A WAVE DIGITAL FILTER MODELING LIBRARY FOR THE FAUST PROGRAMMING LANGUAGE

Dirk ROOSENBURG (droosenb@oberlin.edu) [1], **Eli STINE** (estine@oberlin.edu) [1], **Romain MICHON** (michon@grame.fr) [2], and **Jatin CHOWDHURY** (jatin@ccrma.stanford.edu) [3]

[1] *TIMARA*, **Oberlin College and Conservatory**, OH USA
[2] **GRAME-CNCM**, Lyon, France
[3] *CCRMA*, **Stanford University**, CA USA

ABSTRACT

In this paper, we present WDmodels, a wave-digital modeling library for the Faust programming language. Recent advancements have made wave-digital models a popular method for simulating analog audio circuits. Despite this, wave-digital modeling techniques have remained challenging to implement for amateurs due to high model complexity. Our library provides a straightforward platform for implementing wave-digital models as real-time digital audio effects.

In this paper, we demonstrate how WDmodels is used to implement wave-digital models containing nonlinear dipoles, such as diodes, and linear R-type adaptors. We describe the library-specific implementation of the connection tree, a data structure commonly used when implementing wave-digital models. We also detail the use of common wave-digital adaptors that have already been implemented in the library. We show how the library may be extended to complex wave-digital models through the implementation of custom adaptors. In order to demonstrate the flexibility of the library, we also present implementations of several audio circuits, including the equalization section of the Pultec EQP-1a program equalizer. Finally, we compare benchmarks from WDmodels and a C++ wave-digital modeling library to demonstrate code efficiency.

1. INTRODUCTION

Faust is a programming language for digital signal processing (DSP) that has grown in popularity in recent years. Its high-level approach to DSP has led to its use by both musicians and experienced DSP programmers [1]. Furthermore, Faust's ability to compile into highly optimized C++ and other low-level coding languages makes it a platform suitable for large, computationally intensive physical models [2].

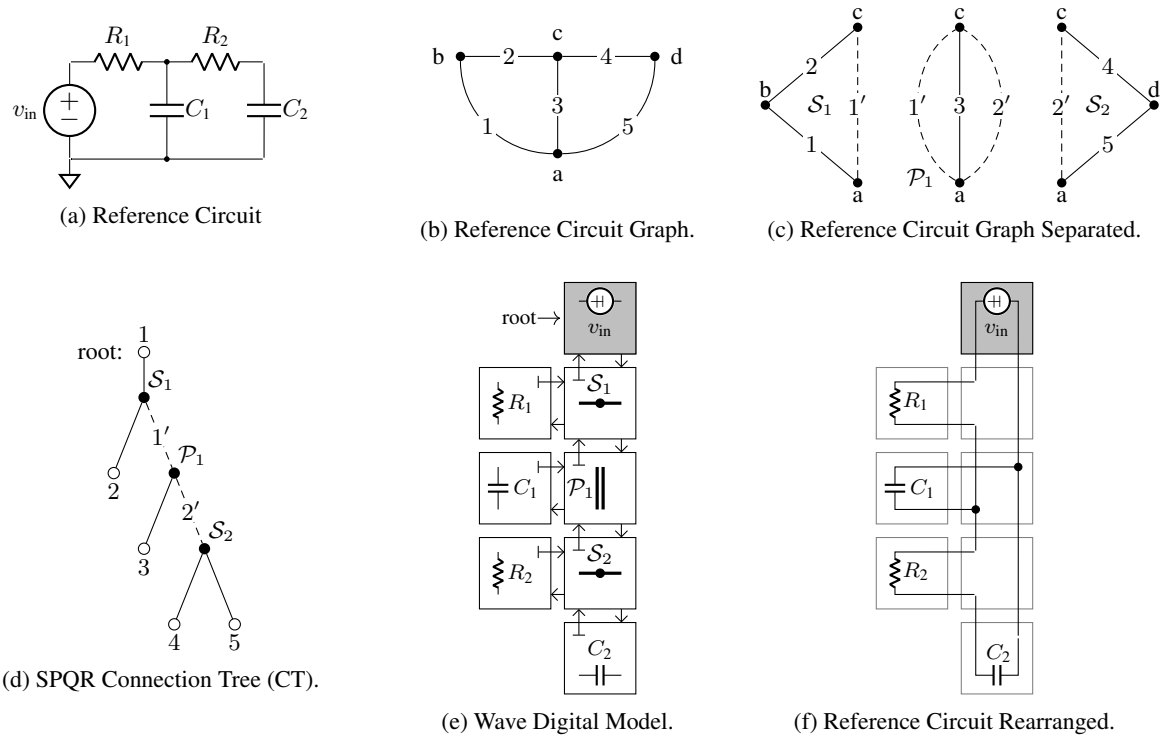
Despite these advantages, Faust's functional method for describing DSP algorithms is incompatible with implementations of physical models that rely on object-oriented data structures. Faust does not currently support direct implementation of multi-directional digital waveguide structures that are commonly found in physical models [3, 4]. To implement a model with multi-directional wave travel in Faust, it must be transformed by inspection into its corresponding direct DSP structure. This process is tedious and unsuitable for large or complex models.

Specific physical modeling methods are supported in Faust through the Faust Libraries [1]. For example, `physmodels.lib` supports the creation of digital-waveguide models of musical instruments by creating custom methods for representing bidirectional traveling waves [5]. `mi.lib` works in conjunction with an external scripting language to generate systems of mass-spring interactions.

In this paper, we present WDmodels, a new addition to the Faust Libraries, that simplifies the creation of wave-digital models of analog audio circuits. Wave-digital models are described by a symbolic representation of the model's connection tree, implemented in Faust using meta-programming. The library uses the Faust compiler to interpret the symbolic connection tree and produce the corresponding direct DSP structure. Many common adaptor types are included in the library, allowing users to easily generate simple models. The library also can be used to create circuit-bendable models of analog-audio circuits for use in real-time processing.

Wave-digital models are a discrete wave-domain representation of physical systems [4]. For circuits, the Kirchoff-based representation is transformed into a traveling-wave-based representation through the bilinear transform. Recent developments extending their capabilities have made wave-digital techniques a popular choice for creating virtual-analog audio effects of analog audio circuits. Their use as a flexible platform for real-time audio simulations has been thoroughly researched [6–8]. Until now, no wave-digital modeling libraries existed in higher-level audio programming languages that are targeted for easy use by artists and musicians. Libraries do exist for C++ [9], a language which is often challenging for beginners, and MATLAB [10], which can be challenging to implement for real-time audio processing. As a result, the modeling technique has remained opaque to many and difficult to learn.

[1] https://github.com/grame-cncm/faustlibraries



(a) Reference Circuit

(b) Reference Circuit Graph.

(c) Reference Circuit Graph Separated.

(d) SPQR Connection Tree (CT).

(e) Wave Digital Model.

(f) Reference Circuit Rearranged.

Figure 1: The process for creating the wave-digital model of a second-order RC lowpass filter using SPQR decomposition.

WDmodels uses a simple process for implementing wave-digital models, allowing those unfamiliar with wave digital modeling to explore the technique for the first time. The block-diagram algebra representation of the DSP processes produced by the library can offer insights into how the compiled model actually functions. The compiled code created by the library runs at speeds comparable to wave-digital model code written in low-level languages. Models may be easily exported to various audio formats for numerous platforms through the `faust2...` tools [11].

In section 2, we provide background information on the creation of connection tree implementations of wave-digital models from analog circuits. Section 3 provides general instructions for using the library. Section 4 introduces two example models and discusses specifics aspects relating to their implementations. Section 5 compares the benchmarking results for WDmodels and a modern wave-digital modeling library written in C++. Section 6 concludes the paper and discusses possible future research directions for this library.

2. BACKGROUND

2.1 Wave-Digital Adaptors

Wave-digital models are networks of connected waveguides called adaptors. When modeling analog circuits, adaptors correspond to parts of the physical system. For example, for each capacitor in the circuit there will be a corresponding adaptor in the wave-digital model. Each adaptor is a wave-scattering junction composed of ports; each port is characterized by an incoming wave, a, a transmitted wave, b, and a port resistance R. The behavior of linear adaptors is described by a scattering equation of the form

$$\mathbf{b} = \mathbf{S}\mathbf{a}, \tag{1}$$

where $\mathbf{b}$ is a vector of transmitted waves, $\mathbf{a}$ is a vector of incident waves, and $\mathbf{S}$ is a scattering matrix. A port of an adaptor is defined to be non-reflective if the port's transmitted wave is not dependent on the current incident wave. An adaptor which includes a non-reflective port is described as "adapted" [4].

The voltage wave definition relates the Kirchhoff behavior of an element to its wave-digital adaptor.

$$\mathbf{b} = \boldsymbol{v} + \mathbf{R}\boldsymbol{i} \tag{2}$$

$$\mathbf{a} = \boldsymbol{v} - \mathbf{R}\boldsymbol{i} \tag{3}$$

Where $\mathbf{v}$ is a vector of voltages across the component, $\mathbf{i}$ is a vector of currents through the component, and $\mathbf{R}$ is a vector of port resistances [12].

Typically, simple circuit components have corresponding one-port adaptors while circuit topology is represented by adaptors with two or more ports. Many linear circuit elements, such as voltage sources, resistors, parallel connections, and series connections, may be digitized directly using the parametric wave definition. Reactive circuit elements are digitized using a conformal frequency mapping, generally the bi-linear transform [8]. The resulting adaptor formulation will rely on sample delay and is said to contain part of state of the system [4].

2.2 The Connection Tree

The connection tree of a model is formed by performing SPQR decomposition on the graph of a circuit. In this process, the circuit's graph is broken into simpler sections by recursively removing series and parallel elements. From

```
1  secondorder(R2, C2, in1) = wd.buildtree(tree)
2  with{
3    //declare components
4    vs1(i) = wd.u_voltage(i, in1);
5    r1(i) = wd.resistor(i, 4700);
6    c1(i) = wd.capacitor(i, 2.2e-6);
7    r2(i) = wd.resistor(i, R2);
8    c2(i) = wd.capacitor_Vout(i, C2);
9    //form connection tree
10   tree = vs1 : wd.series : (r1, (wd.parallel :
       (c1, (wd.series : (r2, c2)))));
11 };
```

Figure 2: The implementation of a second-order RC lowpass filter simulation in Faust using WDmodels. The corresponding wave-digital model is shown in Figure 1

these sections, a tree is formed. Each node in the tree corresponds to an adaptor in the wave-digital model [12, 13]. This process is shown in Figure 1. The leaf nodes (terminating nodes with no downward-going connections) of the connection tree represent circuit components, such as resistors and capacitors. Connection nodes (nodes with both upward-going and downward-going connections) represent circuit topology, denoting elements connected in series or parallel.

Since wave-digital models are a complex network of interconnected scattering junctions, it is critical that adaptors in wave-digital models are arranged in order to prevent delay-free loops within the structure. This process is performed by "adapting" the model, where the port resistances of adaptors are set in order to eliminate these loops by making ports non-reflective. Commonly, this is performed by exploiting the properties of the connection tree. By setting port resistances such that the upward-facing port of each node is non-reflective, this guarantees the resulting structure will contain no delay-free loops. Since the root node is the only node with no upward-facing ports, it is the only node left unadapted. All other nodes within the tree are adapted [14]. Many wave-digital adaptors cannot be adapted, such as ideal voltages sources or nonlinear devices. Thus, an unadaptable adaptor often is chosen as the root of the connection tree.

3. LIBRARY IMPLEMENTATION OVERVIEW

3.1 Approaching Wave-Digital Models in Faust

The library uses meta-programming to simplify implementing a wave-digital model in Faust. For each node in the model's connection tree, a separate function in Faust is declared that describes that node's behavior. To describe the connection tree, the node functions are combined into a single symbolic function using Faust compositional operators. Finally, the symbolic connection tree function is passed to a build-function that generates the model by recursively inspecting the symbolic function.

3.2 Component Declaration

3.2.1 Common Nodes

The library includes many pre-written nodes that correspond to common wave-digital adaptors. These include both nodes of component adaptors, such as resistor, capacitor, and inductor adaptors, and topological adaptors, such as series and parallel adaptors [8, 10]. A current list of all nodes included in the library can be found as part of the Faust Library Documentation [2] .

Code lines 4-8, in Figure 2, shows the declaration of nodes for the component adaptors of the second-order RC lowpass filter; its wave-digital model is shown in Figure 1e. Line 5 declares a resistor adaptor node with a component value of $4.7\,\mathrm{k}\Omega$.

```
1  r1(i) = wd.resistor(i, 4700);
```

Note the parameter i, an index parameter required by the model-building function. Each node in the model must be declared using this form. The prefix `u_` to a node name, as seen in line 4, denotes a node corresponding to the unadapted version of that adaptor.

3.2.2 Model Inputs

Model inputs within the library must be declared explicitly as named parameters of the model. A wave-digital model may receive inputs in the form of voltage, resistance, or any other component value. By declaring inputs as parameters of the enclosing function, they may be called explicitly as component values. This convention is shown in Figure 2; the variables R2, C2, and in1 are model parameters that are used to set component values. Note that the empty signal operator "_" should never be used as a component value, as it will break the internal signal flow of the model.

3.2.3 Model Outputs

Outputs of the model are declared by calling specialized nodes, such as in code line 8, Figure 2.

```
1  c2(i) = wd.capacitor_Vout(i, C2);
```

The model will output the voltage across C_2 as an audio signal.

Nodes which include a model output will have a suffix that describes its output. Only voltage across a component (suffix `_Vout`) and current through a component (suffix `_Iout`) are currently supported as possible outputs from the model.

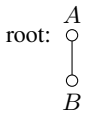
3.2.4 Custom Nodes

Since a wave-digital model might include a specialized adaptor unique to the circuit, the library also includes several functions that help generate nodes from wave-scattering junctions of custom adaptors. Wave scattering junctions can be formed formed according to methods in [3]. Each sequential input-output pair of the scattering junction will correspond to a port of the node. The automatic adapting process requires the scattering junction have the upward-facing port resistances declared as parameters.

`u_genericNode` forms an unadapted node from the scattering junction. `genericNode` forms an adapted node from the scattering junction. The function assumes that the first input-output pair is the non-reflective

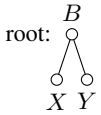
[2] https://faustlibraries.grame.fr/libs/wdmodels/

Connection Tree Structure | **Representation in Faust**



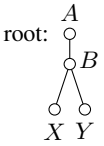

```
1  A : B
```

(a) Simplest two-level tree




```
1  B : (X, Y)
```

(b) Layers with multiple elements




```
1  A : (B : (X , Y))
```

(c) Representation of multi-level trees through subtrees

Figure 3: The meta-programming based representation of the connection tree implemented by the library. The sequence composition denotes a downward going connection. Parallel composition denotes a multi-element layer. Sub-tree based associativity is used to represent multi-level trees.

pair. It also depends on the port resistance rule that creates the non-reflective behavior. `genericNode_Vout` and `genericNode_Iout` form leaf nodes similarly to `genericNode` while also treating the node as a model output.

3.3 Connection Tree Formation

To describe the connection tree, WDmodels implements a custom symbolic representation of trees using existing Faust operators. Sequence composition declares a parent (P) to child (C) relationship between nodes:

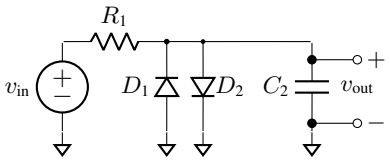
$$P : C. \tag{4}$$

If a parent node has multiple children, they are declared in a list using parallel composition:

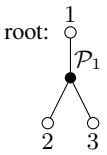
$$P : (C_1, C_2, \ldots C_n) \tag{5}$$

More complex trees are implemented by using this definition recursively. A complex tree can be broken into simple functions representing subtrees, then each subtree function is treated like a single node. This symbolic representation is detailed for several example trees in Figure 3. The declaration of the connection tree for the model in Figure 2 occurs in line 10.

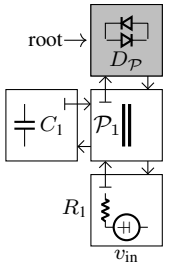
The implemented connection tree must be properly formed along wave-digital model conventions. All nodes in the connection tree must be adapted, except for the root, which must be unadapted. Each node also expects a specific number of parent and child nodes based on its internal



(a) schematic



(b) connection tree



(c) wave-digital model

```
1 diode-clipper(in1) = wd.builtree(tree)
2 with{
3   //declare components
4   c1(i) = wd.capacitor(i, c1)
5   vres(i) = wd.resVoltage(i, in1);
6   dp(i) = wd.u_diodeAntiparallel(i, Is, Vt);
7   //declare connection tree
8   tree = dp : wd.parallel : (c1, vres);
9 }
```

(d) Faust implementation

Figure 4: The wave-digital model and simplified Faust implementation of a one-capacitor diode clipper. The antiparallel diode is modeled using Schottky's diode law and implemented with an iterative Lambert $\mathcal{W}$ function solver.

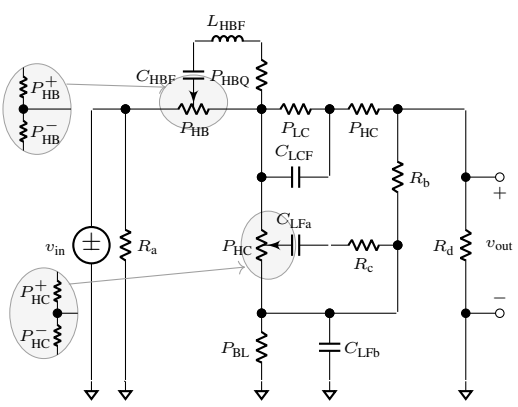
characteristics. For example, an adapted node implementing a three-port parallel adaptor must have two children and one parent in the connection tree.

3.4 Building the Model

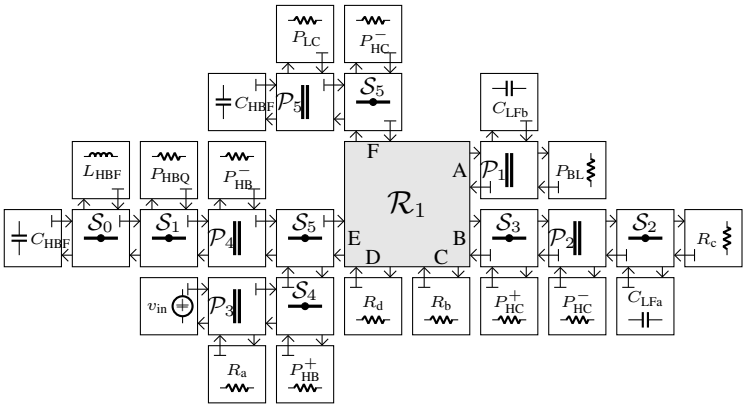
To create a working model, the connection tree function is passed to the model-building function `buildtree`. This step is declared in Figure 2, line 1. `buildtree` interprets the meta-programming of the connection tree and produces the final model function. As part of the model-building process, the wave-digital model is adapted. The port resistances of all adaptors are automatically set to the proper values.

Most implementations of wave-digital filters rely on a tree data structure to implement the connection tree. Adaptors are implemented as node objects in the tree. A recursive tree traversal of the connection tree data-structure is performed for each computation cycle of the connection tree [9, 13].

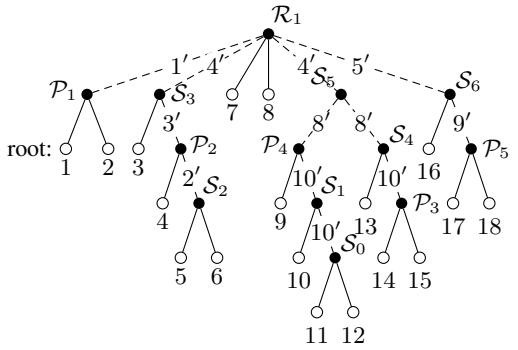
The library uses an alternative implementation of wave-digital models. Instead of performing a tree-traversal at each step, we perform a tree inspection during compilation, generating an instructional method for computing the model. In the tree inspection, the `buildtree` recursively inspects the connection tree function and creates three functions corresponding to the computations of downward-going waves, upward-going waves, and root-



(a) simplified schematic



(b) wave-digital model



(c) connection tree

```
pultec(in1) = wd.buildtree(tree)
with{
  //declare components
  ...
  //declare the R-type by declaring a scattering matrix
  u_6port(i) = u_genericNode(i, scatterJunction)
  with{ scatterJunction(Ra, Rb, Rc, Rd, Re, Rf) =
      ro.matrix(6, 6, scatter)
    with{ scatter = case{
        (1, 1) => S_11
        (1, 2) => S_12
        ...
        (6, 6) => S_66
  };};};
  //declare connection tree
  tree = u_6port : (P_1 : ...), (S_3 : ...), Rb, Rd,
    (S_5 : ...) , (S_6 : ...);
};
```

(d) simplified faust code

Figure 5: The wave-digital model and simplified Faust implementation of the Pultec EQP 1-A passive equalization section. The R-node is implemented by creating 6×6 scattering junction and using library tools to form a custom node.

reflected waves. Each function consists of parametrized node functions and specialized routing for signals.

4. IMPLEMENTATION EXAMPLES

4.1 Diode Clipper

The one-capacitor diode clipper circuit has been thoroughly studied as a virtual analog model [15, 16]. As a wave-digital model, it can be simulated using four adaptors. Figure 4 shows the implementation of a one capacitor diode-clipping circuit as a wave-digital model using WD-models. The resistive voltage source adaptor encloses both v_{in} and R_1. The antiparallel diode pair is modeled using a single nonlinear adaptor chosen as the root of the tree.

The diode pair adaptor is formed using Schottky's ideal diode law, as shown in [17]. The diode adaptor's behavior is described by

$$b = a + 2\text{sgn}(a)\left[RI_s - V_T\mathcal{W}\left(\frac{RI_s}{V_T}e^{\frac{\text{sgn}(a)a+RI_s}{V_T}}\right)\right],$$

where $\mathcal{W}(x)$ is the Lambert $\mathcal{W}$ function, I_s is the diode's saturation current, and V_T is the diode's thermal voltage. The library node which implements an antiparallel diode pair adaptor, `u_diodeAntiparallel`, relies on `lambert`, a custom Lambert $\mathcal{W}$ function approximation that uses Newton-Raphson iteration to approximate the solution [18]. Since compiled Faust code cannot contain loops, `lambert` uses a set number of iterations that will be performed at each sample.

4.2 Pultec EQP-1A

The library also provides several functions which allow the implementation of user-generated adaptors as nodes. This allows for the simulation of complex circuits through the implementation of R-type adaptors.

Here we present an implementation of the Pultec EQP-1a's passive equalization section. The EQP-1a is a program equalizer popular with audio engineers for master bus equalization and general mastering [19]. The equalization is performed by an passive RLC network. The output is then passed through a tube makeup-gain stage.

We implemented the passive RLC network as a wave-digital model as shown in Figure 5. Figure 5a shows the simplified schematic of the EQP 1-A RLC equalization network. By performing SPQR decomposition, we found the connection tree associated with the circuit, shown in Figure 5c. The circuits equivalent wave-digital model is shown in Figure 5b. A resistive voltage source with negligible series voltage was chosen to model the voltage input.

The scattering matrix of this R-type adaptor was derived using methods described in [12]. To implement this adaptor as a node in the library, we used `u_genericNode`. First the adaptor's scattering matrix was used to form a

	C++ WDF		WDModels	
	time (s)	ratio	time (s)	ratio
Second-Order	0.197	50	0.0746	130
Large Network	0.606	16	0.162	61
Diode Clipper	0.176	57	0.507	20

Figure 6: A comparison of computation benchmarks for WDmodels and a C++ wave-digital modeling library. Time to compute for each was averaged over three runs. The ratio of real time to computation time is also displayed for comparison purposes; higher is better.

6×6 scattering junction, `scatterJunction`, using methods in [3]. The six upward-facing port resistances $(R_a, R_b, \ldots, R_f)$ were declared as parameters. The scattering junction was passed to `u_genericNode` to form it into an unadapted node.

5. COMPARISON

To determine both the realizability and optimization of the WDmodels, we benchmarked it against a personal wave-digital modeling library [3] written in C++. The benchmarks were performed on a desktop PC with an AMD Ryzen 2600x processor and 16GB of RAM running Manjaro Linux. Three models were tested: the second-order RC lowpass filter shown in Figure 1e, the diode clipper shown in Figure 4c, and an arbitrary large linear model. The three models were first implemented in both libraries. For the Faust library, the Faust code was then compiled into C++. Each implementation was then tested to determine the time it took to process $10\,\mathrm{sec}$ of randomized audio at $192\,\mathrm{kHz}$ sample rate. A high sample rate was chosen to show the potential for oversampling, as oversampling is commonly used to improve the accuracy of physical models. Three runs for each implementation were performed and times were averaged together to determine a mean computation time for each implementation. The results are displayed in Figure 6. The full code used for testing is available on GitHub [4].

Our benchmarks show that both libraries produce simulations capable of easily running in real time at high sample rates. For the linear networks, WDmodels outperforms the C++ library by a factor of 2-3. This can be attributed to the Faust compiler, which is designed to produce highly optimized DSP processes. It should be noted that the C++ library used is already optimized through templating; comparison to C++ libraries that perform recursion through the model's connection tree in real time would likely have slower performance.

WDmodels is outperformed by the C++ library on the diode clipper by a factor of about 3. This is likely due the C++ library using a more optimized method for computing the Lambert $\mathcal{W}$ function. Further optimization `lambert` would be helpful to improve the performance of the diode clipper.

[3] https://github.com/Chowdhury-DSP/chowdsp_utils
[4] https://github.com/jatinchowdhury18/wdf-bakeoff

These benchmarks show that WDmodels is an excellent platform for real-time simulation using wave-digital models. The highly optimized code produced by the Faust compiler is suitable for use in digital audio effects or other real-time applications.

6. CONCLUSION

In this paper, we presented a new Faust library, WDmodels. The library greatly simplifies the process of implementing wave-digital models in Faust by creating a symbolic representation of the connection tree data structure using meta-programming. We explained how the symbolic representation is used within the context of the library and showed examples that demonstrate its use.

Typical wave-digital modeling libraries rely on recursion through a tree for the computation of each sample, which can be computationally expensive. In the library, a tree traversal is only performed once at compilation instead of every sample at runtime, resulting in a significant reduction of computational complexity of the model. We have also shown that the Faust compiler is able to produce C++ code that rivals or outperforms C++ wave-digital modeling libraries, in some cases. The code produced is suitable for implementation in real-time digital audio effects.

Currently, the library only includes nodes for some one-port nonlinearties, specifically the diode. Most complex audio circuits rely on complex nonlinear elements, such as transistors and vacuum tubes, which are modeled using multiport nonlinearties [20]. Thus, the development of methods to support multiport nonlinear adaptors in faust would greatly widen the scope of models which can created with default library nodes and functions.

Acknowledgments

Thanks to Kurt Werner for assisting with wave-digital model figures and to Rob Owen, Julius O. Smith, Stéphane Letz, and Yann Oraley for aiding with the development of the library.

7. REFERENCES

[1] R. Michon and Y. Orlarey, "The faust online compiler: a web-based ide for the faust programming language," in *Linux Audio Conference*, 2012. [Online]. Available: http://lac.linuxaudio.org/2012/papers/22.pdf

[2] Y. Orlarey, D. Fober, and S. Letz, "Faust: An efficient functional approach to dsp programming," 2009, unpublished. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02159014

[3] J. O. Smith III, *Audio Signal Processing in Faust*, CCRMA, Stanford, CA, USA, 2020. [Online]. Available: https://ccrma.stanford.edu/~jos/aspf

[4] ——, *Physical Audio Signal Processing*, 1st ed. http://www.w3k.org/books/: W3K Publishing, 2010.

[5] R. Michon, J. Smith, C. Chafe, G. Wang, and M. Wright, "The faust physical modeling library: a

modular playground for the digital luthier," in *International Faust Conference*, 2018.

[6] J. Zhang and J. O. Smith III, "Real-time wave digital simulation of cascaded vacuum tube amplifiers using modified blockwise method," in *Proc. 21th Intl. Conf. Digital Audio Effects (DAFx-18)*, 2018.

[7] J. Chowdhury, "A comparison of virtual analog modelling techniques for desktop and embedded implementations," 2020. [Online]. Available: https://arxiv.org/pdf/2009.02833.pdf

[8] K. J. Werner, "Virtual analog modeling of audio circuitry using wave digital filters," Ph.D. dissertation, Stanford University, 2016.

[9] M. Rest, R. W. Dunkel, K. J. Werner, and J. O. Smith III, "RTWDF—a modular wave digital filter library with support for arbitrary topologies and multiple nonlinearities," in *Proc. 19th Intl. Conf. Digital Audio Effects (DAFx-16)*, 2016.

[10] U. Zölzer, X. Amatriain, D. Arfib, J. Bonada, G. De Poli, P. Dutilleux, G. Evangelista, F. Keiler, A. Loscos, D. Rocchesso *et al.*, *DAFX-Digital Audio Effects*, 2nd ed. John Wiley & Sons, 2011.

[11] D. Fober, Y. Orlarey, and S. Letz, "Faust architectures design and osc support." in *Proc. 14th Intl. Conf. Digital Audio Effects (DAFx-11)*, 2011, pp. 231–236.

[12] K. Werner, A. Bernardini, J. Smith, and A. Sarti, "Modeling circuits with arbitrary topologies and active linear multiports using wave digital filters," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. PP, pp. 1–14, 06 2018.

[13] D. Franken, J. Ochs, and K. Ochs, "Generation of wave digital structures for networks containing multiport elements," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 52, no. 3, pp. 586–596, 2005.

[14] A. Sarti and G. De Sanctis, "Systematic methods for the implementation of nonlinear wave-digital structures," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 2, pp. 460–472, 2008.

[15] J. Parker, F. Esqueda, and A. Bergner, "Modelling of nonlinear state-space systems using a deep neural network," in *Proc. 23rd Intl. Conf. Digital Audio Effects (DAFx-19)*, 2019.

[16] D. T. Yeh, J. S. Abel, and J. O. Smith, "Automated physical modeling of nonlinear audio circuits for real-time audio effects—part i: Theoretical development," *IEEE transactions on audio, speech, and language processing*, vol. 18, no. 4, pp. 728–737, 2009.

[17] K. J. Werner, V. Nangia, A. Bernardini, J. O. Smith III, and A. Sarti, "An improved and generalized diode clipper model for wave digital filters," in *Audio Engineering Society Convention 139*. Audio Engineering Society, 2015.

[18] S. D'Angelo, L. Gabrielli, and L. Turchet, "Fast approximation of the lambert w function for virtual analog modelling," *Proc. 23rd Intl. Conf. Digital Audio Effects (DAFx-19)*, vol. 100, 2019.

[19] *Pultec model EQP-1A Manual*, Pulse Techniques, Inc., West Englewood, NJ, 1951.

[20] K. J. Werner, V. Nangia, J. O. Smith, and J. S. Abel, "A general and explicit formulation for wave digital filters with multiple/multiport nonlinearities and complicated topologies," in *2015 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. IEEE, 2015, pp. 1–5.