Recent Progress in Wave Digital Audio Effects
WDF Software Overview and Demo

Julius Orion Smith and Kurt James Werner
CCRMA, Stanford University

Digital Audio Effects (DAFx) Conference

Norwegian University of Science and Technology
Trondheim, Norway

November 30 — December 3
2015

Free WDF Software

Free Software for Wave Digital Filtering (WDF)

- Object-oriented Matlab in the book
  DAFX: Digital Audio Effects - 2nd Edition, 2011
- C++ inspired by the DAFx-book, posted to the JUCE Forum under “Useful Tools and Components” by `maxprod`
- Wave Digital Filter Framework in C++ (coming soon)

DAFx-Book Matlab

DAFx Book Wave Digital Filter Software

Object-oriented Matlab in the second DAFx Book:

DAFX: Digital Audio Effects - 2nd Edition, 2011

Free software download from `www.dafx.de`:

- Click on "DAFX - Digital Audio Effects (Second Edition)" in the upper-right corner of the main DAFx website `www.dafx.de`
- Click on "`Matlab Files`"
- Click on "`Matlab for Chapter 12`"
- Open "`WDFDiodeExample.m`"
- Try it out in Matlab or Octave!
- License as of 2015-11-28:

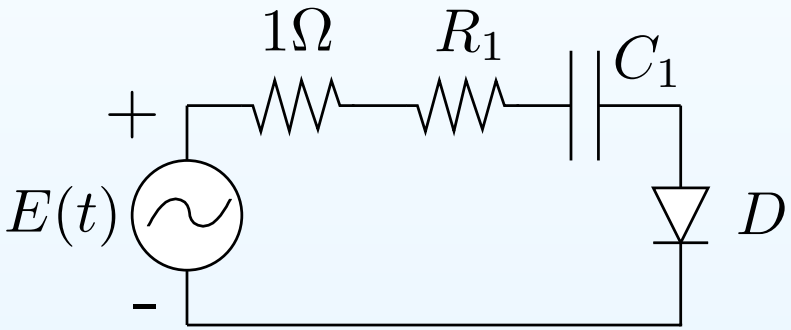
> *"It may be used for educational purposes and not for commercial applications without further permission."*
>
> (Copyright Wiley & Sons 2011)

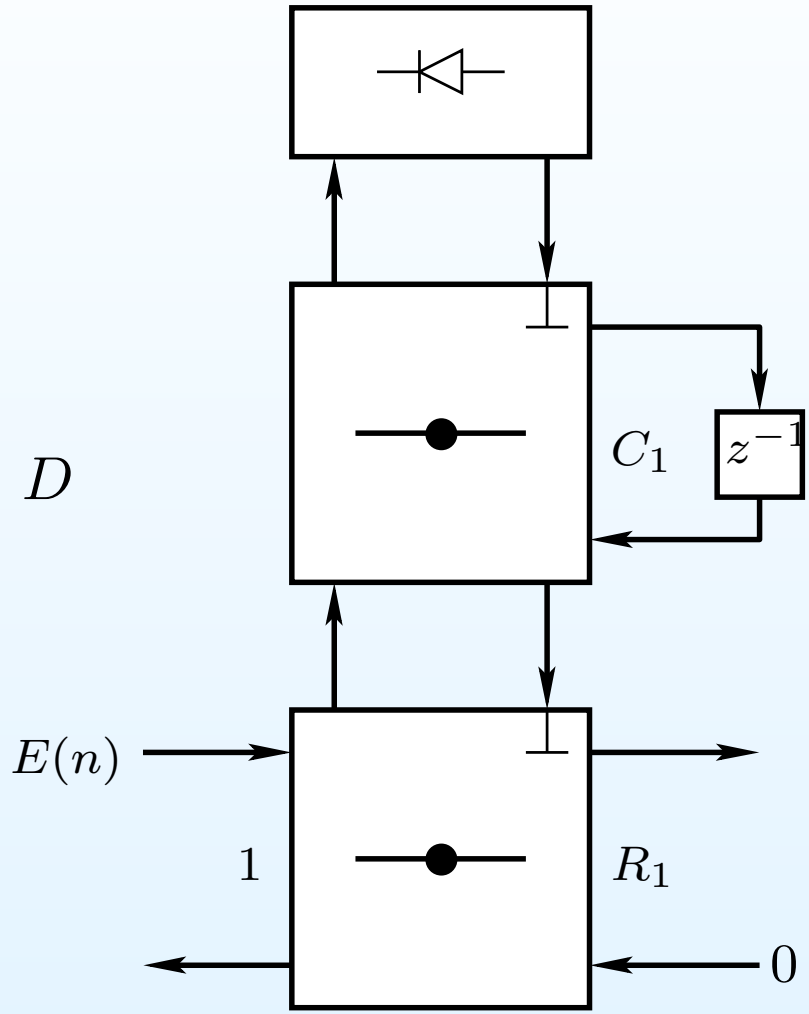
DAFx-Book Wave Digital Filter Software, Continued

Object-oriented Matlab for a simple WDF diode circuit:

- Influential but not Complete:
- Includes:
  - Wave-digital capacitor and resistor
  - Series three-port adaptor
  - Binary Connection Tree
- Left as an exercise for the reader:
  - Parallel adaptors
  - Inductors
  - Memoryless nonlinearity
  - Adapted voltage source
- Not yet known at the time:
  - R-node scattering junctions
  - Multiple nonlinearities

DAFx Book: Circuit Example





DAFx Book Matlab: WDFDiodeExample.m

```
...
% Build the circuit:
V1 = V(0,1); % voltage source with 0 initial voltage & 1-Ohm resistor
R1 = R(80);  % 80 Ohm resistor R1
CapVal = 3.5e-5; % capacitance value in Farads
C1 = C(1/(2*CapVal*Fs));  % Capacitor C1
s1 =  ser(V1,ser(C1,R1)); % WDF tree = series connection of V1,C1,R1
Vdiode = 0; % initial value for the voltage over the diode

% Simulation loop:
for n = 1:N % n = time in samples
  V1.E = input(n); % input signal = voltage source
  WaveUp(s1);      % propagate waves from leaves to root of WDF tree
  Rdiode = 125.56*exp(-0.036*Vdiode); % diode = nonlinear resistor
  r = (Rdiode-s1.PortRes)/(Rdiode+s1.PortRes); % dynamic reflectance
  s1.WD = r*s1.WU; % wave leaving the diode (at root of WDF tree)
  Vdiode = (s1.WD+s1.WU)/2; % current diode voltage
  output(n) = Voltage(R1);  % output = voltage across resistor R1
end;
```

DAFx Book Matlab: WDFClasses.m

```
...
%----------------------WDF Class-----------------------
classdef WDF < hgsetget % WDF base class < Handle Graphics Get/Set
  properties
    PortRes % WDF port resistance
  end
  methods
    function Volts = Voltage(obj)    % voltage (V) across WDF element
      Volts = (obj.WU + obj.WD) / 2; % WDF wave variables to voltage
    end
  end;
end

%--------------------Adaptor Class----------------------
classdef Adaptor < WDF % parent of series and parallel 3-port adaptors
  properties
    KidLeft  % handle to the WDF element connected at the left port
    KidRight % handle to the WDF element connected at the right port
  end;
end
```

DAFx Book Matlab: WDFClasses.m, Continued

```
%----------------------Ser Class-----------------------
classdef ser < Adaptor % series 3-port adaptor
  properties
    WD % down-going wave at the adapted port
    WU % up-going wave at the adapted port
  end;
  methods
    function obj = ser(KidLeft,KidRight) % constructor function
      obj.KidLeft = KidLeft; % connect the left 'child'
      obj.KidRight = KidRight; % connect the right 'child'
      obj.PortRes = KidLeft.PortRes+KidRight.PortRes; % adapt. port
    end;
    function WU = WaveUp(obj) % the up-going wave at the adapted port
      WU = -(WaveUp(obj.KidLeft)+WaveUp(obj.KidRight)); % wave up
      obj.WU = WU;
    end;
    ...
  end
end
```

DAFx Book Matlab: WDFClasses.m, Continued

```
%----------------------Ser Class-----------------------
classdef ser < Adaptor % series 3-port adaptor
  properties
  ...
  methods
    ...
    function set.WD(obj,WaveFromParent) % sets the down-going wave
      obj.WD = WaveFromParent; % set down-going wave for the adaptor
      % set waves to 'children' according to the scattering rules:
      set(obj.KidLeft,'WD',obj.KidLeft.WU-(obj.KidLeft.PortRes/...
        obj.PortRes)*(WaveFromParent+obj.KidLeft.WU+obj.KidRight.WU));
      set(obj.KidRight,'WD',obj.KidRight.WU-(obj.KidRight.PortRes/...
        obj.PortRes)*(WaveFromParent+obj.KidLeft.WU+obj.KidRight.WU));
    end;
  end
end
```

JUCE WDF C++

JUCE WDF Software

C++ inspired by the DAFx-book, posted to the JUCE Forum under "Useful Tools and Components" by `maxprod`:

`http://www.juce.com/forum/topic/wave-digital-filter-wdf-juce/`

`http://www.juce.com/forum/topic/-wdf-new-restructuration-project-audio-processor`

(Search the JUCE Forum for 'WDF')

- Compiles and runs on Mac OS X after changing Build Settings to use OSX SDK 10.9 instead of 10.10 (else "Point" not defined)
- Should work on Linux and anywhere else supported by JUCE (AU, VST, . . .)
- Nice hierarchical object-oriented class structure (like the Matlab example from DAFx Book II)
- Suitable for real-time deployment (unlike Matlab code)
- Similarly needs upgrades

JUCE WDF Software Copyright

Copyright license is unclear:

- The word “copyright” does not appear anywhere in the Source
- In the second JUCE Forum post, `maxprod` says

> “Feel free to use the WDF++ project, it’s for the humanity benefice.”

A definite copyright choice would be preferred!

JUCE WDF Software Copyright, Continued

It is probably safest to treat the JUCE Forum code from `maxprod` as GNU GPL, since this is typical for JUCE modules:

- `README.txt` for the JUCE Library, downloaded 2014-12-13 from `http://www.juce.com`:

  > "Most JUCE modules are shared under the GNU Public Licence (GPLv2, v3, and the AGPLv3). This means that the code can be freely copied and distributed, and costs nothing to use in other GPL applications. One module (the juce_core module) is permissively licensed under the ISC."

- Could JUCE Forum posts have an implied copyright license?

Preferred License = MIT

*Opinion: **MIT License** is the best choice*

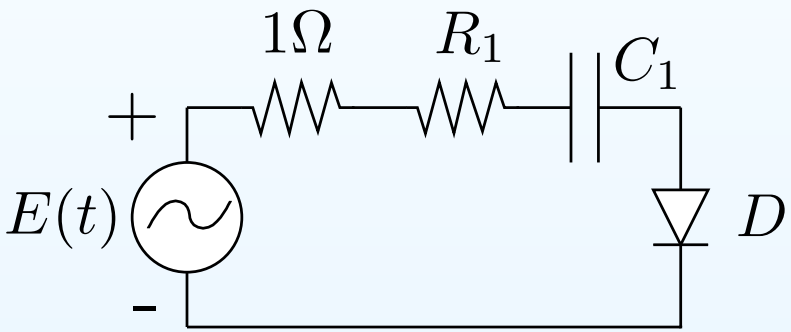
- Anyone can use the software for whatever, even at companies
- We can fully share our software, even in our consulting work
- Companies don't have to disclose it (marketing decision)
- Bugs more likely to be found and fixed
- One can supplement the MIT License to request (nonbinding) contribution of bugfixes and general-purpose extensions

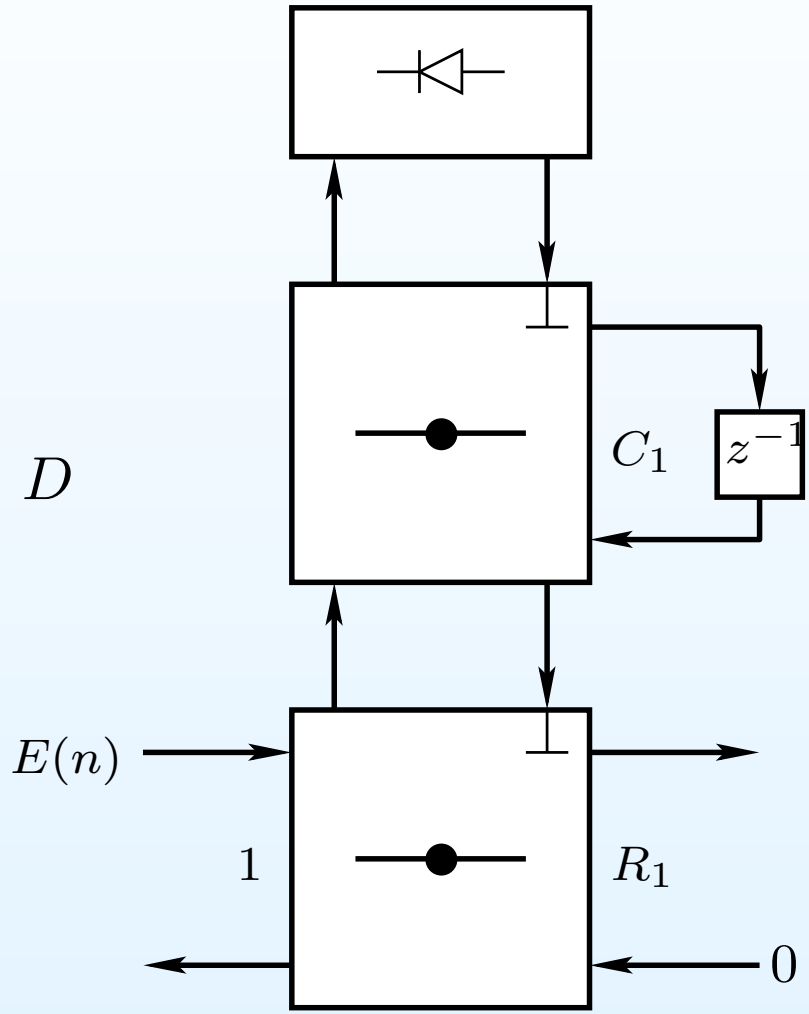
FAUST License $\approx$ MIT

The FAUST project licenses are generally close to the MIT License:

- Enables companies to use *and contribute to* the FAUST project
- FAUST's consulting-compatible licensing led to many contributions in
  - `oscillator.lib`
  - `filter.lib`
  - `effect.lib`
- **Reason:** Companies can easily afford to contribute
  - bugfixes and incremental extensions
  - *not* total rewrites from scratch

Reminder of the DAFx-Book Diode Circuit Example





JUCE WDF Code: DiodeWDF.h: Build and Run Diode Circuit in C++

```
  ...
//Build Circuit:
  VoltageSource Vin(0, 1);  // initial voltage 0V, source R = 1 Ohm
  Resistor R1(80.0);
  Capacitor C1(3.5e-5, Fs); // Cap. & Inductor need sampling rate Fs
  Series RC(&R1, &C1);      // R1 and C1 are connected in series
  Series root(&Vin, &RC);   // Voltage src and RC ckt also in series
  ...
//Simulation Loop:
  for (int n=0; n<max; n++) { // for each time sample
    Vin.Vs = input[n];    // input signal -> voltage source
    b = root.reflected(); // propagate waves from leaves to root
    Rdiode = Is * exp(-Vt * Vdiode); // diode resistance (FIXME)
    r = (Rdiode - root.R)  // update dynamic diode reflectance
      / (Rdiode + root.R); // (still have an extra sample of delay)
    root.incident(r * b);  // reflected wave from diode (at root)
    Vdiode = root.voltage();  // update current diode voltage
    output.add(R1.voltage()); // output = voltage across resistor R1
  }
  return output;
  ...
```

JUCE WDF Code (WDF.h) WDF Class

```
class WDF { // Every element and every adaptor is a subclass of WDF
 public:
  WDF (double Rp, std::string varName="WDF")
    : R(Rp), G(1/Rp), a(0), b(0), name(varName) {}

  virtual inline double reflected () = 0; // pure virtual function
  virtual inline void incident (double value) = 0; // (WDF abstract)

  double voltage () { // v
    return 0.5 * (a + b); // so traveling voltage-waves are a/2, b/2
  }
  double current () { // i
    return 0.5 * (a - b) * G; // (vp-vm)/R in waveguide variables
  }
  double R; // WDF port resistance (top port for adaptors)
  double G; // conductance G = 1/R = inverse port resistance
  double a; // incident WDF wave variable (incoming voltage wave * 2)
  double b; // reflected WDF wave variable (outgoing voltage wave * 2)
  std::string name; // same as the variable name used, for debugging
}
```

JUCE WDF Code (WDF.h) Resistor

```
class Resistor : public WDF {

 public:

  Resistor (double R, std::string varName="R") : WDF (R,varName) {}

  virtual inline double reflected () {
    b = 0;
    return b;
  }

  virtual inline void incident (double value) {
    a = value;
  }

}
```

JUCE WDF Code (`WDF.h`) Short Circuit

```
class ShortCircuit : public WDF {
 public:
  ShortCircuit (double R) : WDF (R) {}
  virtual inline double reflected () { // tree evaluation to root
    b = -a;   // a = incoming voltage wave / 2
    return b; // voltage waves sign-flip when reflecting from a short
    // Note one-sample delay: a was written a sample ago
  }
  virtual inline void incident (double value) {
    a = value; // set when pushing down the tree from the root
  }
}
```

Better to use an *adapted* "approximate short-circuit" here:

- Retain microscopically small resistance $R = \epsilon$
- Set the connecting adaptor port resistance to ϵ
- Inverting reflection now happens at the adaptor
- No reflection from the near-short-circuit itself

JUCE WDF Code (WDF.h) Open Circuit

```
class OpenCircuit : public WDF {
 public:

  OpenCircuit (double R)
    : WDF (R) {}

  virtual inline double reflected () {
    b = a;
    return b;
  }

  virtual inline void incident (double value) {
    a = value;
  }
}
```

JUCE WDF Code (WDF.h) Voltage Source

```
class VoltageSource : public WDF {
 public:

  VoltageSource (double V, double R, std::string varName = "Vs")
    : Vs (V), WDF (R, varName) {}

  virtual inline double reflected () {
    b = -a + 2.0 * Vs;
    return b;
  }

  virtual inline void incident (double value) {
    a = value;
  }

  double Vs;
}
```

JUCE WDF Code (WDF.h) Current Source

```
class CurrentSource : public WDF {
 public:

  CurrentSource (double I, double R, std::string varName="Is")
    : Is (I),
    WDF (R,varName) {}

  virtual inline double reflected () {
    b = a + 2.0 * R * Is;
    return b;
  }

  virtual inline void incident (double value) {
    a = value;
  }

  double Is;
}
```

JUCE WDF Code (WDF.h) Capacitor

```
class Capacitor : public WDF {
 public:

  Capacitor (double C, double T, std::string varName="C")
    : WDF (T/2.0*C, varName),
    state (0) {}

  virtual inline double reflected () {
    b = state;
    return b;
  }

  virtual inline void incident (double value) {
    a = value;
    state = value;
  }

  double state;
}
```

JUCE WDF Code (WDF.h) Inductor

```
class Inductor : public WDF {
 public:

  Inductor (double L, double T, std::string varName="L")
    : WDF (2.0*L/T,varName),
    state (0) {}

  virtual inline double reflected () {
    b = -state;
    return b;
  }

  virtual inline void incident (double value) {
    a = value;
    state = value;
  }

  double state;
}
```

JUCE WDF Code (WDF.h) WDF Adaptor Class

```
class Adaptor : public WDF {
 public:
  WDF *left;    // WDF element connected at the left port
  WDF *right;   // WDF element connected at the right port

  Adaptor (WDF *l, WDF *r, double R, std::string varName)
    : left (l), right (r), WDF (R, varName) {}

  virtual inline double reflected () = 0;

  virtual inline void incident (double wave) {
    // set the waves to the children according to the scattering rules
    left->incident  ( left->b - ( left->R / R)
                  * (wave + left->b + right->b));
    right->incident (right->b - (right->R / R)
                  * (wave + left->b + right->b));
    a = wave;
  }
}
```

JUCE WDF Code (WDF.h) Series Adaptor

```
class Series : public Adaptor {
 public:
  Series (WDF *l, WDF *r, std::string varName="SeriesAdaptor")
    : Adaptor (l, r, (l->R + r->R), varName) {}
  virtual inline double reflected () {
    b = -(left->reflected() + right->reflected());
    return b;
  }
}

class Parallel : public Adaptor {
 public:
  Parallel (WDF *l, WDF *r, std::string varName="ParallelAdaptor")
    : Adaptor (l, r, (l->R * r->R / (l->R + r->R)), varName) {}
  virtual inline double reflected () {
    b = (left->G/G) * left->reflected()
     + (right->G/G) * right->reflected();
    return b;
  }
}
```

WDF Framework C++

WDF Framework

"Coming soon" is the Wave Digital Filter Framework:

- Written in C++ at CCRMA by Visiting Researcher Maximilian Rest from TU Berlin <m.rest@mailbox.tu-berlin.de>
- Assistance from CCRMA MA/MST student Ross Dunkel
- Includes support for general R-node scattering matrices
- Soon to be released as Free Open-Source Software (FOSS) under a suitable license

WDF Framework Adapted Voltage Source

- An *adapted* voltage source has a source resistance equal to the resistance of its attached WDF adaptor port
- This results in *no reflection* from the voltage source, so it can be at any leaf node of the WDF tree

```
class wdfResVolt : public wdfLeaf {
 public:
  double Vs;         // Voltage-source voltage
  double RSer;       // Voltage-source internal resistance
  const double Rp;   // Resistance of attached port

  inline wdfResVolt (double Vs, double RSer)
         : Vs (Vs), Rp (RSer), RSer (RSer), wdfLeaf (0) { }
  inline double calculateUpRes(double T) { return Rp; }
  inline double calculateUpB(void) { return Vs; }
  inline void calculateDownB(double wave) { } // incoming wave absorbed
};
```

WDF Framework R-Node

```
class wdfRootRtype : public wdfRoot {
 public:
  int num_ports;
  double (**S);
  inline wdfRootRtype (std::vector<wdfNode*> childNodes)
    : wdfRoot (childNodes), num_ports (childNodes.size()) {
    upPort = NULL;
  }

  inline void initScatterMatrix(double (**S)) {
    this->S = S;
  }

  inline double calculateUpB(void) {
    fprintf(stderr,"*** wdfRootRtype: call processWaves()\n");
  }

  inline void processWaves(void) { ... <next slide> ... }
};
```

WDF Framework R-Node, Continued

```
class wdfRootRtype : public wdfRoot {
 public:
  ...
  inline void processWaves(void) {
    int i = 0; // row index for R-node scattering matrix S(i,j)
    int j = 0; // col index for R-node scattering matrix S(i,j)
    double b = 0; // outgoing wave variable
    for (wdfPort* dpb : downPorts) {
      b = 0;
      j = 0;
      for (wdfPort* dpa : downPorts) {
        b += dpa->a * S[i][j]; // scattering mtx times incoming waves
        j++;
      }
      dpb->b = b; // outgoing wave on i'th output port
      i++;
    }
  }
};
```

Real-Time WDF Demo

Live WDF Demonstration in GeoShred for iPad!

The Tube Screamer is a classic distortion unit:

- The Tube Screamer C++ WDF requires around **12% CPU** at a 44.1 kHz sampling rate on an iPad 3
- Contains one nonlinearity (two diodes) — no R-nodes

Following is a demonstration of a C++ WDF Tube Screamer using the upcoming app **GeoShred** for iPad

Conclusions

Experience to date indicates:

- Wave Digital Filters (WDFs) are an excellent choice for accurate real-time digital modeling of analog audio effects
- Computational performance and accuracy are highly competitive with other approaches
- A C++ class hierarchy is a clear and natural choice of software organization
- The *Binary Connection Tree* (BCT) implementation of WDFs requires only three-port WDF *adaptors*
- Extending BCTs to include more general scattering matrices (for R-type nodes such as bridge circuits) works out fine