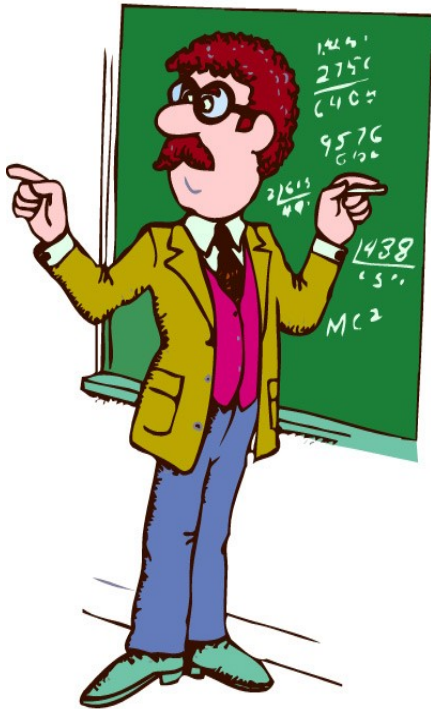




Building Signal Processing Application

Dr. János Selmeczi
HA5FT
ha5ft@freemail.hu

SDRflow What is it all about?



- Short introduction to sdrflow
- Demo application
 - Block diagrams of the application
 - The implementation
- How to get started with sdrflow
- Live demonstration

The framework



- For constructing signal processing apps
- Hierarchical synchronous data flow
- Component based: primitives, composites
- A language for constructing composites
- A compiler for compiling composites
- A runtime for running composites
- Lightweight

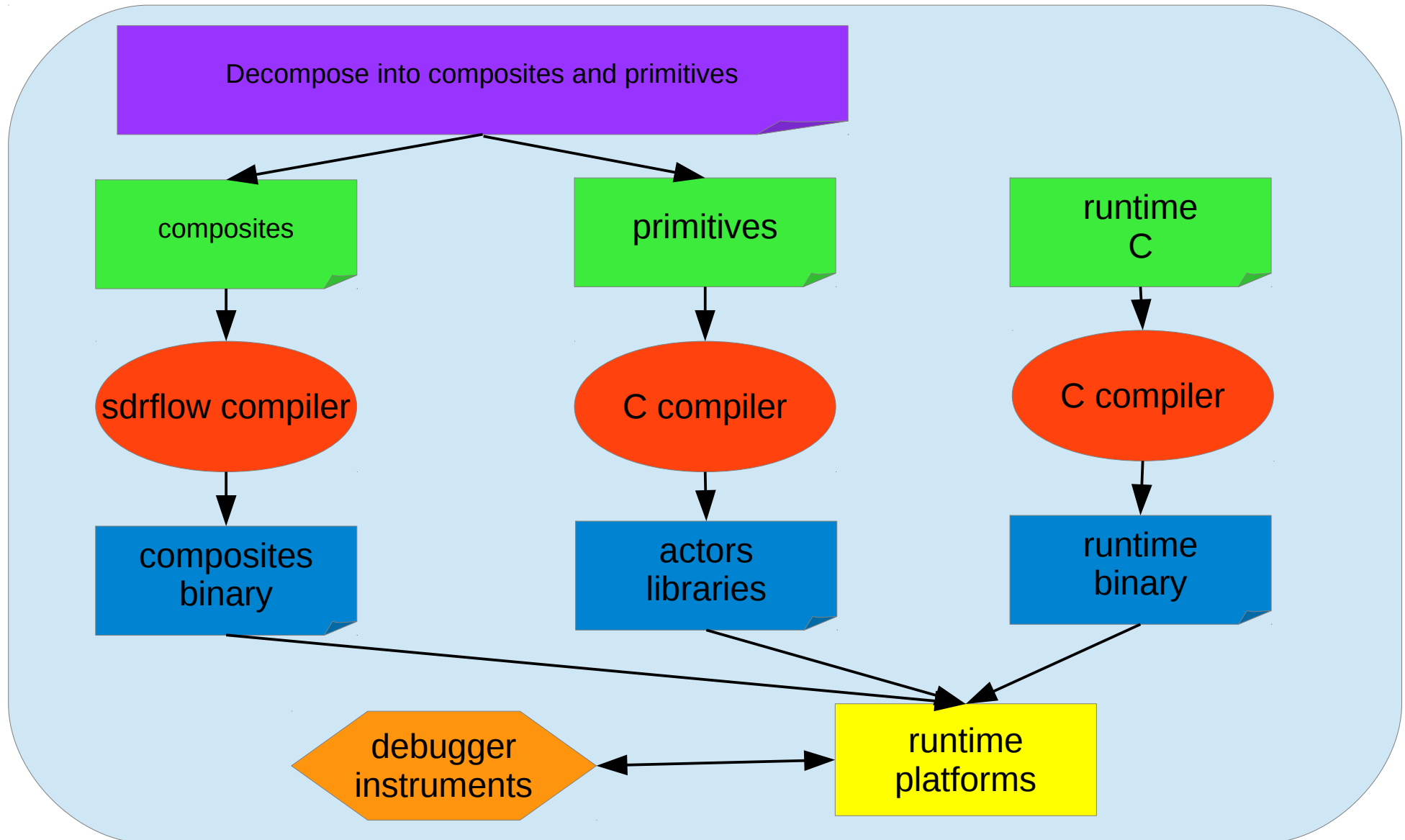


Availability

- Open Source
- Licensed under GNU GPL 3 or any later
- Published on github
 - <https://github.com/ha5ft/sdrflow>
- Current version needs **64 bit Linux**
- Tested on ubuntu 16.04.4 and 18.04



SDRflow Development process



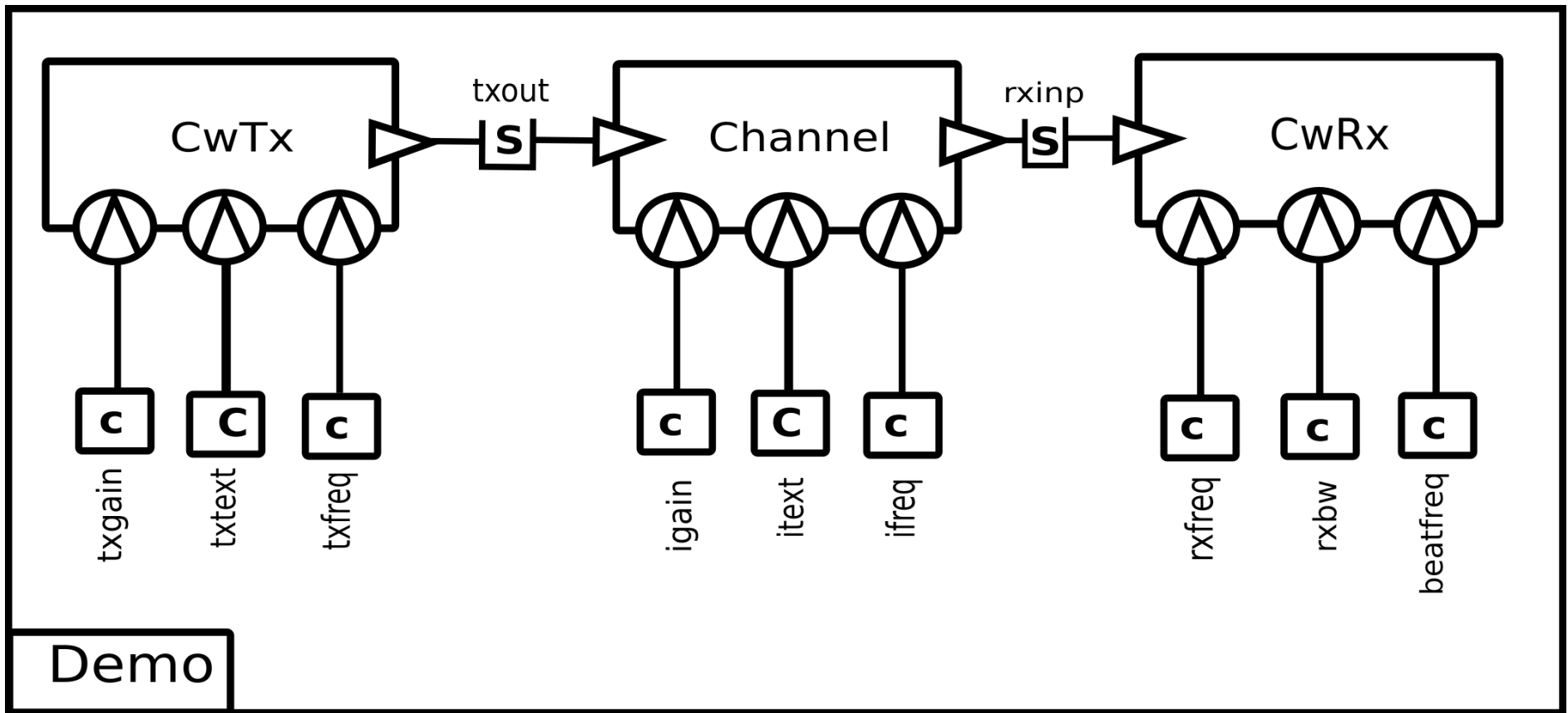
SDRflow The Demo application

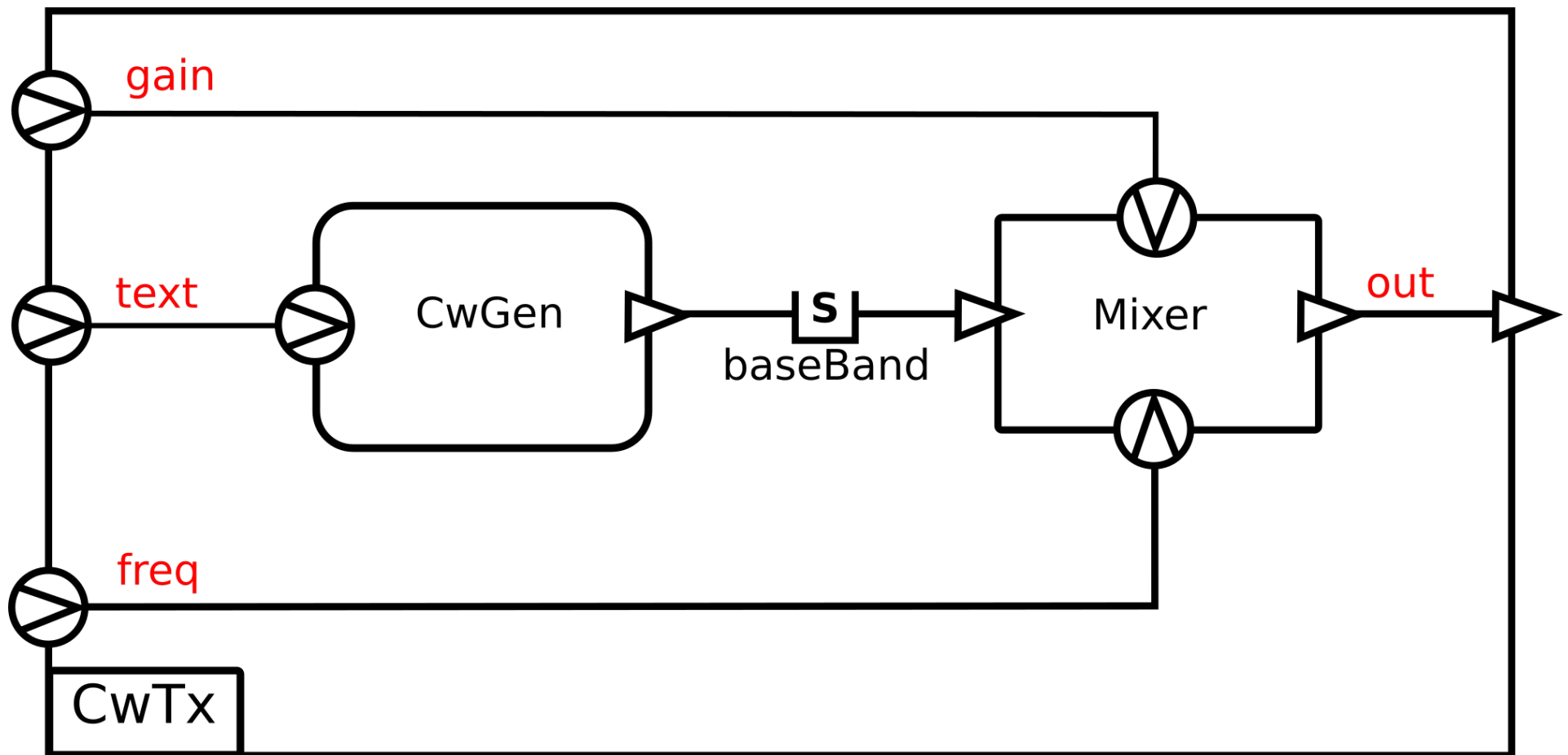
- A *CW TX* connected to a *CW RX* through a *Channel*
- Working in the *(-24kHz,+24kHz)* frequency range
- Sampling rate *48000 sample/sec*
- CW TX
 - Continuously repeat a text on the TX frequency
- Channel
 - Add an interfering CW signal
- RX
 - Filter CW signal on the RX frequency
 - Demodulate the CW signal
 - Send it to the audio card





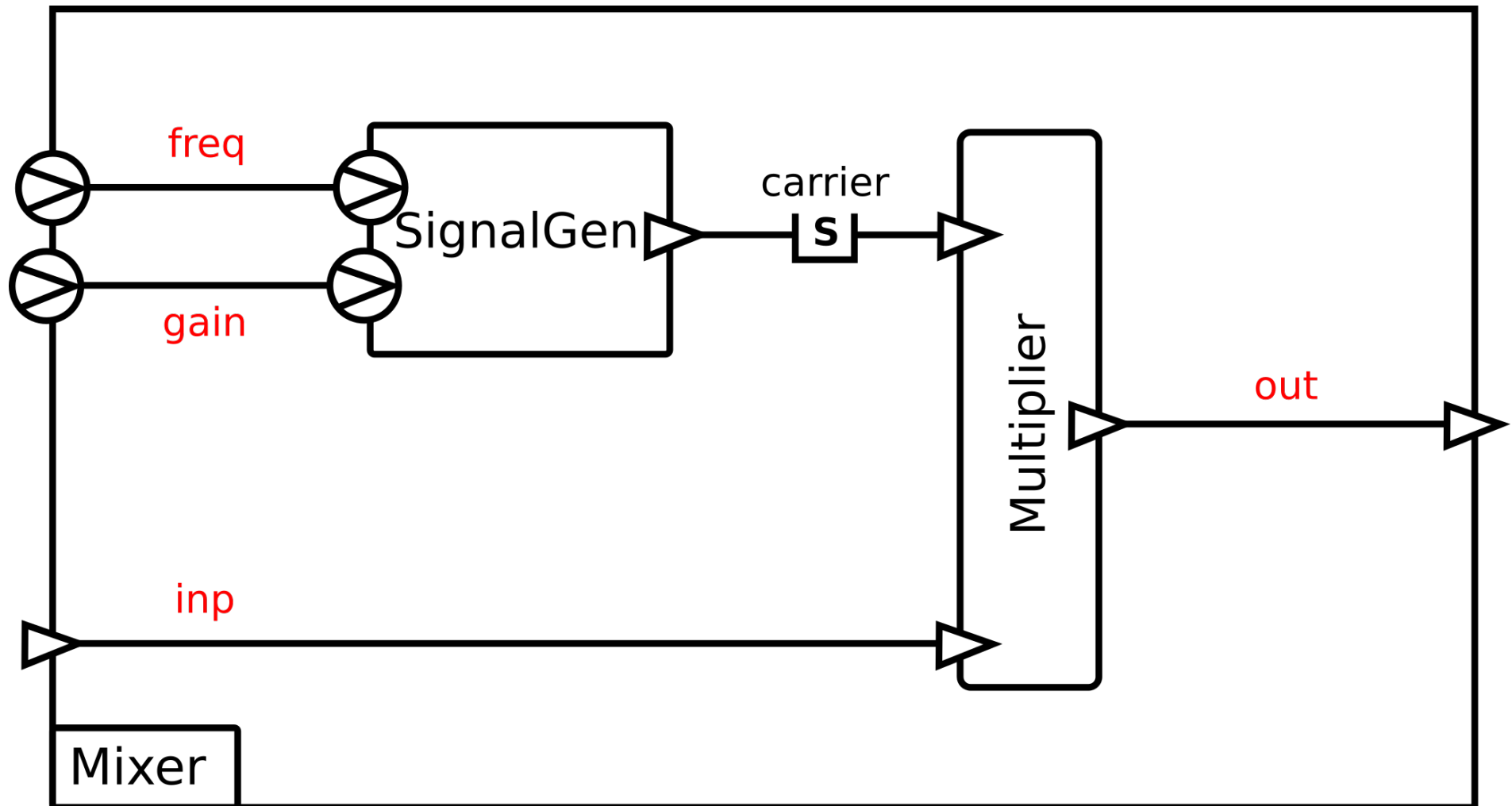
The Demo composite



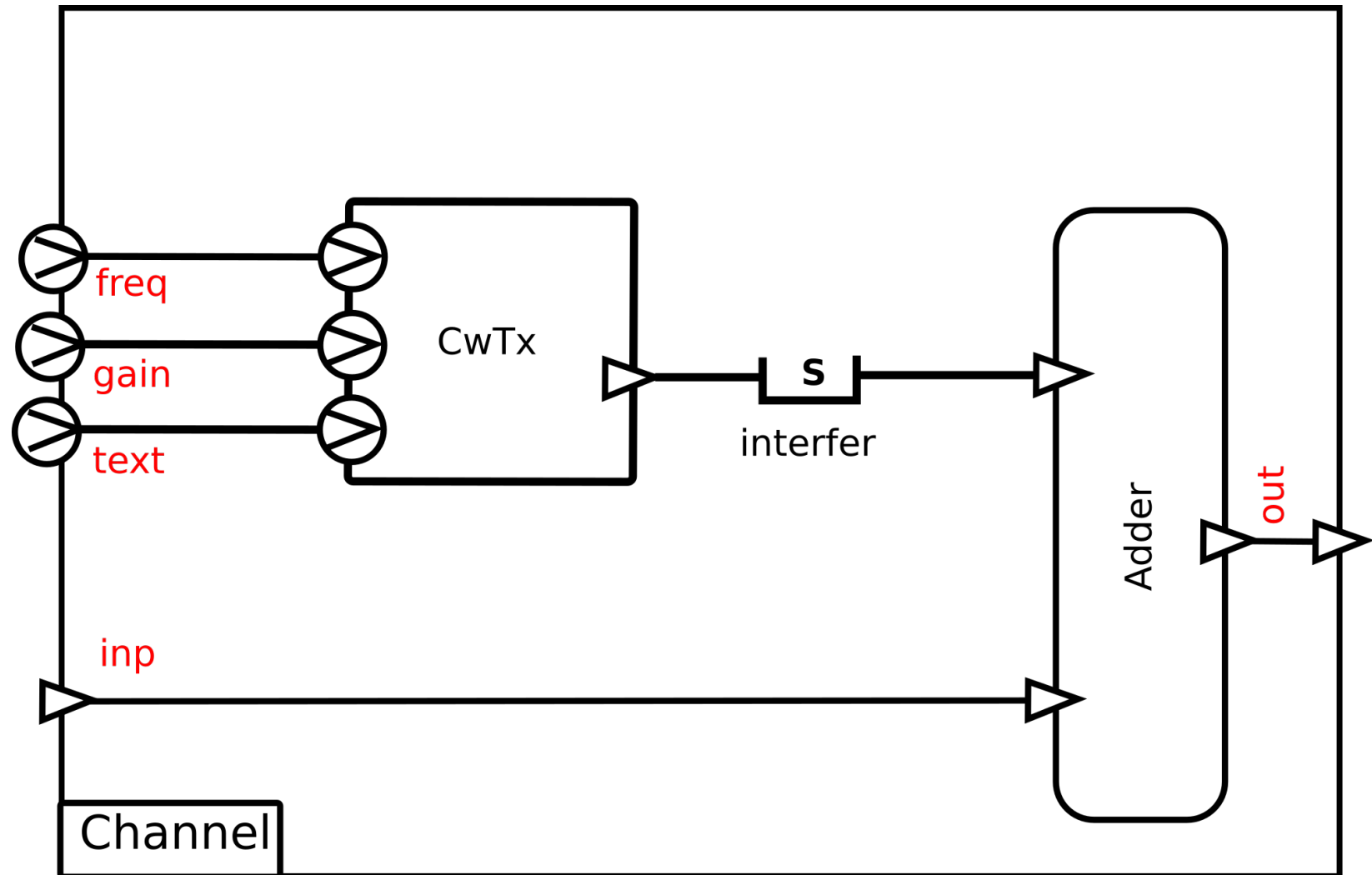


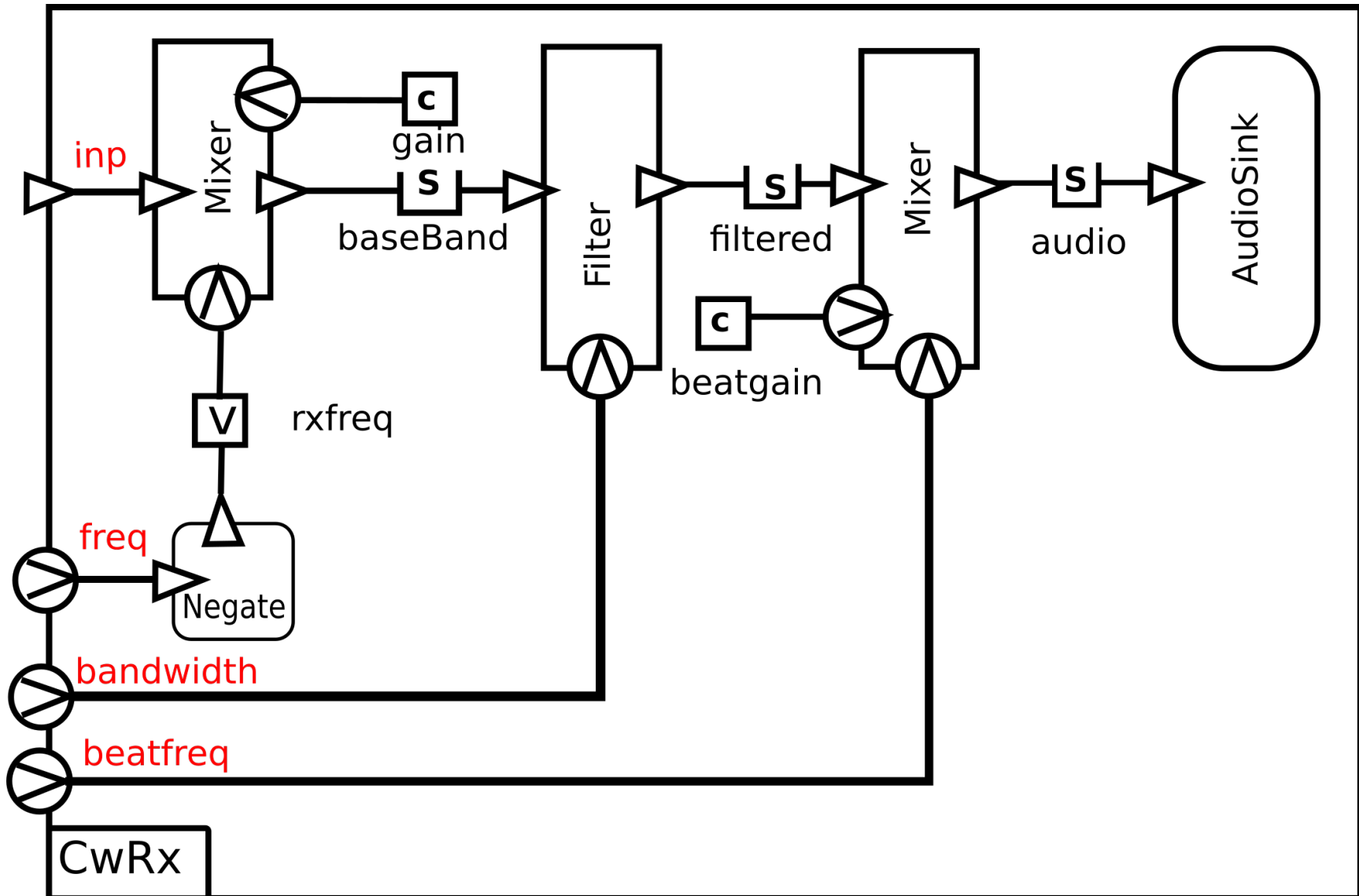


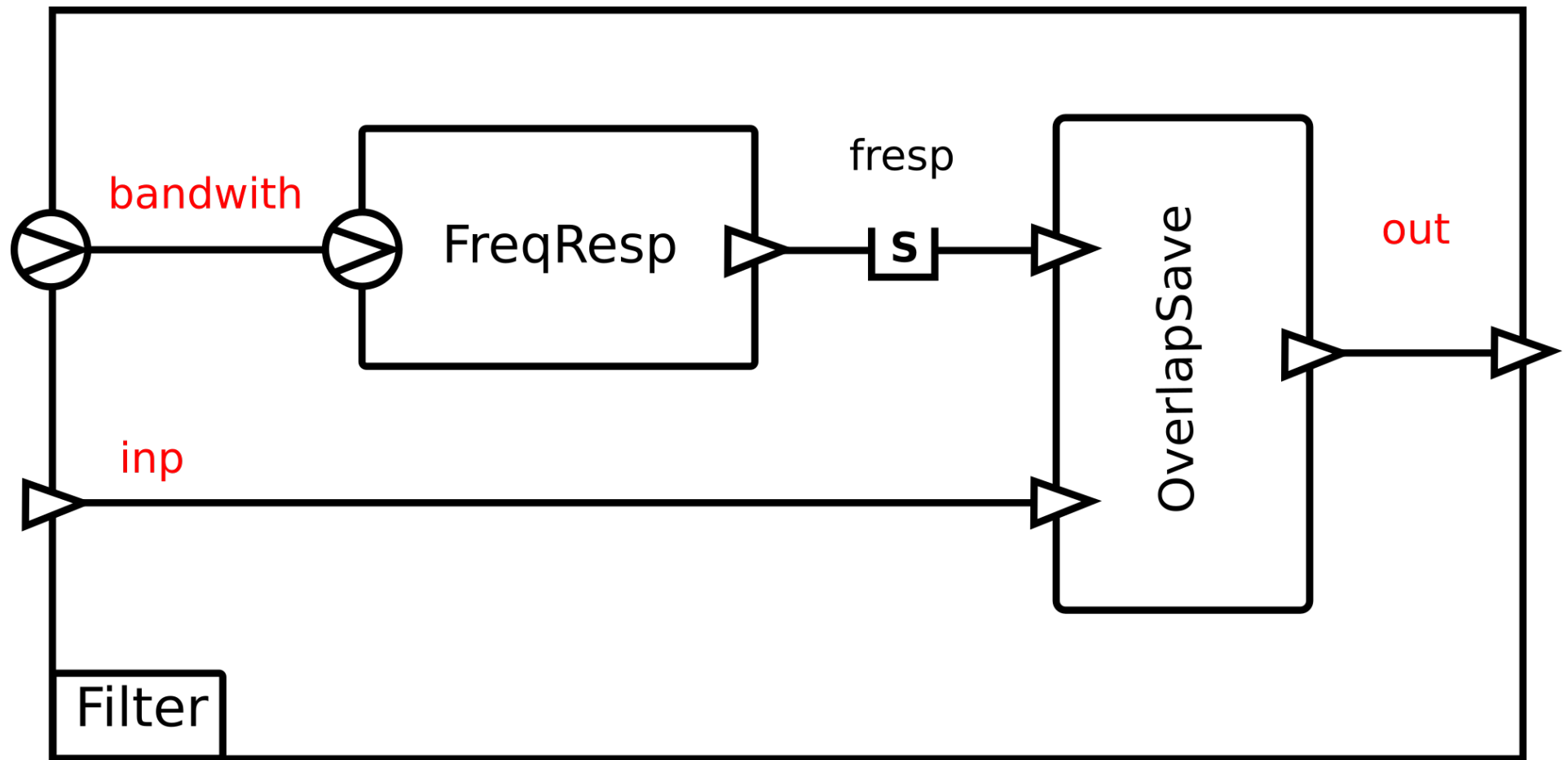
The Mixer composite



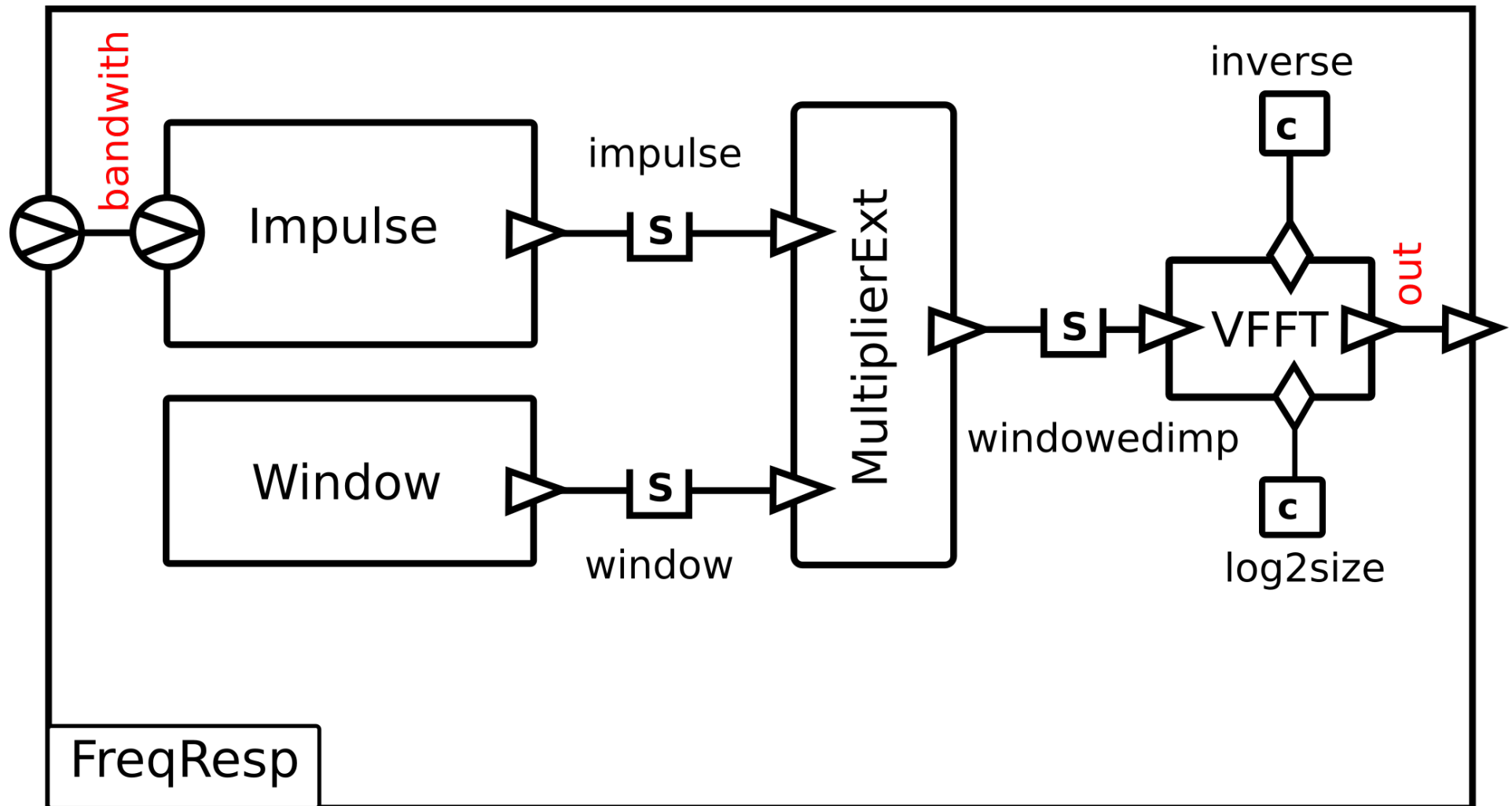
SDRflow The Channel composite



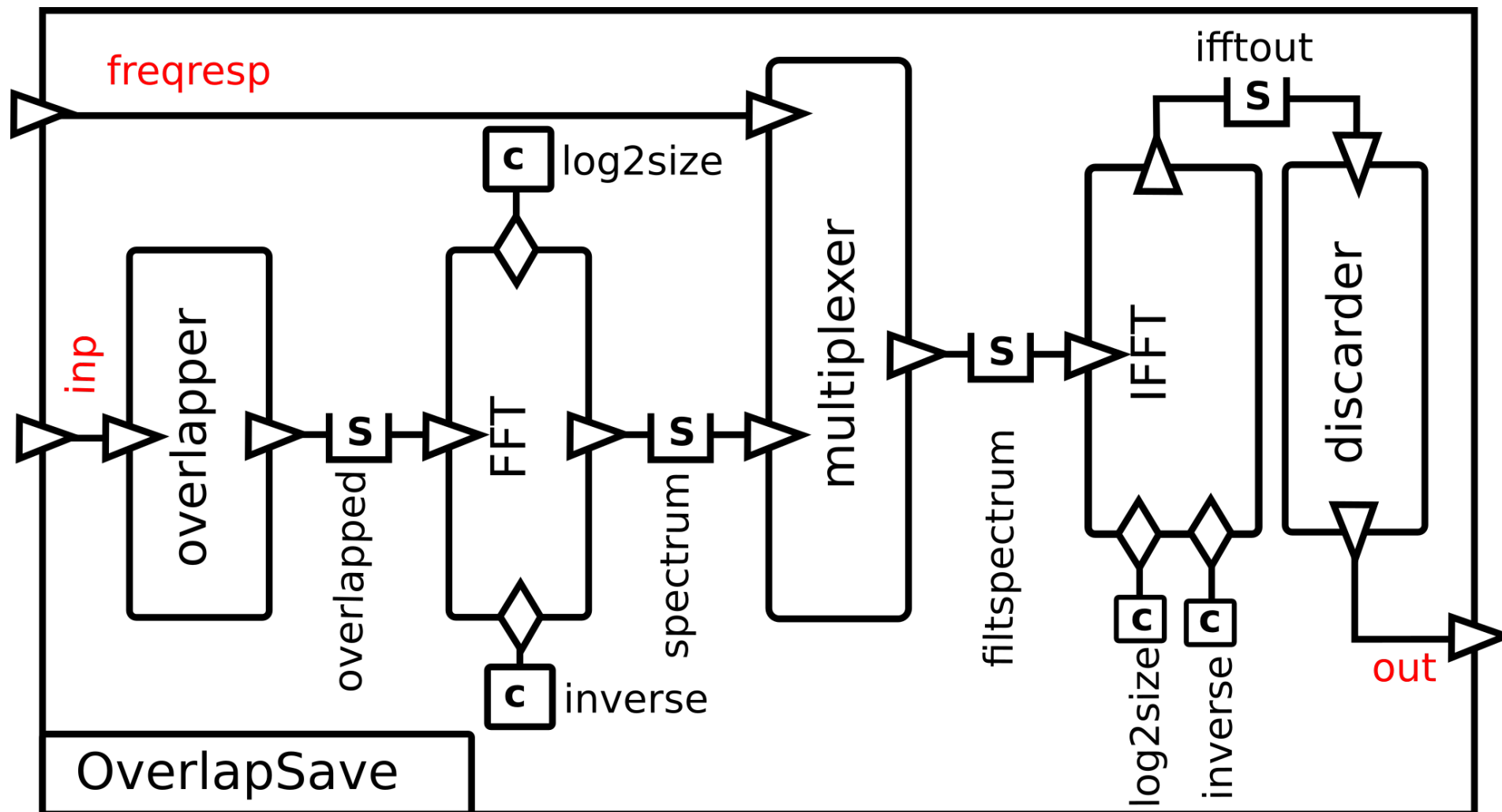




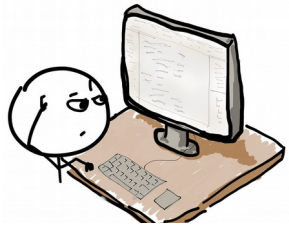
The Frequency Response Generator composite



The Overlap & Save composite



- Create files and directory from template
 - *>make -f primitive.create.mk YourPrimitive*
- Create the interface declaration in YourPrimitive.sdf.ctx
- Fill the context structure in YourPrimitive.h
- Write the 5 function in YourPrimitive.c
- Compile the primitive
 - *>make YourPrimitive*





Negate primitive interface definition

```
primitive Negate
  context
    input    float inp[1]
    output   float out[1]
  end
end
```




Negate primitive context and self structure

```
#include "../include/primitive_interface.h"
#define VECTOR_SIZE 1
struct _negate_self
{
    char *instance_name;
};
typedef struct _negate_self negate_self_t;
struct _negate_context
{
    negate_self_t *self;
    float *inp;
    float *out;
}__attribute__((packed));
typedef struct _negate_context negate_context_t;
```



Negate primitive functions

```
int negate_load(void *context)
{
    return 0;
}
int negate_init(void *context)
{
    return 0;
}
int negate_fire(void *context)
{
    (((negate_context_t *)context)->out) =
        - (((negate_context_t *)context)->inp));
    return 0;
}
int negate_cleanup(void *context)
{
    return 0;
}
int negate_delete(void *context)
{
    return 0;
}
```



Negate primitive

Primitive catalog

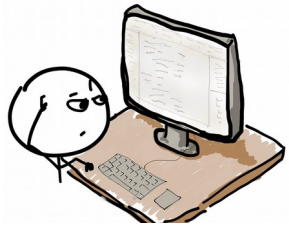
```
#include"../..//include/primitive_interface.h"
#include"Negate.h"

primitive_catalog_t  Negate_catalog =
{
    .name           =  "Negate",
    .self_size      =  sizeof(negate_self_t),
    .init           =  &negate_init,
    .fire           =  &negate_fire,
    .cleanup        =  &negate_cleanup,
    .load           =  &negate_load,
    .delete         =  &negate_delete
};
```



Create composites

- Create file and directory from template
 - *>make -f composite.create.mk YourComposite*
- In YourComposite.sdf.src
 - *Declare the used primitives and composites*
 - *Create the interface declaration*
 - *Declare the signals*
 - *Declare components instances*
 - *Declare the topology*
 - *Declare schedule hints*
- Compile composite
 - *>make YourComposite*



```
use      SignalGen
use      Multiplier

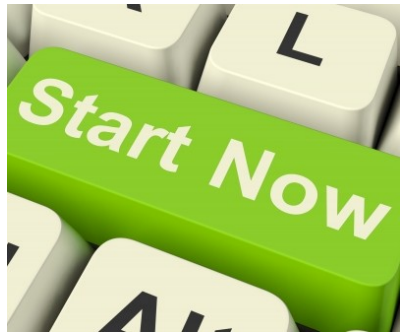
composite Mixer
  context
    input      float[1024] inp_re[]
    input      float[1024] inp_im[]
    output     float[1024] out_re[]
    output     float[1024] out_im[]
    parameter  float      freq
    parameter  float      gain
  end
  signals
    stream     float[1024] carrier_re
    stream     float[1024] carrier_im
  end
  actors
    primitive  SignalGen  localosc
    primitive  Multiplier mult
  end
end
```

```
topology
    localosc.out_re    >>    carrier_re
    localosc.out_im    >>    carrier_im
    localosc.freq      <<    freq
    localosc.gain      <<    gain
    mult.a_re          <<    inp_re
    mult.a_im          <<    inp_im
    mult.b_re          <<    carrier_re
    mult.b_im          <<    carrier_im
    mult.axb_re         >>    out_re
    mult.axb_im         >>    out_im
end
schedule
    auto      localosc
end
end
```



Getting started

- Install ubuntu 16.04.4 **64 bit operating system**
- Install prerequisites
 - `>sudo apt-get install git make gcc libpulse-dev libfftw3-dev`
- Create working directory
 - `>mkdir ~/Sdrwork`
- Clone the repository
 - `>cd ~/Sdrwork`
 - `Git clone https://github.com/ha5ft/sdrflow`
- Build the system
 - `> cd sdrflow`
 - `> make all`





Getting started

- Connect speakers or headphone to the computer
- Start the demo program
 - `>./bin/sdfrun`
 - `sdrrflow runtime version 0.1`
 - `sdrrflow>load /Demo/demo`
 - `OK`
 - `sdrrflow>start /demo`
 - `OK`
 - `sdrrflow># try other commands`
 - `sdrrflow>kill /demo`
 - `OK`
 - `sdrrflow>exit`
 - `>`





- Extending the platform support
 - *Complete the 32 bit Linux support*
 - *ARM Cortex A9 runtime support without OS*
 - *ARM Cortex M4 runtime support*
- Instrumentation for debugging
 - *GUI labor instruments (scope, spectrum analyzer)*
- Data flow extensions
 - *Conditional composite*
 - *C like switch composite*
 - *Iterator composite*
- Built-in external communication