Netlist Wave Digital Filter Simulator Documentation

1 Introduction

The Netlist Wave Digital Filter (WDF) Simulator is a MATLAB (R2023a) implementation of a circuit simulator based on the Wave Digital Filter method. It can parse circuit descriptions from LTSpice netlist files and simulate the behavior of the circuits.

2 File Organization

The project files are organized as follows:

- `Main.m` and `MainTree.m`: These are the main MATLAB scripts for the application, implementing the two different versions of the program (detailed later).
- `utils` folder: Contains MATLAB utility functions related to parsing the circuit topology and creating/simulating the WDF structure. It is further subdidived into
  - `common`: contains utility functions common to both code versions
  - `main`: contains utility functions for the `Main` version
  - `mainTree`: contains utility functions for the `MainTree` version
- `TriconnectedDecomp` folder: Contains the code implementing the triconnected decomposition algorithm.
- `data` folder: Contains sample netlist files and input/output signals for testing and evaluation.

3 Compilation and Running

The MATLAB scripts can be executed as they are. However, the triconnected decomposition requires a compiled version of the OGDF library[1] and of the C++ MEX interface (implemented in `TriconnectedComponents.cpp`).

[1] `https://github.com/ogdf/ogdf/`

An already compiled minimum version of the OGDF library is found in `TriconnectedDecomp/ogdf/`. Otherwise, if needed, it can be collected from the official repository and compiled (for example using CMake and Visual Studio/XCode or similar tools).

The MEX interface can instead be compiled by launching this command inside the `TriconnectedDecomp` folder:

```
mex -DDEBUG -O -largeArrayDims -Iogdf/include/ -Iogdf/build/include/
  -Logdf -lOGDF -lCOIN TriconnectedComponents.cpp -v
```

where `ogdf` is the folder containing the library source and the compiled version.

4 Adjustable Netlist Variables

At the beginning of both `Main.m` and `MainTree.m`, there are some variables that can be adjusted based on the netlist to be simulated. These variables include:

- `makeGeneratorsReal`: If set to `true`, it will add a negligible resistance value to ideal voltage or current generators in order to make them adaptable.
- `netlistFilename`: Specifies the name of the netlist file (without the file extension).
- `refEdgeId`: Specifies the (netlist) ID of a non-adaptable element in the circuit.
- `outputPortsIds`: Specifies a list of netlist IDs for computing the output voltages.
- `referenceSignalFilenames`: Specifies the filenames of wave files used for result validation. Leave the list empty if no reference file is available.

5 `Main.m` (Single Junction Version)

The `Main.m` script parses an input netlist, creates a WDF structure with a single adapted junction, and simulates the desired output voltages.

5.1 `Main` Utility Functions (`utils/main`)

This section describes the utility functions used in `Main.m`.

5.1.1 `getTopology`

- Inputs:
  - `netlistFilename`: The netlist filename.
- Outputs:
  - `G`: The complete circuit graph resulting from parsing.
  - `B`: The fundamental loop matrix.
  - `Q`: The fundamental cut-set matrix.
  - `orderedEdges`: A list of edges sorted according to the performed tree-cotree decomposition.

The `getTopology` function computes the circuit's topological properties, such as the graph, loop matrix, and cut-set matrix. If the function was already called on a graph which is isomorphic to the current one, it will attempt reusing the previously computed `B` and `Q` matrices (which are stored in an appropriate `.mat` file).

5.1.2 `getZS`

- Inputs:
  - `netlistFilename`: The netlist filename.
  - `B`: The fundamental loop matrix.
  - `Q`: The fundamental cut-set matrix.
  - `G`: The complete circuit graph.
  - `orderedEdges`: A list of edges sorted according to the performed tree-cotree decomposition.
  - `Fs_signal`: The sampling frequency.
  - `refEdgeId`: The reference edge ID.
- Outputs:
  - `Z`: The vector of free parameters.
  - `S`: The scattering matrix for a single junction WDF structure.

The `getZS` function computes the free parameters and scattering matrix for a single junction WDF structure based on the circuit's properties.

Similarly to `getTopology`, it will attempt re-using previously computed `Z` and `S` (this is only possible if the network is isomorphic and both the circuital values and the sampling frequency have not been changed).

6 `MainTree.m` (Multiple Junctions Version)

The `MainTree.m` script is similar to `Main.m`, but it decomposes the reference circuit into parallel, series, or rigid triconnected components. Each of these components becomes a junction in the WDF structure.

6.1 `MainTree` Utility Functions (`utils/mainTree`)

This section describes the utility functions used in `MainTree.m`.

6.1.1 `getZSFromTriconnected`

- Inputs:
  - `T`: Contains the triconnected components.
  - `numEdges`: Total number of edges in the graph.
  - `refEdgeIndex`: The index of the reference edge.
  - `E`: Edges structure containing circuit values and information.
  - `Fs`: The sampling frequency.
  - `endpoints`: Structure containing the nodes adjacent to each edge.
- Outputs:
  - `Tree`: An "augmented" version of the input `T`, containing information needed for the simulation step.
  - `Z`: Vector of computed free parameters.
  - `S`: A collection of scattering rows for all junctions.

The `getZSFromTriconnected` function computes the free parameters and scattering matrix for the WDF structure with multiple junctions based on the triconnected components.

This function works mainly as an interface between the main code and the `handleComponent` recursive function.

The matrix S contains the scattering rows for each edge of the network. Virtual edges need two scattering rows (one for forward scan, one for backward scan). The forward scan row is stored in the Tree structure field `scatteringUp`. Since the number of ports for each junction is not constrained, the matrix S rows should have different number of columns depending on the edge. For simplicity, we allocate a number of columns large enough for the junction with the most ports, and we zero-pad when needed.

6.1.2 `handleComponent`

- Inputs:
  - `T`: Contains the triconnected components.
  - `N`: Number of components.
  - `numEdges`: Total number of edges in the graph.
  - `compIndex`: Index of the current component.
  - `lastParentEdge`: Edge used to reach this component.
  - `E`: Edges structure containing circuit values and information.
  - `Z`: Vector of computed free parameters.
  - `Fs`: The sampling frequency.
  - `depth`: Depth of the current component.
  - `endpoints`: Structure containing the nodes adjacent to each edge.
  - `overallS`: Collection of scattering rows to be filled during execution.
- Outputs:
  - `T`: An "augmented" version of the input `T`, containing information needed for the simulation step.
  - `Z`: Vector of free parameters.
  - `overallS`: A collection of scattering rows for all junctions.

The `handleComponent` function is called by `getZSFromTriconnected` and recursively explores, adapts, and computes the scattering matrix for each triconnected component in a depth-first manner.

When first called by `getZSFromTriconnected`, `compIndex` should be the index of the component containing the non-adaptable element, and `lastParentEdge` the edge corresponding to that component.

7 Common Utility Functions (`utils/common`)

7.1 `graphFromNetlist`

- **Inputs:**
  - `netlistFilename`: The filename of the LTSpice circuit netlist.
- **Outputs:**
  - `G`: The graph representing the circuit.
  - `EdgeTable`: The table containing information about the circuit edges, including end nodes, types, IDs, and values.

The `graphFromNetlist` function parses an LTSpice circuit netlist and returns the graph `G` and the edge table `EdgeTable` representing the circuit. Note: This function uses the `readmatrix` and `graph` functions available in MATLAB to perform the parsing and graph creation.

7.2 getScatteringMatrix

- **Inputs:**
  - `B`: The fundamental loop matrix.
  - `Q`: The fundamental cut-set matrix.
  - `Z`: The vector of free parameters.
- **Outputs:**
  - `S`: The scattering matrix.

The `getScatteringMatrix` function computes the scattering matrix `S` based on the given fundamental loop matrix `B`, fundamental cut-set matrix `Q`, and vector of free parameters `Z`.

7.3 getBQ

- **Inputs:**
  - `G`: The graph representing the circuit.
- **Outputs:**
  - `B`: The fundamental loop matrix.
  - `Q`: The fundamental cut-set matrix.
  - `orderedEdges`: A list of edges sorted according to the performed tree-cotree decomposition.

The `getBQ` function computes the fundamental loop matrix `B`, fundamental cut-set matrix `Q`, and the ordered list of edges `orderedEdges` based on the given circuit graph `G`. The topological matrices are obtained through the tree-cotree decomposition method. We arbitrarily choose to use a minimum spanning tree, which can be obtained using the MATLAB function `minspantree`.

The function creates the `orderedEdges` list by concatenating the edges from the cotree and the tree in the order of their appearance.

7.4 getElementsFunctions

- **Inputs:**
  - `types`: A vector containing the types of elements.
- **Outputs:**
  - `funcs`: An array of function handles describing the scattering relation of the elements.

The `getElementsFunctions` function returns an array of function handles `funcs` that describe the scattering relation of the elements based on the types contained in the input vector `types`.

The function iterates over the elements in `types` and assigns a corresponding function handle to each element type. The scattering relations are defined as anonymous functions that take three arguments: `b` (incident wave), `Vin` (input voltage signal), and `ii` (current sample index). The last two arguments are only needed for sources and are ignored by other elements.

The advantage of previously defining these function handles is that we can significantly improve the simulation time, by avoiding checking the type of an element every time we need to compute its reflected wave.

7.5 `adaptRJunction`

- **Inputs:**
  - `Gprime`: The graph without the edge of the port to adapt (reference edge).
  - `ZParams`: The already computed Z parameters.
  - `refEdgeEndpoints`: The endpoints (nodes) of the reference edge.
  - `graphNodesOrder`: The list of graph nodes, ordered as they are in the graph object.
- **Outputs:**
  - `thevImp`: The free parameter value needed to adapt a port of a generic R-type junction (equivalent Thevenin impedance).

The `adaptRJunction` function computes the free parameter value `thevImp` needed to adapt a port of a generic R-type junction. This value corresponds to the equivalent Thevenin impedance seen at that port.

7.6 `plotResults`

- **Inputs:**
  - `VOut`: The output voltage signals obtained from the simulation.
  - `referenceSignalFilenames`: Filenames of the reference signals for result comparison (optional).
  - `Fs`: The sampling frequency of the signals.

The `plotResults` function plots the results of the simulation. If provided, it will also plot the reference signals specified by `referenceSignalFilenames` for result comparison. In the current version there is no re-sampling step supported, therefore the sampling frequency `Fs` must be the same for all signals (both simulated and reference).