.5 may deteriorate.

## Examples

```
Hsn = nlpf('butter',5);
Hs1 = nlp2lp(Hsn,fz2fs(0.15));
Hz1 = Hs2Hz(Hs1);
plotHz(Hz1)

Hs2 = Hs_vlach(5,1,fz2fs(0.15),fz2fs([0.2 0.25]),2,1);
Hz2 = Hs2Hz(Hs2);
plotHz( [Hz1;Hz2], 1,3,2,5000,'Transfer Functions', ...
                                  ['Butter, N = 5';'Vlach, N5+2UE'])
```

## See Also

`plotHs` Magnitude and phase plots for transfer function(s) in the s-domain.

# rho2ripple

**Purpose** Reflection coefficient to ripple conversion.

**Syntax**

```
ripple_dB = rho2ripple(rho)
```

**Description**

`ripple_dB = rho2ripple(rho)` converts a reflection coefficient `rho` (given as a percentage) to a pass band ripple in dB.

**Examples**

**See Also**

| | |
|---|---|
| `ripple2rho` | Ripple to reflection coefficient conversion. |

# ripple2rho

**Purpose** Ripple to reflection coefficient conversion.

**Syntax**

```
rho = ripple2rho(ripple_dB)
```

**Description**

`rho = ripple2rho(ripple_dB)` converts a pass band ripple in dB to a reflection coefficient `rho` (written as a percentage).

**Examples**

**See Also**

| | |
|---|---|
| `rho2ripple` | Reflection coefficient to ripple conversion. |

# showLadder

**Purpose** Print the values and plot the schematics of a ladder filter.

**Syntax**

```
showLadder(Ladder)
showLadder(Ladder,figNo)
showLadder(Ladder,figNo,figNameString)
```

**Description**

`showLadder(Ladder)` prints the element values and plots the schematics of a ladder topology, given in the MATLAB structure `Ladder`.
`Ladder` should contain the fields

`Ladder.elements` - a string describing the ladder
`Ladder.values` - a (set of) column vector(s) with the element values

The `elements`-string may consist of the following elements:

'`r`' for the source resistance, '`R`' for the load resistance,
'`l`' or '`L`' for an inductor,
'`c`' or '`C`' for a capacitor,
'`s`' or '`S`' for a serial resonator LC-circuit and
'`p`' or '`P`' for a parallel resonance LC-circuit.

The lowercase notation is used to identify elements in series arms, while the uppercase is used for elements in shunt arms.
Moreover, also Unit Elements can be present, denoted by a '`U`'.
The values-vector contains the values of the elements in the same sequence as given in the elements-string. Each resonator circuit needs two values, where always the first one denotes the inductor value and the second one the capacitor value.
In case two or more resonator circuits are present in the `NlpLadder`, say representing frequencies `f1` and `f2`, `NlpLadder.values` may contain several columns, representing the various permutations of the frequencies (here column1: `f1-f2` and column2: `f2-f1`).

`showLadder(Ladder,figNo)` indicates which figure to use for the plot.

`showLadder(Ladder,figNo,figNameString)` adds an identification text to the figure's Title Bar.

**Examples**

**See Also**

`ladderSynthesis` Compute ladder element values given the input reactance function.

# showLWDF

**Purpose** Display the coefficients and the structure of an LWDF.

**Syntax**

```
showWDF(LWDF)
showWDF(LWDF,dlyLorR)
showWDF(LWDF,dlyLorR,figNo)
```

**Description**

`showLWDF(LWDF)` prints the coefficients of the Lattice Wave Digital Filter `LWDF` in the workspace window, and plots a block diagram of the corresponding filter structure in Figure 1.
The structure `LWDF` will contain the fields

`LWDF.wdaCodes` and
`LWDF.gamma`,
and an optional `LWDF.insRegs` field.

From these, `LWDF.wdaCodes` should be an array of 2 strings, which describe the presence and positions of the adaptors to be used.
The following adaptor combinations are recognized

'`t`' - a single delay element
'`s`' - one 2-port and one delay element
'`S`' - one 2-port with two cascaded delay elements
'`d`' - two 2-ports with two delay elements
'`D`' - two 2-ports with two times two cascaded delay elements
'`x`' - only an interconnection in this slot

`LWDF.gamma` gives the coefficient values for the 2-ports.

For realistic hardware realizations, pipeline registers may have been inserted in top and bottom chains. The presence of such register pairs are listed in an additional vector field `LWDF.insRegs` ( a 1 means that a register pair should be inserted between this slice and the next one ).

`SHOWLWDF(WDF,dlyLorR)` can be used to specify where to draw the 'connecting' delay in 2nd degree sections, viz. in the left arm (`dlyLorR = 'L'`) or in the rightmost arm (`dlyLorR = 'R'`). If omitted, an `'L'` will be used.

`showLWDF(LWDF,dlyLorR,figNo)` controls the plot option:

`figNo` = 0 means that no circuit diagram should be shown, while
`figNo` = # results, next to the print, in a circuit diagram in figure(#).

**Examples**

**See Also**

| | |
|---|---|
| `Hs2LWDF` | Calculate the coefficients for a Lattice Wave Digital Filter. |
| `LWDF2Hz` | Calculate the transfer function H(z) given an LWDF. |
| `showWDF` | Show info and structure of Wave Digital Filter. |

# showWDF

**Purpose** Show info and structure of Wave Digital Filter.

**Syntax**

```
showWDF(WDF)
showWDF(WDF,dlyLorR)
showWDF(WDF,dlyLorR,figNo)
```

**Description**

`showWDF(WDF)` prints the coefficients of the Wave Digital Filter `WDF` in the workspace window, and plots a block diagram of the corresponding filter structure in Figure 1.

`WDF` should be a structure that contain the fields

```
WDF.wdaStruct
WDF.wdaNo
WDF.mulFacs
```

In here, `WDF.wdaStruct`, describes the WDF block diagram:

A WDF block diagram is represented with 2 strings, one describing the adaptors in the signal path (bottom row), the second one (top row) describing the elements or adaptors connected to the serial or parallel ports of the first mentioned adaptors.

So, the bottom row can only consist of the following codes

- `'s'` - for a reflection free 3-port serial adaptor,
- `'p'` - a reflection free 3-port parallel adaptor,
- `'S'` - a 3-port serial adaptor with two coefficients,
- `'P'` - a 3-port parallel adaptor with two coefficients,
- `'m'` - an output inverter or scaling factor, if needed.

For all these adaptors, port 1 is the input, port 3 the (reflection free) output, and port 2 the interface to the top row elements.

Each element in the top row string is connected to port 2 of the adaptor in the same position in the bottom row string. Possible codes are:

- `'+'` - a single delay element (translation of a capacitance),
- `'-'` - a delay element in series with an inverter (inductance),
- `'s'` - a reflection free serial adaptor (series LC resonator),
- `'p'` - a reflection free parallel adaptor (parallel LC resonator),
- `'S'` - a 2-port translation of a serial LC resonance circuit,
- `'P'` - a 2-port translation of a parallel LC resonance circuit,
- `'x'` - for an empty slot.

With the `'s'` and `'p'` adaptors, port 1 is connected to a single delay element (translation of the capacitance), port 2 to a delay element in series with an inverter (the inductance), while the reflection free port 3 is connected to port 2 of the corresponding bottom row adaptor.

`WDF.wdaNo` defines the numbering of the individual adaptors,

`WDF.mulFacs` lists the multiplication coefficients of the adaptors, starting from adaptor one. The very last adaptor, which is not reflection free, needs two coefficients, while, if the bottom row string ends with an `'m'`, the last value will be the scaling coefficient.

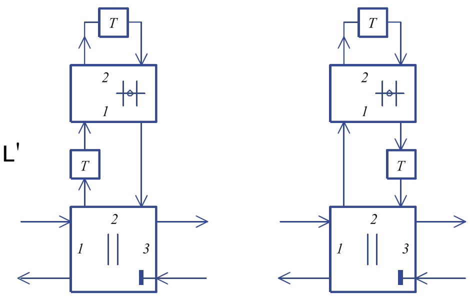
`SHOWWDF(WDF,dlyLorR)` can be used to specify where to draw the first delay when 2-port adaptors are used in the top row (respectively in the left A1 or the rightmost B1 connection of the top-row adaptor). `dlyLorR` should be an '`L`' or '`R`'. If omitted, an '`L`' will be assumed.

`showWDF(WDF,dlyLorR,figNo)` controls the plot option:

> `figNo` = 0 means that no circuit diagram should be shown, while
> `figNo` = # results, next to the print, in a circuit diagram in figure(#).

## Examples

`dlyLorR = 'L'`



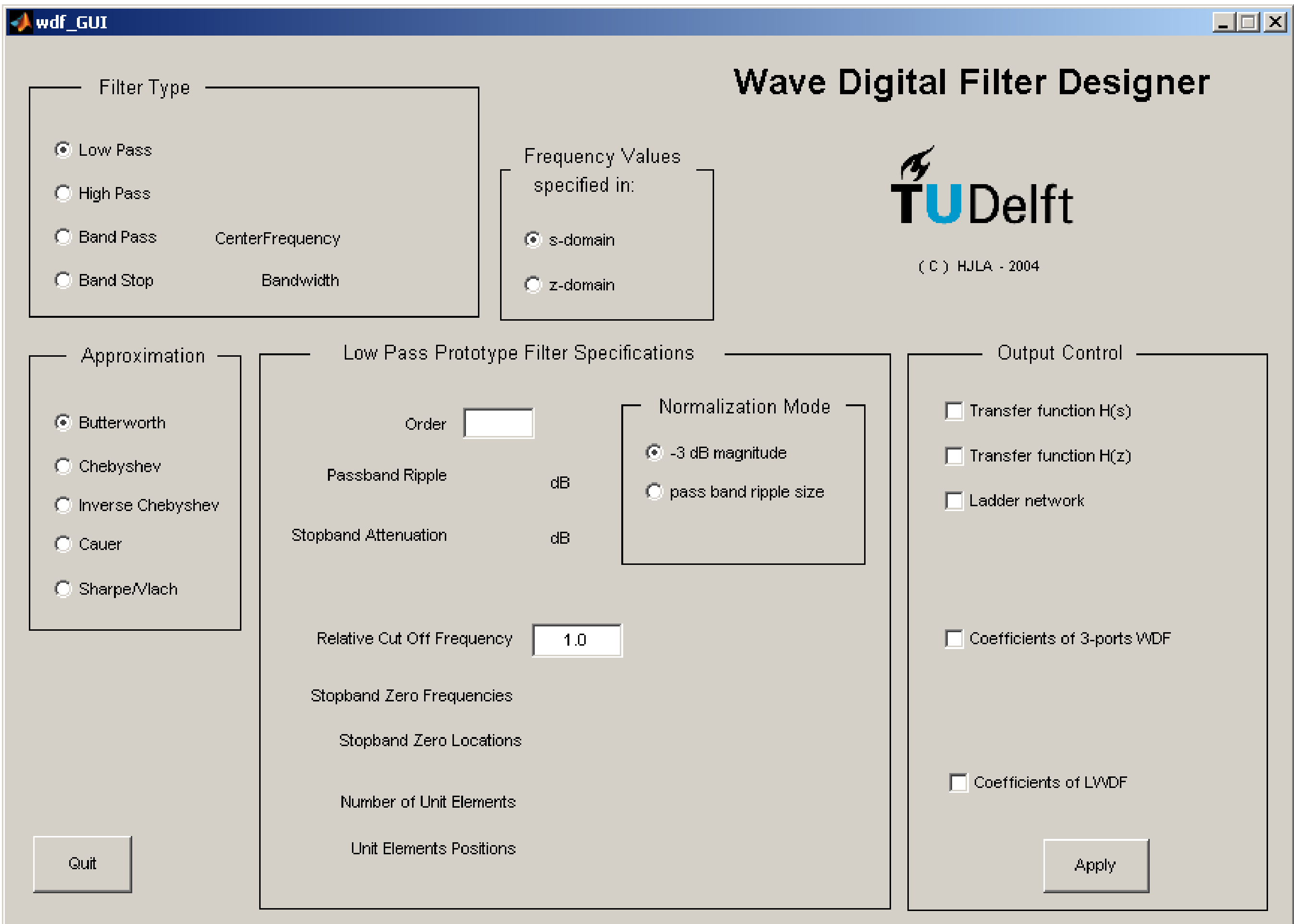
`dlyLorR = 'R'`

## See Also

| | |
|---|---|
| `ladder2WDF` | Translate a ladder filter into a Wave Digital Filter structure. |
| `showLWDF` | Show info and structure of Lattice Wave Digital Filter. |

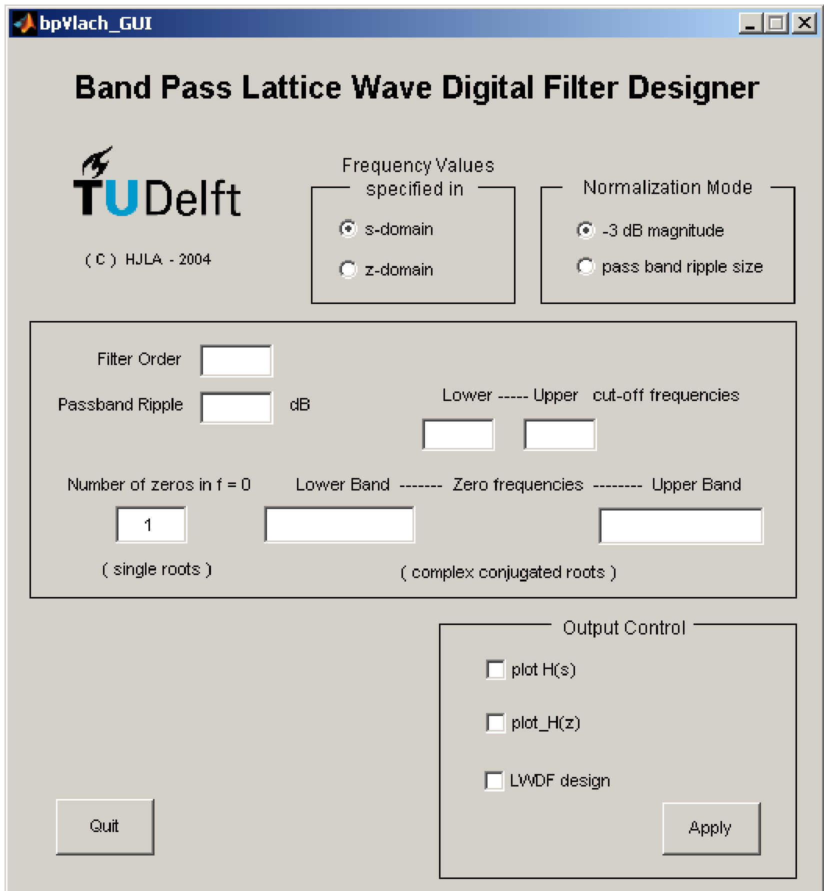
## wdf_GUI

Almost all of the functions listed above are accessible through a Graphical User Interface (GUI), called `wdf_GUI`. A screen shot of this GUI –in startup mode– is shown below.



# bpVlach_GUI

The `bpVlach.m` function is also accessible through a Graphical User Interface (GUI), `bpVlach_GUI`. A screen shot of this GUI –in startup mode– is shown below.



# LWDF_insRegs

**Purpose**

Insert pipeline registers between the slices of an LWDF.

**Syntax**

```
LWDF = LWDF_insRegs(LWDF,regsVec)
[LWDF,Hz] = LWDF_insRegs(LWDF,regsVec)
```

**Description**

`LWDF = LWDF_insRegs(LWDF,regsVec)` is used to insert pipeline registers between the slices of a previously calculated Lattice Wave Digital Filter (LWDF).
At input, `LWDF` is a structure that should contain the fields

`LWDF.wdaCodes` and
`LWDF.gamma`,

while the output LWDF will be extended with the field `LWDF.insRegs`
This field is a copy of the `regsVec` input, which should be a vector of ones and zeros which specify where to insert the registers: a one means that a registers should be inserted at the appropriate place, both in the top and bottom rows.

`[LWDF,Hz] = LWDF_insRegs(LWDF,regsVec)` additionally returns the resulting discrete-time magnitude transfer function `Hz` as a structure (see LWDF2Hz).

**Examples**

**See Also**

| | |
|---|---|
| `Hs2LWDF` | Calculate the coefficients of a Lattice Wave Digital Filter. |
| `showLWDF` | Display the coefficients and the structure of an LWDF. |

# LWDF2cir

**Purpose**

Writes the LWDF structure as a `.cir` description for scheduling purposes.

**Syntax**

```
coeffs = LWDF2cir(LWDF,dlyLorR,cirFilename)
```

**Description**

`coeffs = LWDF2cir(LWDF,dlyLorR,cirFilename)` converts the structure in `LWDF`, which should contain the fields

`LWDF.wdaCodes` and

`LWDF.gamma,` and optionally `LWDF.insRegs`

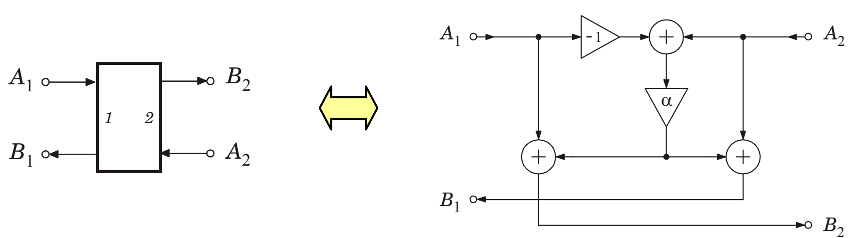
in the `.cir` format used by the scheduling functions and writes it to the file `cirFilename`.

`dlyLorR` should be an '`L`' (for left arm) or '`R`' (for right arm) and specifies the location of the 'interconnect' delay between the 2-port adaptors of a 2nd degree section (if any). If not specified, an '`L`' will be assumed.

If `cirFilename` is not specified, output is written to the command window.
The optional field `LWDF.insRegs` specifies whether and where pipeline registers should be inserted and –if present– consists of a vector of `1`'s and/or `0`'s.
In the cell-array `coeffs`, the operation names assigned in the `.cir` file are linked to the multiplication constants (`LWDF.gamma`) of `LWDF`.

**Notes**

The 2-ports that make up the LWDF are translated into the components shown below. The value of $\alpha$ is the value given in `coeffs`.




**See Also**

| | |
|---|---|
| `Hs2LWDF` | Calculate the coefficients of a Lattice Wave Digital Filter. |
| `showLWDF` | Display the coefficients and the structure of an LWDF. |
| `WDF2cir` | Writes the WDF structure as a `.cir` description. |

# WDF2cir

**Purpose** Writes the WDF structure as a `.cir` description for scheduling purposes.

**Syntax** `coeffs = WDF2cir(WDF,dlyLorR,cirFilename)`

**Description** `coeffs = WDF2cir(WDF,dlyLorR,cirFilename)` converts the structure in `WDF`, which should contain the fields

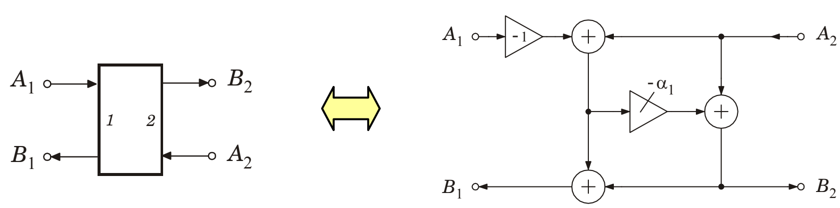
`WDF.wdaStruct`,
`WDF.wdaNo`
`WDF.mulFacs`

in the `.cir` format used by the scheduling functions and writes it to the file `cirFilename`.

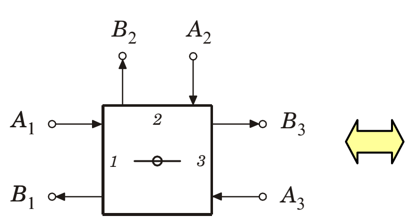
`dlyLorR` should be an '`L`' or '`R`' and specifies the location of the delay when 2-port adaptors are used in the top row (respectively in the left A1 or the right B1 connection between top-row and bottom-row adaptor, defaults to '`L`').

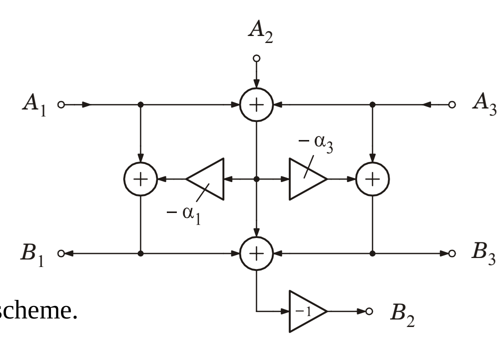
If `cirFilename` is not specified, output is written to the command window.
In the cell-array `coeffs`, the operation names assigned in the `.cir` file are linked to the multiplication constants (`WDF.mulFacs`) of `WDF`.

**Notes** The 3-ports and possibly 2-ports that make up the WDF are translated into the components shown below. The values of α is the value given in `coeffs`.

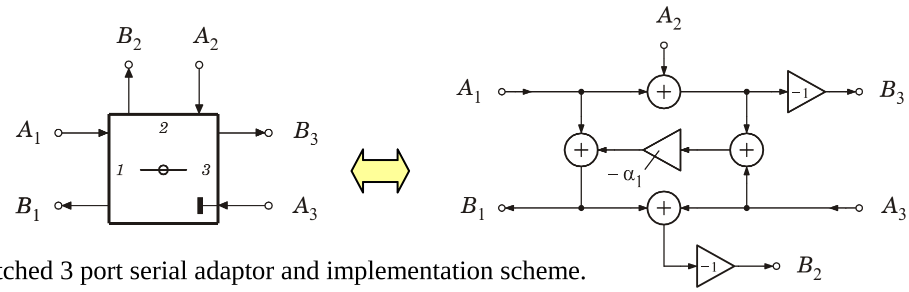



2 port adaptor symbol and implementation scheme.

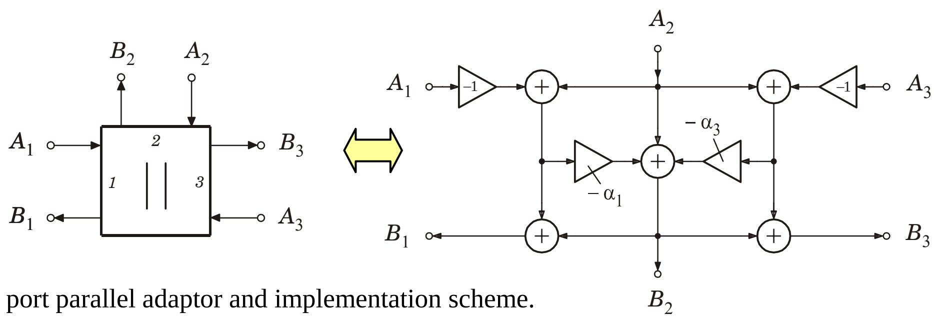


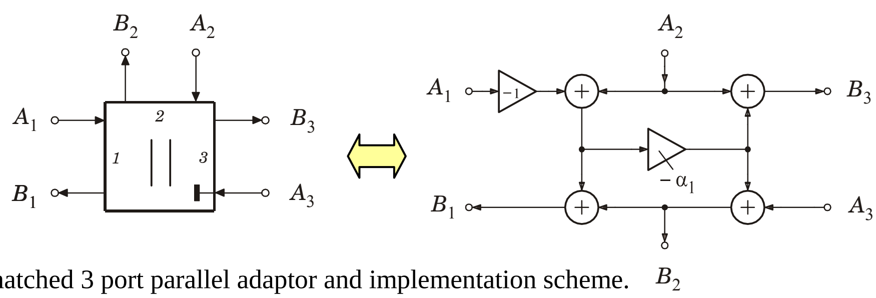

3 port serial adaptor and implementation scheme.




matched 3 port serial adaptor and implementation scheme.




3 port parallel adaptor and implementation scheme.




matched 3 port parallel adaptor and implementation scheme.

In the cir-file, the descriptions for the implementations are optimized in such a way that stand-alone sign-change operations are avoided.

## See Also

| | |
|---|---|
| `ladder2WDF` | Translate a ladder filter into a Wave Digital Filter structure. |
| `showWDF` | Display the coefficients and the structure of WDF. |
| `LWDF2cir` | Writes the LWDF structure as a `.cir` description. |