# Ciaramella: a synchronous data flow programming language for audio DSP

**P. Marrone, S. D'Angelo, F. Fontana, G. Costagliola, G. Puppis**

Orastron srl & Università di Udine & Università di Salerno

https://ciaramella.dev

## DESIGN GOALS

- Declarative (SSA form)
- Synchronous Data Flow (SDF) [2]
- Strictly audio DSP oriented
- Minimal keywords set
- Unconstrained and conventional syntax
- Composability and modularity

## CIARAMELLA

### CONCEPTS

- **Block**: encapsulates an operation. It can be **atomic** or **composite** if contains other blocks. It is the equivalent of the SDF *actor*. Built-in blocks: arithmetic operations and **delay1** block.
- **Port**: a communication endpoint attached to a block. It can be of input or output type.
- **Connection**: a directed flow of data (a signal) between two ports.

## ZAMPOGNA

Compiler written in JavaScript (~1000 line of code). Steps:

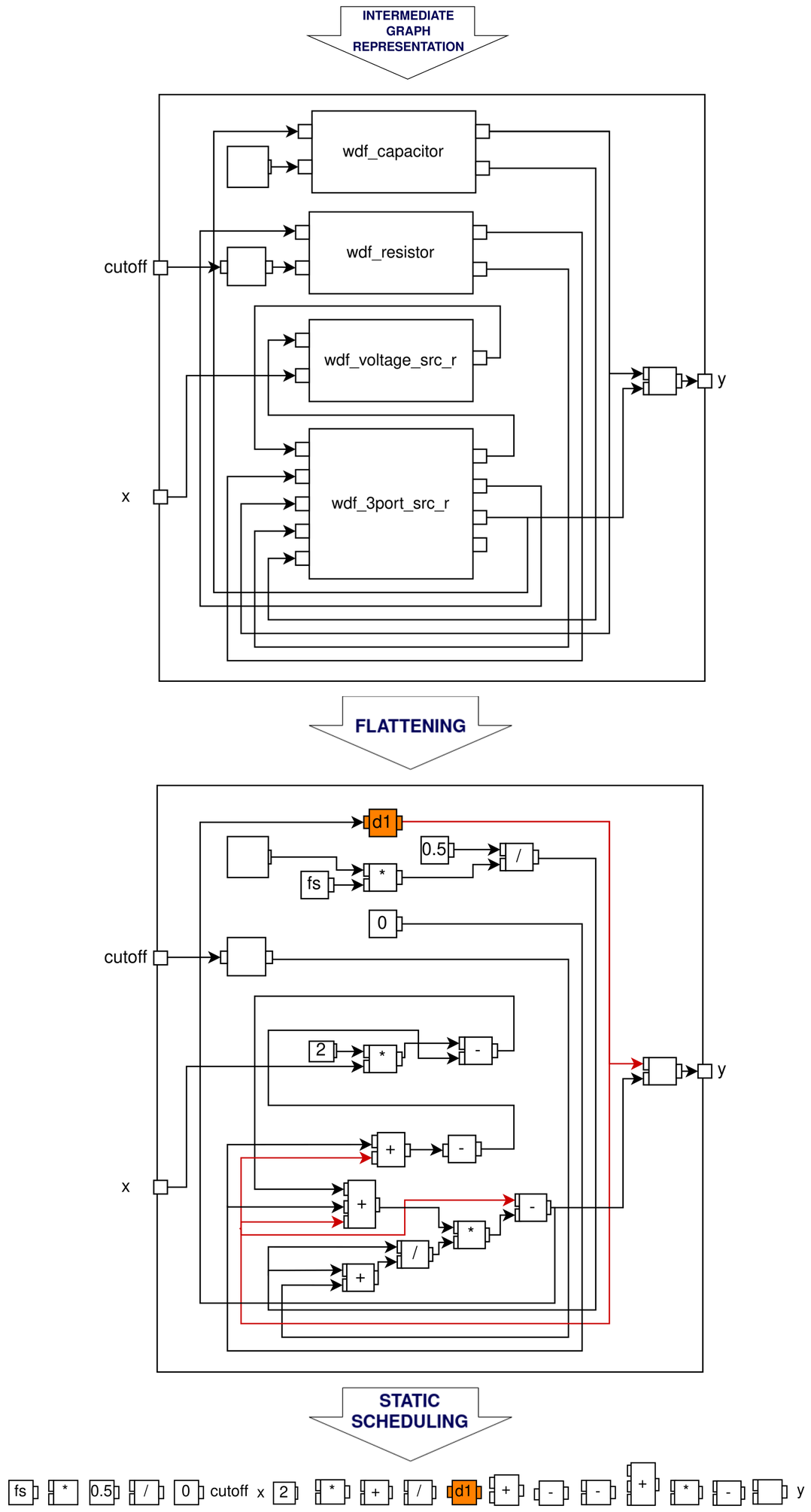
Ciaramella code → AST → **Intermediate Graph Representation** → Flattening → Optimizations → Static scheduling [3] → Output code generation → MATLAB / C++ / JavaScript code.

## CASE STUDY: A Wave Digital Filter [1] Implementation

```
pi = 3.141592653589793          # Constant

b, R0 = wdf_resistor(a, R) {    # Composite block
        b = 0                   # definition
        R0 = R
}
b, R0 = wdf_capacitor(a, C) {
        b = delay1(a)
        R0 = 0.5 / (C * fs)
}
b = wdf_voltage_source_root(a, E) {
        b = 2 * E - a
}
bu, bl, br, R0 = wdf_3port_series(au, al, ar, Rl, Rr) {
        bl = al - Rl / (Rl + Rr) * (al + ar + au)
        br = ar - Rr / (Rl + Rr) * (al + ar + au)
        bu = -(al + ar)
        R0 = Rl + Rr
}
y = lp_filter(x, cutoff) {
        fc = (0.1 + 0.3 * cutoff) * fs
        C = 1e-6
        R = 1 / (2 * pi * fc * C)

        bR, RR = wdf_resistor(aR, R)     # Composite block
        bC, RC = wdf_capacitor(aC, C)    # instantiations
        bV = wdf_voltage_source_root(aV, x)
        aV, aR, aC, Rp = wdf_3port_series(...
        bV, bR, bC, RR, RC)
        @aC = 0                          # Initial value
                                         # for the delay
        y = 0.5 * (aC + bC)
}
```




### Notes

The compiler does not add implicit delays at any point. The flattening operation allows the compiler to discover the "hidden" delays within the instantiated composite blocks and to statically schedule the blocks correctly.

## Forthcoming Research

Conditional and loop blocks; multi-rate support; libraries; $n$-delay operations, arrays and matrices; optimizations; symbolic simplification of expressions; pattern recognition; WebAssembly target; etc.

## References

[1] Alfred Fettweis. Wave digital filters: Theory and practice. *Proceedings of the IEEE*, 74(2):270–327, 1986.

[2] Edward A Lee and David G Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[3] Edward Ashford Lee and David G Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on computers*, 100(1):24–35, 1987.