**README.md of the process design kit (PDK) for the H.R.3.3 3D printer**

All of the components for the library are placed in the Components subdirectory which is further broken out into subdirectories into their respective functional directory (valve, mixer, ...). Inside those directories are the respective specific component containing three different files for the software:

* LEF (.lef)
* Verilog-AMS (.va)
* OpenSCAD (.scad)

Directory:

* Building the libary
* Adding components to the library
  + Place and Routing
  + SCAD
    - Installing OpenSCAD libraries on local OpenSCAD
  + Xyce (Spice)

**Building the library**

Building the library the following commands can be run. Each command will make a merge file containing all of the library contents or create a lib binary for simulations.

```

make build\_lef

# h.r.3.3\_merged.lef will be placed in Components directory

make build\_scad

# h.r.3.3\_merged.scad will be placed in scad\_lib directory

make build\_va

# MFXyce.so will be placed in the Components/verilogA\_bulid directory

```

**Place and Route (.lef)**

Running the following command

make build\_lef

Will generate a merged LEF file called h.r.3.3\_merged.lef of the available components, provided they have the correct extension.

**Xyce (.va)**

Running the follow command will compile the Xyce library assuming Xyce is installed. To be able to take advantage of building Xyce libraries. The code does need to be compiled from source with the ADMS instructions provided in <https://github.com/Xyce/Xyce/blob/master/INSTALL.md>.

make build\_va

**3D model generator (.scad)**

Running the following command will build the library into a merge scad file.

make build\_scad

Will generate a merged scad file called h.r.3.3\_merged.scad of the available components, provided they have the correct extension.

**Adding the library to SCAD system library**

To include the scad library components into the main scad library directory it can be install by make. This is not required to use OpenMFDA flow with this library but can be useful in development or for using the components outside of OpenMFDA.

make install\_scad\_libary

or by running the python script:

python3 install\_scad\_libary.py

or with components as there individual files,

python3 install\_scad\_libary.py --unmerged

The scad library can than be included by having

use <h.r.3.3/h.r.3.3\_merged.scad>

// or

use <h.r.3.3/replace\_with\_component\_name.scad>

For using the LEF for SCAD generating script refer to the README.md in the scad\_include directory.

**Adding to the component library**

You can make a new subdirectory for the component that you would like to add to the library. This component will be to be a subdirectory call in the GENERAL\_SRC\_DIR variable in the make file with the appropriate extension (such .va for Xyce, .lef for place and routing, and .scad for 3D rendering ). Otherwise the software will not see it.

Components can be made in new subdirectories (component directories) of the subdirectories of the current Component directory (like within the serpentine directory, or component class directories)

If additional component class directories are made they need to be added to the Makefile variable (within the Component directory) GENERAL\_SRC\_DIR for the software to find them.

Otherwise new component directories can be made and build with the following make commands within the Component directory.

**Individual file instructions for additional components**

Adding new components will require at a minimum creating the appropriate .scad and .lef file. Components without a .va file will not be able to be used with validation tools.

To start a component can be add with an STL file exported from another CAD program, or from generating a .scad script. Geometry can be checked with OpenSCAD to validate the geometry. This library also contains tools for generating basic .lef file from OpenSCAD.

**LEF**

This file format is typically used to decribe an abstract representation of a library component.

LEF files can be created manually through the LEF definition, which we describe here.

We provide a serpentine\_100px device as an example of the necessary statements of a component. This guide will give a brief overview of creating a component in the library. For those wanting to use the more advanced features of LEF files, a detailed overview of the additional LEF/DEF rules can be found through various language references.

MACRO serpentine\_100px\_0

CLASS CORE ;

ORIGIN 0 0 ;

SIZE 180 BY 180 ;

SYMMETRY X Y ;

SITE CoreSite ;

PIN in\_fluid

DIRECTION INPUT ;

USE SIGNAL ;

PORT

LAYER met1 ;

RECT 23 23 37 37 ;

END

END in\_fluid

PIN out\_fluid

DIRECTION OUTPUT ;

USE SIGNAL ;

PORT

LAYER met1 ;

RECT 143 23 157 37 ;

END

END out\_fluid

OBS

LAYER met1 ;

RECT 30 30 150 150 ;

END

END serpentine\_100px\_0

This device is defined as a MACRO with the name of the component following.

The CLASS statement is used to define which placement area with cell is used for.

The ORIGIN statement is used to define the location origin of the component when aligning it to the component grid with the two numbers following denoting the X and Y position. When omitted the default value is 0 0.

The SIZE statement is used to describes the outer most bounding box of the component. The rectangle always extends out from (0,0). Placers will use this to avoid overlap of other placed components.

The SYMMETRY statement specifies which orientations can be attempted during placement. These orientations will be flip/mirror transformations along the axis specified.

The SITE statement specifies the placement area associated with the macro. We only have one site in the flow which CoreSite for placement.

The PIN statement specifies the named pin of the component and contains statements of the named pin that is enclosed by `END pin\_name` that specifies the details of that pin.

The pin DIRECTION statement declares the direction of the named pin, INPUT, OUTPUT, and INOUT are typical values. This can be useful for design rule checks (such as checking whethter two inputs are connected together).

The pin USE statement describes the purpose of the pin, most microfluidic pins will be SIGNAL.

The pin PORT statement Indicates that the following LAYER, POLYGON, and VIA statements are all part of one PORT connection, until another PORT statement occurs. If this statement is missing, all of the LAYER, POLYGON, and VIA statements are part of a single implicit PORT for the PIN.

The OBS block is a group of statements enclosed by OBS at the beginning and terminated by the END statement. Inside are the geometric

LEF shapes will generally be a grouping of two statements a LAYER definition and shape definition. In our example we use the RECT statement for our shapes which is defined as RECT ptx1 pty1 ptx2 pty2, where the two pts defined are opposing corners of the rectangle. The LAYER statement is used to define which layer the shape belongs to. The LAYER definition is first when defining other shapes and remains in effect until the next LAYER statement.

**OpenSCAD**

OpenSCAD is an opensource textual CAD modeling software. It is primarily used to generate the 3D rendering of the device and components to use in slicing.

Components developed for the library are to be developed as extruded features of the printed geometry (the channels and internal features are developed as the solid bodies). In the OpenMFDA process the interal geometry is build as a single solid body that is later subtracted out of the "bulk" of the device.

Recommended approaches:

1. Building with OpenSCAD
2. Importing STL files

**Buliding the geometry in OpenSCAD**

On can learn to use OpenSCAD to develop thier microfluidic components. OpenSCAD provides some tutorials on how to use the software. We also provide useful modules for microfluidic development.

Each component will need to be developed as a stand alone module with the same name as the file. Each component at a mininum will require the arguments xpos, ypos, zpos, orientation to be the first four arguments. All additional arguments will need to be optional (has a preset value). Most components in the library will be static non-parameteric geometries. An example of a template of a component will look like:

```

module component\_name(xpos, ypos, zpos, orientation) {

// component geometry

}

```

Alternatively, we also provide a set of components that can be used to develop a standard geometry from simpling calling the module and changing it's parameters. An example of a scad portion of a component is provided below:

```

use <../../../scad\_include/scad\_objects/p\_valve.scad>

module serpentine\_100px\_0(xpos, ypos, zpos, orientation) {

p\_serpentine(xpos, ypos, zpos, orientation, L1=100, L2=30, turns=3) ;

}

```

The idea is the standard component is wrapped around the parameteric component with the parameters defined for the specific implementation.

**Importing an SLT**

OpenSCAD allows for the import of other 3D geometries file formats which allows users to build there geometry in a different software such as SolidWorks, AutoCAD, Inventor, Creo, Fusion360. This feature is currently being worked on and this section will be updated when ready.

**VERILOG-AMS**

The OpenMFDA flow uses Xyce as the simulation tool to verify the flow, pressure and chemical concentration. In the library components are developed with both fluid ports and chemical concentration ports as separate existing ports. One can develop using the provided `fluidDynamics` discipline in this tool with the access variables P and Qfl, for pressure and flow rate respectively. The components are to be developed as 1-dimension representations of the component they are trying to capature. We adopt a similar paradigm as

Kang, Seok-Won, and Debjyoti Banerjee. "Modeling and simulation of capillary microfluidic networks based on electrical analogies." (2011): 054502.

These files are not required to run the layout and rendering modules of the flow, but will be useful for validating the function of the device.

A quick reference of Verilog-AMS is avalible at <https://verilogams.com/quickref/basics.html#variables>

It is suggested that all behavioral definitions for each verilog-ams file follow the guidelines outlined in:   
[C. C. McAndrew et al., "Best Practices for Compact Modeling in Verilog-A," in IEEE Journal of the Electron Devices Society, vol. 3, no. 5, pp. 383-396, Sept. 2015, doi: 10.1109/JEDS.2015.2455342.](https://ieeexplore.ieee.org/document/7154394)