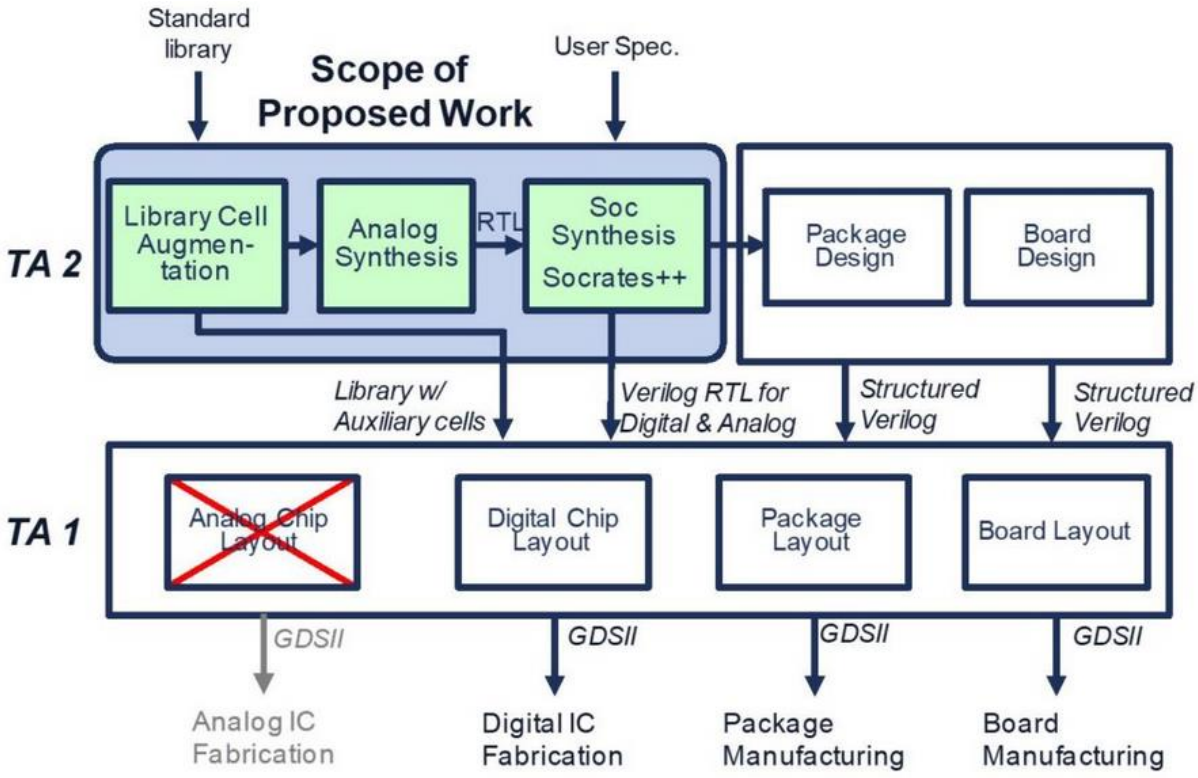


FASoC/IDEA FLOW TUTORIAL (GF12LP)

An In-Depth Guide to Your First FASoC Design

Overview

This tutorial serves as a guide to a new user of the IDEA/FASoC program and will guide you through all of the steps required to get your first design tapeout ready. There is also useful information suited for people outside of IDEA/FASoC such as general information about working in fin-FET technologies like GF12lp and working in GitHub.

Michels, Noah
nmichels@umich.edu

Last Updated 10/26/21

Table of Contents

Getting Started 2
Overall Steps to Complete 3
Useful Locations 4
Working in GF12 and Fin-FET Technologies 4
Extra GF12 Layout info: 7
Analog Design in GF12 8
AUX Cell File Generation 10
.sp: 10
.cdl: 12
abstract & .lef: 13
.gds: 19
.lib: 19
FASoC Block Generator 20
./ 20
./blocks/ 20
./src/ 20
./scripts/dc/ 20
./scripts/innovus/ 21
Finishing Up 21
Post-PEX Simulations 22
PEX Extraction of Design 22
Spice Testbench Generation 24
Simulating Design 25
Pre-PEX Simulations 26
Top-Level for Tapeouts 26
./ 26
./fasoc_soc/soc_top/ 26
./src/ & ./scripts/dc/ 27
./scripts/innovus/ 27
Verifying Design 27
Working with GitHub 28
Merging Changes 28
Pushing Final Changes 28
Final Tapeout Upload 29
Appendix 30

Getting Started

Note: *Some file locations may not be general enough to be directly applicable to your design, so be sure to keep your working directory in mind. Also know that some file locations or file structures may have changed since this document was first made. Feel free to reach out to one of the FASoC members for help on first setup or any questions you may have.*

The first step toward making your design, is to setup your working directory for the current technology you are working in, which for the purpose of this tutorial will be GF12lp. It is probably best to just copy the directory of another FASoC student. For now, lets use my directory as an example. There are probably many files you don't need from this directory, so if you want to copy only the necessary files that is fine too. Also before doing any of this be sure you have gotten access to the FASoC directories and the PDK files.

```
$cp -r /n/Marquette/v/nmichels/GF12 ~/GF12
```

Great! The GF12 directory will be where you create your initial schematics and do your first tests of your design. It will also be the directory where you create you work to generate your aux cells.

Next let's go ahead and setup your FASoC working directory in your home directory. This directory is going to be a copy of the current GitHub master branch and will be where you will do the traditional FASoC work such as making your automated design (this will be covered in more detail later, so don't worry too much about this for now). Also before doing this step be sure you have been given access to the FASoC GitHub.

```
$cd
$mkdir fasoc
$cd fasoc
$git clone --recursive git@github.com:idea-fasoc/fasoc.git
$cd fasoc/private
$git checkout master
$git submodule update --recursive --init
```

Now we will go ahead and setup your block generator within the fasoc directory. You won't be using this much until later, but let's go ahead and get it out of the way.

```
$cd ~/fasoc/private/generators
$mkdir ignore_[BLOCK_NAME]
```

There is a lot that goes into one of the generator directories, so for now let's just copy one of the other blocks' subdirectories.

```
$cd ~/fasoc/private/generators
$cp -r pll-gen/gf12lp/flow_dco/* ./ignore_[BLOCK_NAME]
```

We will worry about actually changing all of the files in your new generator directory later, but for now we have a good jumping off point.

Last thing is you will want to get someone's ".tcshrc" file for loading all of the necessary modules when working in the fasoc directories. For now, we will just use mine again. Can change file name if you don't want to overwrite your current ".tcshrc".

```
$cp /n/Marquette/v/nmichels/.tcshrc ~/.tcshrc
```

Nice! Now that we are setup, we will go over the overall steps we need to complete to get your design ready.

Overall Steps to Complete

Note: *Everything through step 2.b will be done in GF12 directory and the rest will take place in fasoc generator directory. Can focus on just getting on design to work for now, and worry more about auto generation later. Also note that there won't be a section on creating/testing design schematic in Virtuoso, as it is assumed that you already know how to do this (still feel free to reach out with questions if having trouble).*

1. First step is to make design in GF12lp
    a. Just worry about schematic initially, then focus on the layout of aux cells
        i. Goal is to use standard cells, so don't use rf_fets for basic components
        ii. If standard cell doesn't exist for part, make custom std cell (AUX cell) that meets normal std cell sizing (equal height, and int. multiple of width)
    b. After schematic made, run sims and optimize
    c. Once optimized, test layout to get idea of how things will be placed
    d. GF12lp has many more rules (finfet), so if problem takes >1hour, ask for help

2. Now break overall design into smaller AUX cells (hopefully considered this while making original)
    a. Aux cells should meet standard cell sizing requirements (other than caps/inductors/resistors)
    b. Generate files needed for AUX cells (see AUX Cell File Generation Section)
    c. Make Verilog files defining IO for each of these aux cells
    d. Make Verilog top level which defines connections between blocks

3. Next Synthesize design
    a. Synthesis scripts located in ./scripts/dc/
    b. make synth
        i. check results/dc/design_name.mapped.v
        ii. Note that blank connectivity means block is connecting to power/gnd. Power connections are defined in Innovus

4. Next APR design
    a. APR scripts located in /scripts/innovus/
    b. Also have to specify placements for blocks in custom_place.tcl
    c. There are series of "stages" for APR. "stage" runs all previous "stages" if not run yet.
    d. make "stage" for stage = init, place, cts, postcts_hold, route, postroute, signoff
        i. Also have make debug_"stage"

5. LVS & DRC
    a. Focus on LVS first, as this is quicker and DRC will likely have issues
    b. make lvs; make debug_lvs
    c. make drc; make debug_drc

6. Sims
    a. Use Finesim/HSPICE to perform sims

i. Need custom python script to generate PEX results and post-PEX sims

7. **TOP LEVEL & PADS….**
   a. This will be covered later, for now just focus on getting through block generation

Useful Locations

Recommend alias to quickly access some of these files, especially design manual. If you copy my .bashrc/.tcshrc, then you will have some already.

Tool Documentation: Most files found in /usr/caen/ for tool information (Ex: Innovus, finesim, hspice)

Technology Documentation: Most files found in /afs/eecs.umich.edu.edu/kits/ for pdk information. Documents below are for GF12lp, but can give idea of where to look if using a different technology.

- **Design Manual:** /afs/eecs.umich.edu/kits/GF/12LP/tapo_V1.0_4.1/source/12LP_Rev1.0_4.0.pdf
- **STD Cells Rules/Info:** /afs/eecs.umich.edu/kits/ARM/GF12LP/arm/gf/12lp/platform_userguide/r0p0/doc/sc_12lp_doc_userguide.pdf
- **STD Cell Files:** /afs/eecs.umich.edu/kits/ARM/GF12LP/arm/gf/12lp/sc10p5mcpp84_base_rvt_c14/r0p0/
- **PAD Info:** /afs/eecs.umich.edu/kits/ARM/GF12LP/arm/gf/12lp/io_gppr_t18_mv10_mv18_fs18_rvt_dr/r1p0/

Working in GF12 and Fin-FET Technologies

Note: *This section will cover just some general tips and tricks for working in GF12 and other Fin-FET technologies. If you are already familiar with this type of work, you can skip this section and focus on the stuff relating more to FASoC.*

Before working in your GF12 directory, be sure to source the .bashrc file. Some differences between two .bashrc files.

```
$source .bashrc_pre_gf12_bkp
```

or

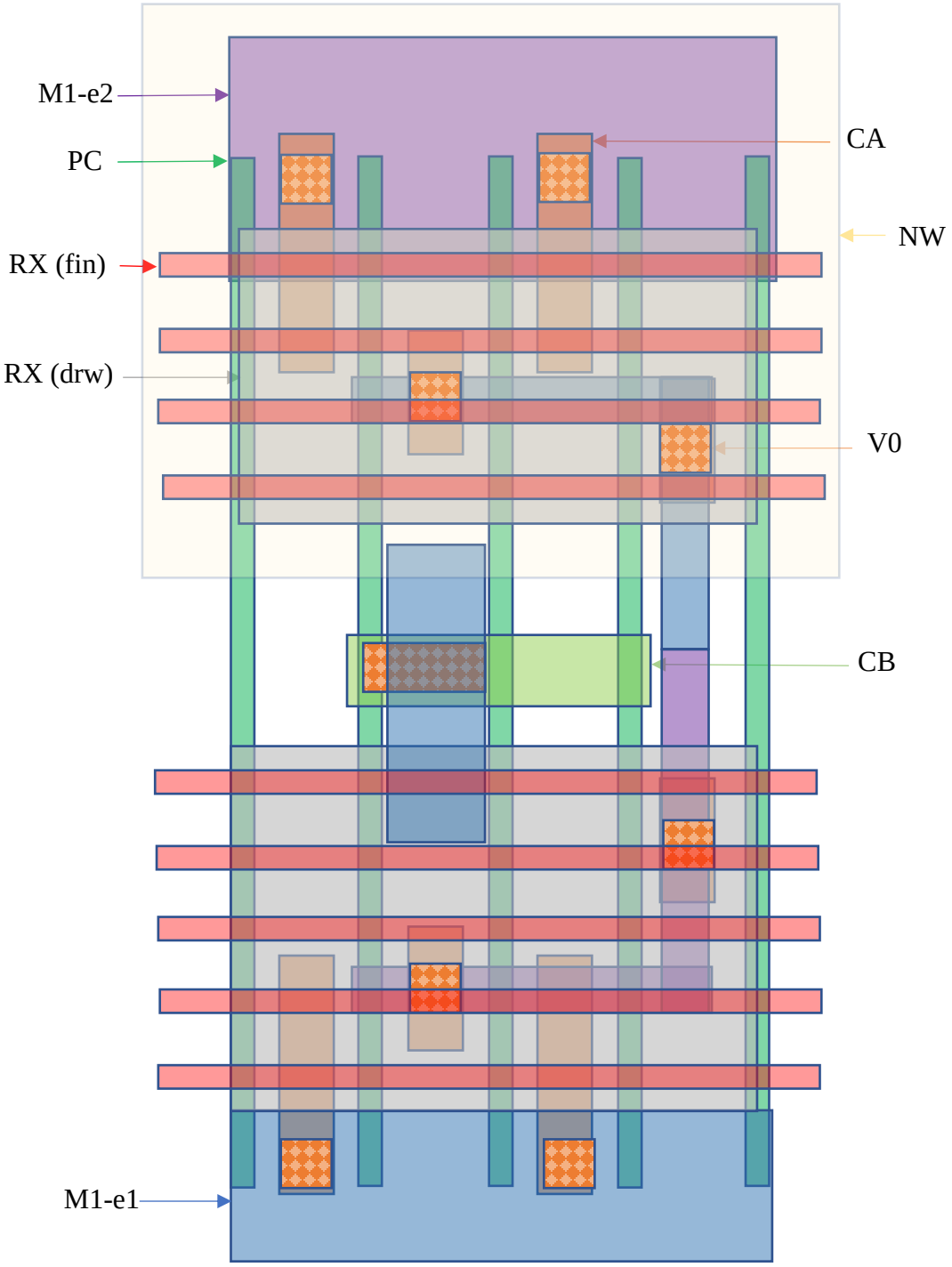
```
$source .bashrc_gf12
```

First thing we will cover is just some general layout explanations for this PDK. All of the images in this section have been generated outside of Virtuoso in an attempt to not break any NDA rules, but feel free to follow along in virtuoso to see exactly what I am talking about. Also, a quick note in case you haven't taken 427/627. In your classes you have likely been using IBM 130 and therefore are used to a lot of freedom in the layout process. In smaller technologies such as GF12, things are laid out in a grid like manner. This means things like PC are always going to go in one direction and will occur at standard intervals, so don't expect to be able to do any creative routing with this layer.

Okay, so before going into any more detail I think it is best to actually see an example. For this example, we are going to look at an inverter standard cell in the sc10p5mcpp84_12lp_base_rvt_c14 std cell library. This is the library that we were using at the time of writing this document. More information can be found in the Arm user guides which are listed in the section discussing useful locations.

The layout has a lot going on, so let's start on the next page so we can clearly see everything. I highly recommend having the design manual and the inverter standard cell open while reading through the next section so you can get a better understanding.

Consider an inverter:



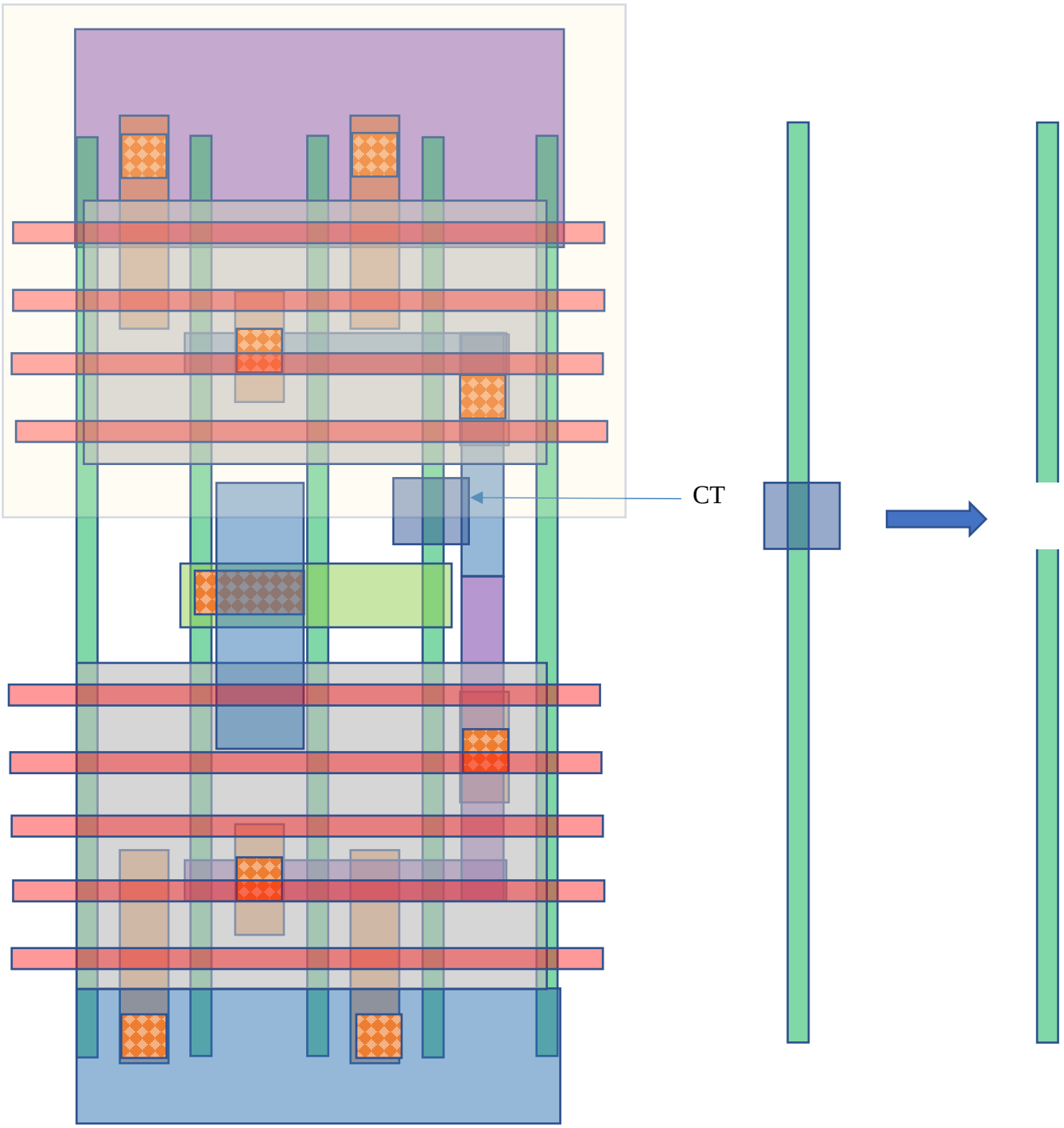
M1-e1 and M1-e2 occupy same metal layer but are fabricated during different lithography steps (This double metal layer is the case for only M1-M3). NW and RX(drw) operate pretty much same as most other PDKs. CA is used to make contact to the RX(fins) whereas CB is used to make contact with the PC. To connect to upper metal layers, the CA/CB have to be followed by a V0 (a zeroth via). Note that CA doesn't have to directly contact all RX(fins) thanks to the TT layer (not shown). The TT layer occupies the same area as the RX(drw) layer and connects the CA to the RX. There is also another layer known as the TB layer (not shown) which

marks the area where TT should not be placed. The TB layer overlaps the PC and prevents a short between the source and drain due to the TT layer.

The layer naming for the rest of the metal layers and vias is some variation of the following (changes depending on the exact metal stack that is being used):
CA/CB-*V0*-**M1**-*V1*-**M2**-*V2*-**M3**-*J3*-**C4**-*A4*-**C5**-*A5*-**C6**-A6-**C7**-*CK*-**K1**-*U1*-**K2**-*KG*-**G1**-*T1*-**G2**-*W*-**LB**

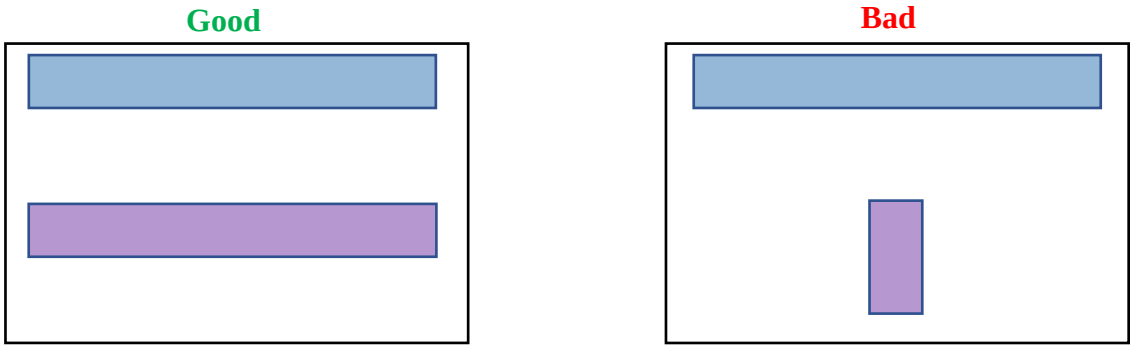
More information and a useful diagram of the metal layering can be found in the design manual.

What if you need to control the gate of a PMOS without controlling the gate of a complementary NMOS? This is done by using the CT layer, which effectively cuts the PC which it is over. Thus, you should still run the PC and TB layer from the top to bottom of the std cell, but just place a CT layer between the two RX layers to prevent the gate controlling both of them (note that TB layer is not shown in this fig).



Extra GF12 Layout info:

- GF12 standard cells may not have schematic view initially, and there are issues with importing the .cdl files, so ask for someone who already has the files.
- GF12 has many layer "purposes" (ex: drawing, e1, e2, label). In FinFET technologies, some metal layers require two masks (each mask corresponding to a separate lithography-etching process, meaning that M1-e1 is the first fabricated & then M2-e2 is the second fabricated, although they are in the same layer). In 12LP, we need e1, and e2 masks for the M1/M2/M3 layers. There are specific design rules for e1-e1, e2-e2, and e1-e2. Best to use e1/e2 layers at start rather than drawing layer for M1/M2/M3.
    - Can ignore many of the other purposes for a given layer, though some are still used (ex: apmom for defining capacitor areas for mom caps)
- If you make custom aux cells, they should try to follow standard cell format
    - Cell height should be 0.672 for 10p5 std cells
    - Cell width should be 0.186+n*0.084 for some int n
- STD cell naming convention:
    - sc9 vs sc10p5: different top power rail sizing and metal layer purpose
    - INVP vs INV: "cell"P includes parasitic FETS
    - INV_X0P6"F/N/R": Cells sized for either falling, normal, or rising delays.
- If your FASoC design is having trouble routing later, can make standard cell wider by using FLTGATE layer. Just add extra PC column and cover with FLTGATE so it is not considered as a FET during LVS
- When trying to route M1-e1 and M1-e2 in close proximity, try to do so by having them run parallel to each other. The DRC rule for spacing is more forgiving in this scenario. Example below:



- There will probably be many DRC errors when checking your AUX cells. My suggestion is to open up a standard cell and run DRC on it in parallel. The standard cell will also have many DRC errors, but these all go away once more standard cells are placed adjacent to each other and end cells are placed at ends (happens normally during IDEA/FASoC flow). As long as your AUX cell only has errors that the other STD cell has, you should be okay.
    - Note that this is not the case for passive cells such as capacitors which should be DRC clean
    - Also note that more general analog design requires more manual changes to ensure DRC will be clean at the end.
    - Aman is working on tutorial for more general analog design in GF12, so consider reaching out to him if this would be of assistance.

The next section will cover a few tips on analog design in GF12 and can be skipped if this is not relevant to your work. Once you have got your schematic finished and AUX cells laid out, you are ready to generate the files for your AUX cells that will be needed to get through the FASoC flow. This will be covered in the "AUX Cell File Generation" section which comes after the "Analog Design in GF12" section.

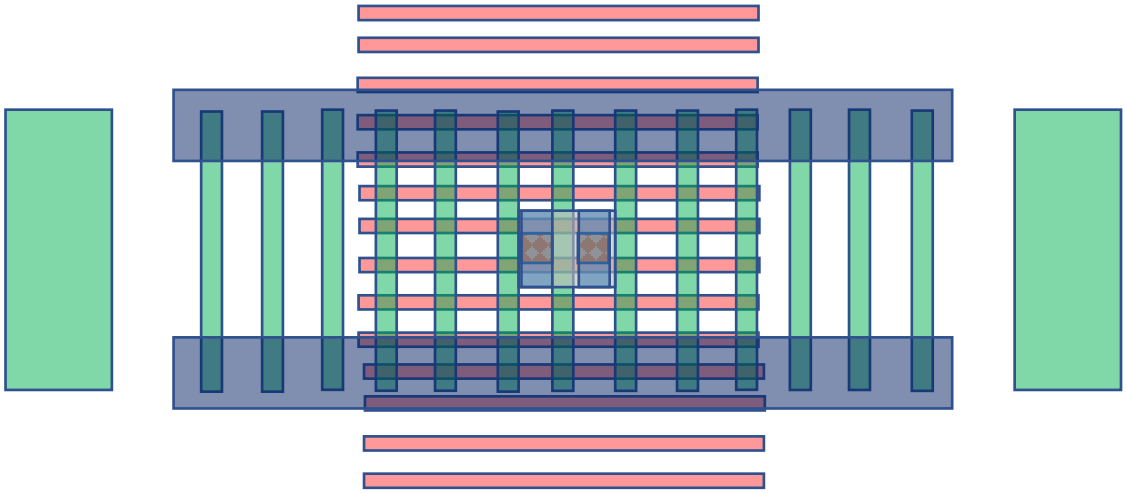
Analog Design in GF12

Note: *This section will cover just some general tips and tricks for doing analog layout in GF12 and other Fin-FET technologies. If you are mostly concerned with STD cell layout for the FASoC flow, this section can be skipped.*

While we've already gone over some general advice on layout in GF12 and Fin-FET, we have previously mostly concerned ourselves with layout following the STD cell formatting. Most of what we discussed previously will still hold true here, but there are a couple of things that should be noted when working with analog designs.

The first step will of course be to just make your design in a schematic cell view. Feel free to use whatever gate length and fin number that you need for the best performance for your device, as we are no longer limited by STD cell sizing requirements. Once you have your design working as you like, can work on generating the layout.

The first step for making your layout will just be to instantiate the pcell layouts of the transistors you used in your schematic. I feel that now is a good point to say that I highly recommend working in layout XL if you aren't already (can easily copy pcells from schematic into layout and lots of other useful features). The pcells of the transistors are going to look pretty weird, but this is to be expected. I've drawn a sort of arbitrary pcell layout below for a random transistor (some layers not shown).



One of the first things to note is that if you ran DRC at this point, it would not be clean. There are a few issues with the pcells when generated: no body connections, no colored metal, and too small of metal. The coloring of the metal layers and increasing their size is easy enough but adding the body connection is worth discussing. In general, my preferred method is to stretch the PC lines up and add a few more RX fins bounded by a new CT layer (approximately 5 RX fins for minimum area). In this new space you can basically add a minimum sized RX area for your "body" connection. Often easiest to get this RX area from a similarly sized FET. Body connection is a bit weird the first time, but pretty easy after doing it once (can just copy from old designs as well).

One of the more annoying issues with the pcells is that if it includes the CT layer, then it will likely not have enough space for a gate connection. To fix this issue, it will be necessary to flatten the pcell and manually stretch/move parts of the design.

One thing you might be curious about is if all the extra PC columns and the large PC columns in the pcells are necessary. The answer is…sometimes. The main thing is that any set of transistors are going to need to be bounded by some number of pcell columns and the large pcell columns. That is to say, if you are using multiple transistors in your design, you can flatten them and overlap some of the inner PC columns and delete the large inner PC columns. As long as the transistor on the farthest right and left side has the supporting PC columns, the design should pass DRC.

Another thing to note when doing analog layout is that if you need to use a resistor, I recommend using rmres. This is one of the more compact resistors, has less things you need to manually adjust, and will pass LVS unlike NWRES.

The last thing to cover is DRC setup. The main thing you need to concern yourself with are the “Customization Settings” that pop-up when you first open DRC.

- DESIGN_TYPE = CELL_NODEN
- IOTYPE = INLINE_50

There are also a couple more customization settings you should use if you are using uncolored STD cells located under “DECOMPOSITION SWITCHES”:

- DP_CHECK_DESIGN_ALL = YES & check the box
- DP_GENERATION_TOOL_ALL = YES & check the box

The decomposition switches essentially fill in the color for uncolored metal layers as best as it can. This will allow you to get away with not coloring all of the metals when using STD cells.

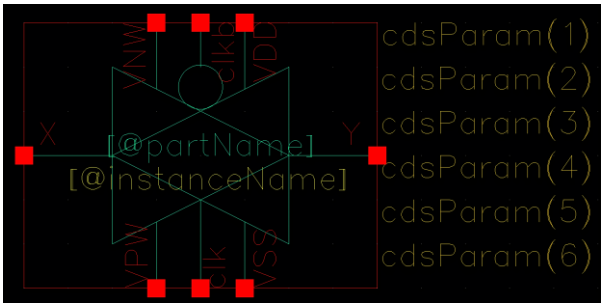
Lastly it should be noted that you can ignore all of the GROUTLINE warnings that will pop-up during DRC checks.

AUX Cell File Generation

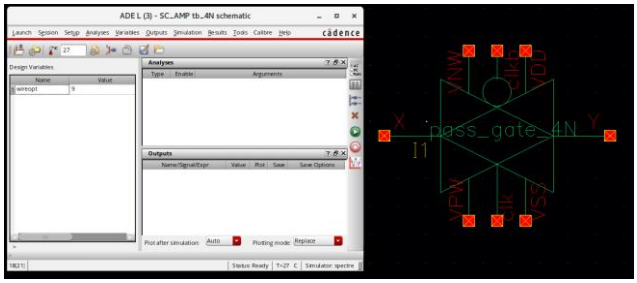
For each AUX Cell, need to generate .sp, .cdl, .lef, .gds, and .lib. This section will cover the creation of each of these file types. Even if you are familiar with how to generate these files, it is recommended you review this section as some of the files require some manual changes after they have been generated.

.sp:

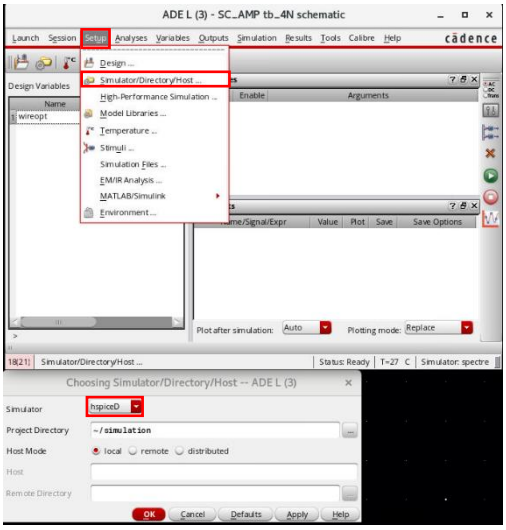
1) Generate symbol for circuit



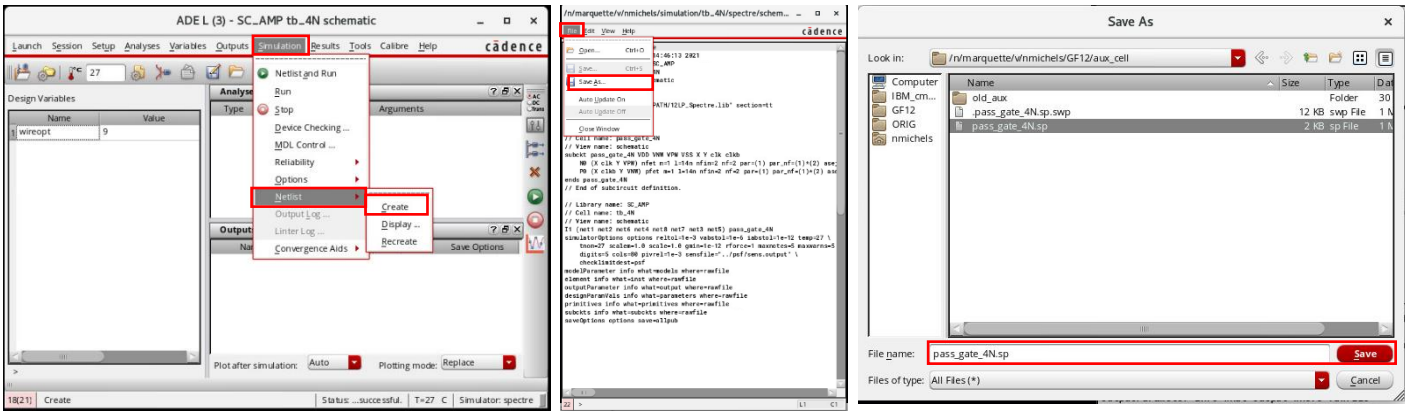
2) Add symbol to a testbench schematic and launch ADEL



3) In ADEL go to Setup -> Simulator/Directory/Host and set Simulator to hspiceD



4) Go to simulation -> Netlist -> Create and save the resulting file as [filename].sp

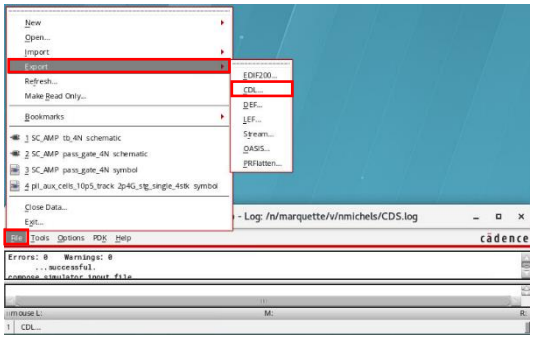


5) Open file and delete the following portions to finish

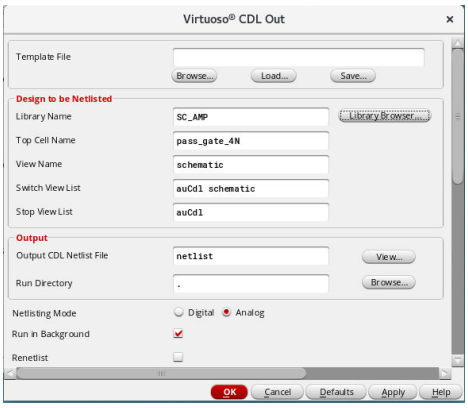
```
1 ** Generated for: hspiceD
2 ** Generated on: May  1 14:15:56 2021
3 ** Design library name: SC_AMP
4 ** Design cell name: tb_4N
5 ** Design view name: schematic
6 .LIB "/afs/eecs.umich.edu/kits/GF/12LP/V1_0_2.1/Models/HSPICE/models/12LP_Hspice.lib" TT
7 .PARAM wireopt=9
8
9
10 .TEMP 25.0
11 .OPTION
12 +    ARTIST=2
13 +    INGOLD=2
14 +    PARHIER=LOCAL
15 +    PSF=2
16
17 ** Library name: SC_AMP
18 ** Cell name: pass_gate_4N
19 ** View name: schematic
20 .subckt pass_gate_4N vdd vnw vpw vss x y clk clkb
21 xn0 x clk y vpw nfet m=1 l=14e-9 nfin=2 nf=2 par=1 par_nf=2 asej=2.64e-15 adej=1.32e-15 psej=524e-9 pdej=240e-9 p
   devdops=1 pdevlgeos=1 pdevwgeos=1 psw_acv_sign=1 plnest=1 pldist=1 plorient=0 cpp=84e-9 fpitch=48e-9 xpos=-99 ypo
   s=-99 ptwell=0 sca=0 scb=0 scc=0 pre_layout_local=-1 ngcon=1 p_vta=0 p_la=0 u0mult_fet=1 lle_sa=77e-9 lle_sb=77e-
   9 lle_rxrxa=84e-9 lle_rxrxb=84e-9 lle_rxrxn=192e-9 lle_rxrxs=192e-9 lle_pcrxn=65e-9 lle_pcrxs=65e-9 lle_nwa=2e-6
   lle_nwb=2e-6 lle_nwn=192e-9 lle_nws=192e-9 lle_ctne=0 lle_ctnw=0 lle_ctse=0 lle_ctsw=0 lle_sctne=0 lle_sctnw=0 ll
   e_sctse=0 lle_sctsw=0 lrsd=30e-9 dtemp=0 l_shape=0 l_shape_s=0 nsig_dop1=0 nsig_dop2=0 nsig_dibl=0 nsig_pc=0 nsig
   _rx=0 fc_index=0 fc_sigma=3 analog=-1 nf_pex=2
22 xp0 x clkb y vnw pfet m=1 l=14e-9 nfin=2 nf=2 par=1 par_nf=2 asej=2.64e-15 adej=1.32e-15 psej=524e-9 pdej=240e-9
   pdevdops=1 pdevlgeos=1 pdevwgeos=1 psw_acv_sign=1 plnest=1 pldist=1 plorient=0 cpp=84e-9 fpitch=48e-9 xpos=-99 yp
   os=-99 ptwell=0 sca=0 scb=0 scc=0 pre_layout_local=-1 ngcon=1 p_vta=0 p_la=0 u0mult_fet=1 lle_sa=77e-9 lle_sb=77e
   -9 lle_rxrxa=84e-9 lle_rxrxb=84e-9 lle_rxrxn=192e-9 lle_rxrxs=192e-9 lle_pcrxn=65e-9 lle_pcrxs=65e-9 lle_nwa=2e-6
    lle_nwb=2e-6 lle_nwn=192e-9 lle_nws=192e-9 lle_ctne=0 lle_ctnw=0 lle_ctse=0 lle_ctsw=0 lle_sctne=0 lle_sctnw=0 l
   le_sctse=0 lle_sctsw=0 lrsd=30e-9 dtemp=0 l_shape=0 l_shape_s=0 nsig_dop1=0 nsig_dop2=0 nsig_dibl=0 nsig_pc=0 nsi
   g_rx=0 fc_index=0 fc_sigma=3 analog=-1 nf_pex=2
23 .ends pass_gate_4N
24 ** End of subcircuit definition.
25
26 ** Library name: SC_AMP
27 ** Cell name: tb_4N
28 ** View name: schematic
29 xi1 net1 net2 net6 net4 net8 net7 net3 net5 pass_gate_4N
30 .END
```

.cdl:

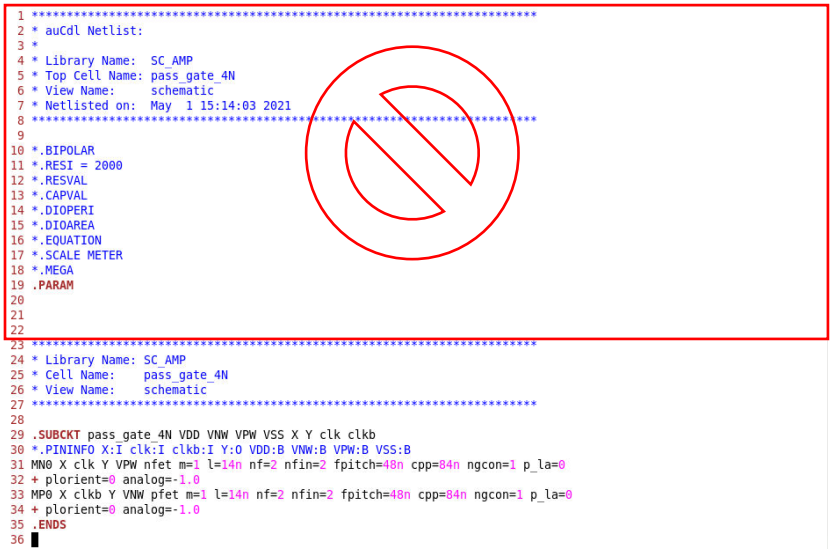
1) From Virtuoso CIW go to files -> Export -> CDL



   a. See example settings below: everything else left default



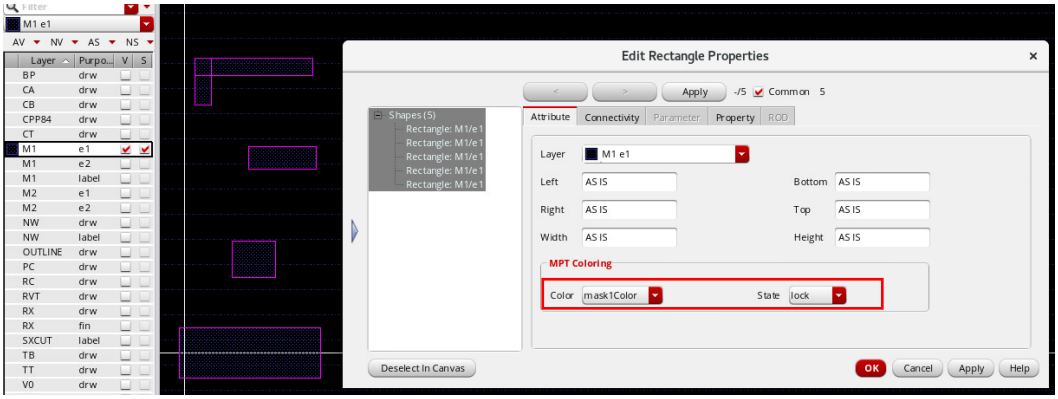
2) Save output netlist file -> Save as [filename].cdl
   a. Sometimes doesn't show output netlist on first try, so may run twice
3) delete the following portion to finish



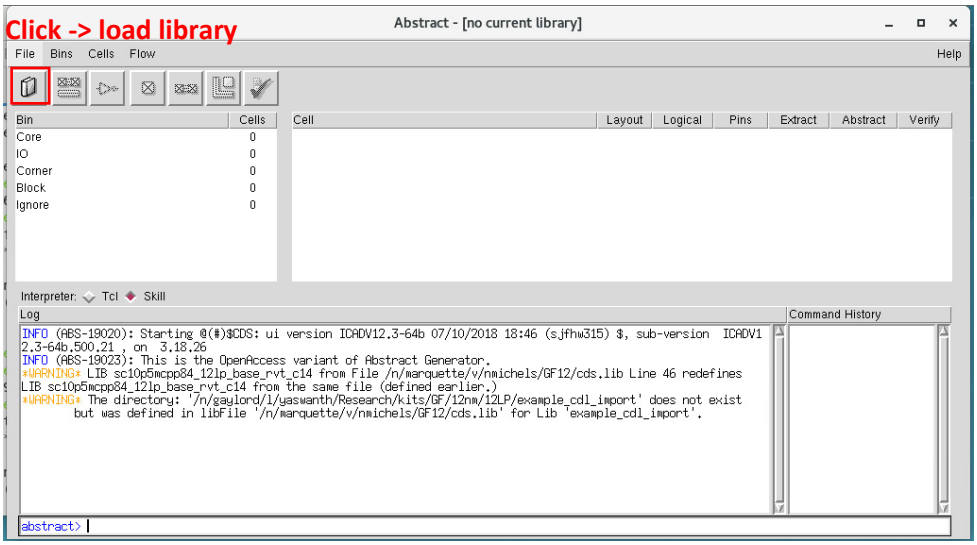
```
1  ********************************************************************************
2  * auCdl Netlist:
3  *
4  * Library Name:  SC_AMP
5  * Top Cell Name: pass_gate_4N
6  * View Name:     schematic
7  * Netlisted on:  May  1 15:14:03 2021
8  ********************************************************************************
9
10 *.BIPOLAR
11 *.RESI = 2000
12 *.RESVAL
13 *.CAPVAL
14 *.DIOPERI
15 *.DIOAREA
16 *.EQUATION
17 *.SCALE METER
18 *.MEGA
19 .PARAM
20
21
22
23 ********************************************************************************
24 * Library Name: SC_AMP
25 * Cell Name:    pass_gate_4N
26 * View Name:    schematic
27 ********************************************************************************
28
29 .SUBCKT pass_gate_4N VDD VNW VPW VSS X Y clk clkb
30 *.PININFO X:I clk:I clkb:I Y:O VDD:B VNW:B VPW:B VSS:B
31 MN0 X clk Y VPW nfet m=1 l=14n nf=2 nfin=2 fpitch=48n cpp=84n ngcon=1 p_la=0
32 + plorient=0 analog=-1.0
33 MP0 X clkb Y VNW pfet m=1 l=14n nf=2 nfin=2 fpitch=48n cpp=84n ngcon=1 p_la=0
34 + plorient=0 analog=-1.0
35 .ENDS
36
```

abstract & .lef:

1) Prior to exporting abstract or .lef file, need to make sure layout metal layers have mask “colors” locked
    a. Open Layout and select all metal layer of a given mask
    b. open properties (q)
    c. set mask color to 1 for e1 layer and 2 for e2 layer, then set state to lock
    d. repeat for each metal layer with masks



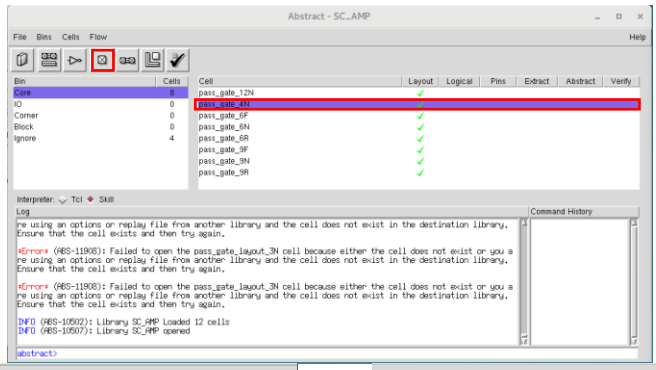
2) From working directory, run /usr/caen/icadv-12.3.500.2100/bin/abstract &
    a. If have source .bashrc_gf12, can use absgen12 (later lefgen12)
    b. Load library you are working in

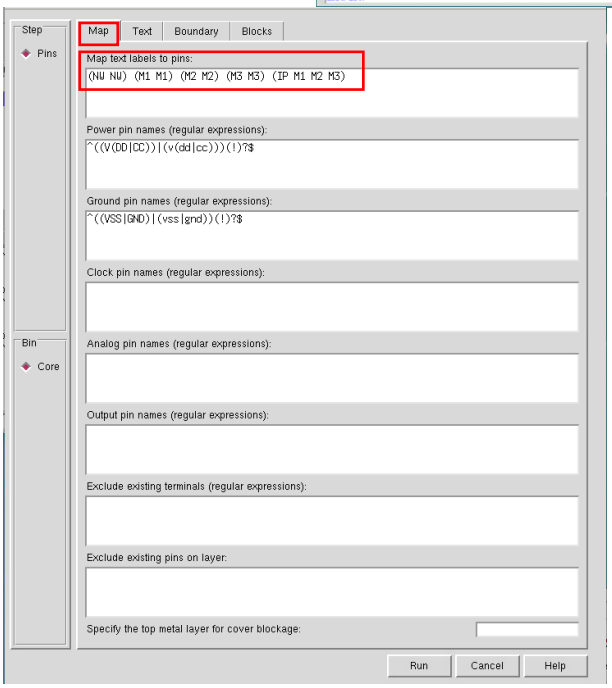


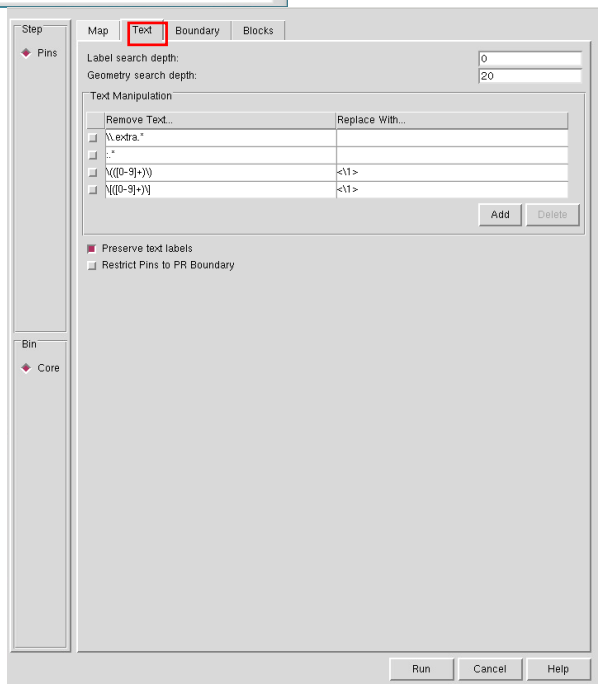
3) Select cellview you want to generate abstract for, then run “pins” with the following settings:

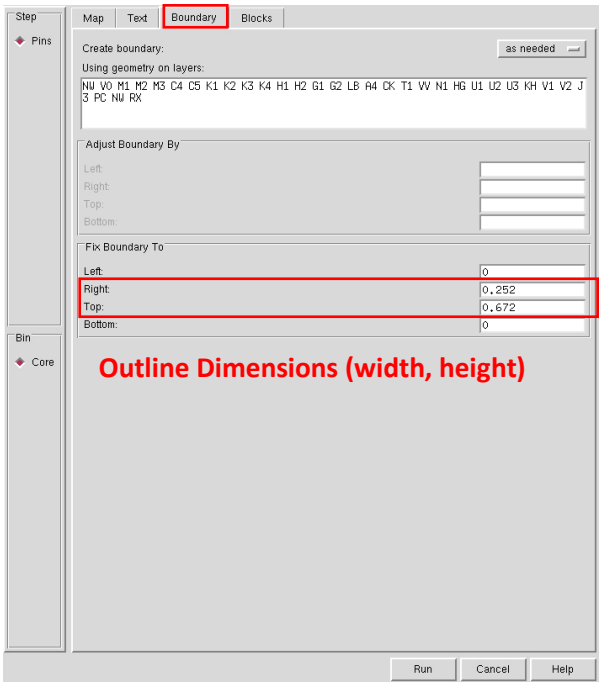
May also add SXCUT layers where appropriate

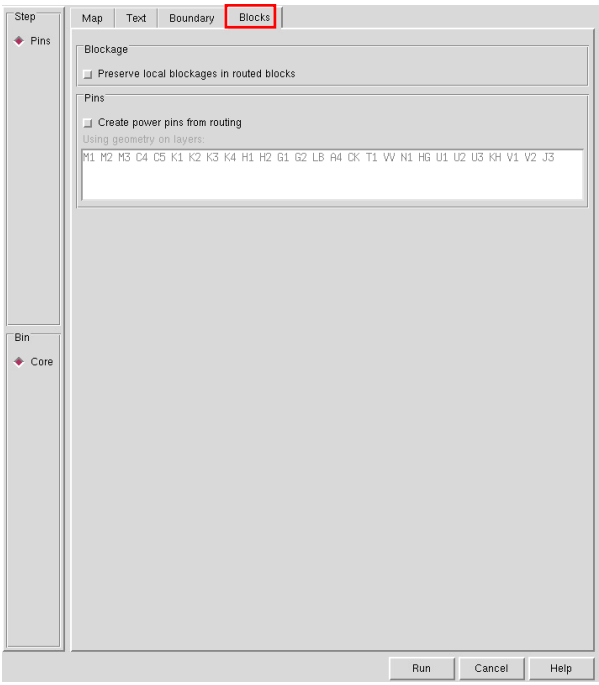
If using higher metal layers, then may need to add these as well. Use your best judgement!



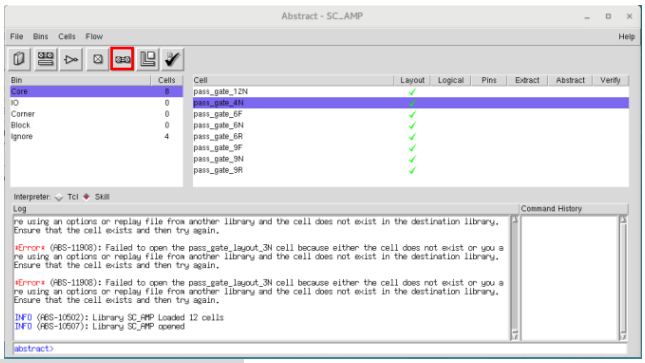


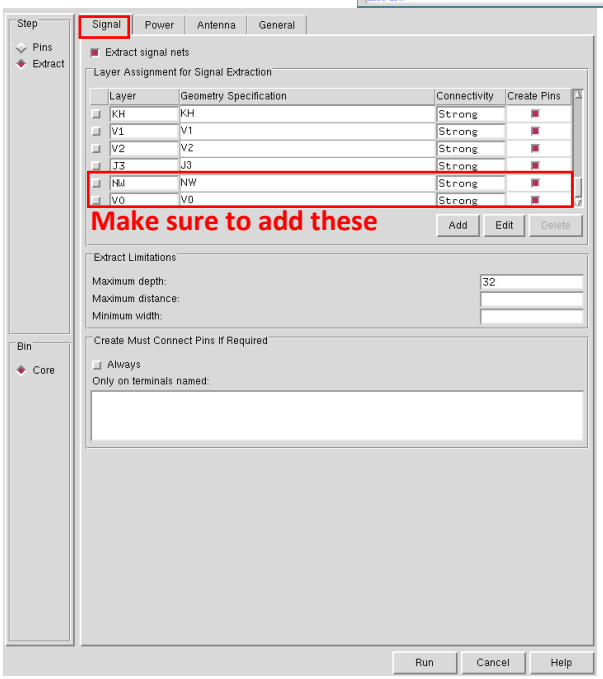


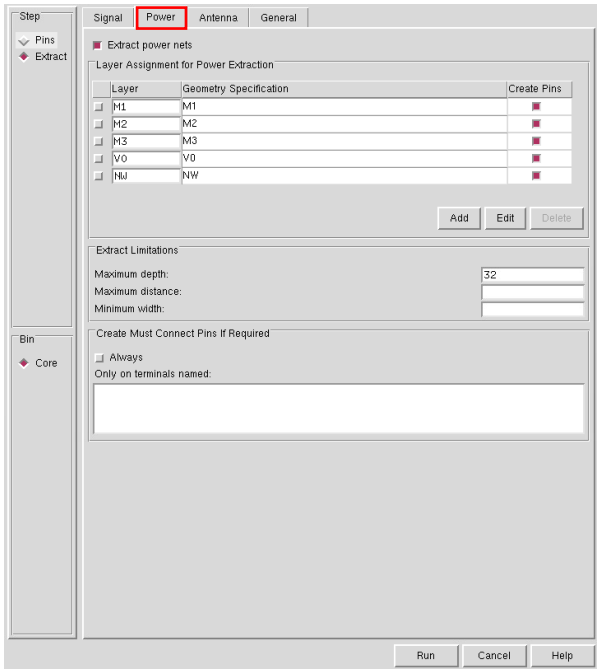


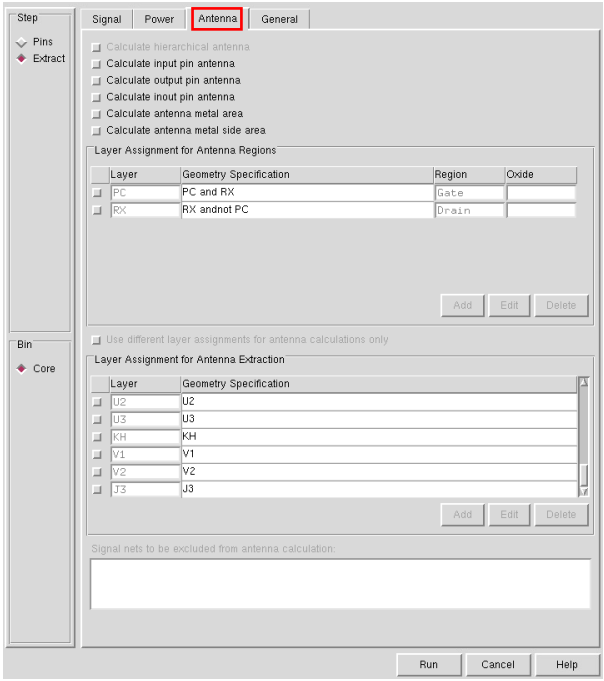


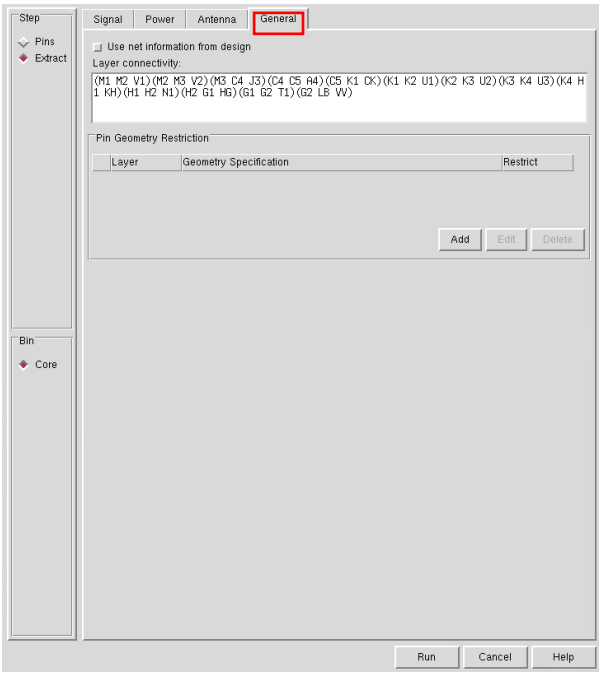
4) Run “Extract” with the following Settings (May also add SXCUT layers where appropriate):



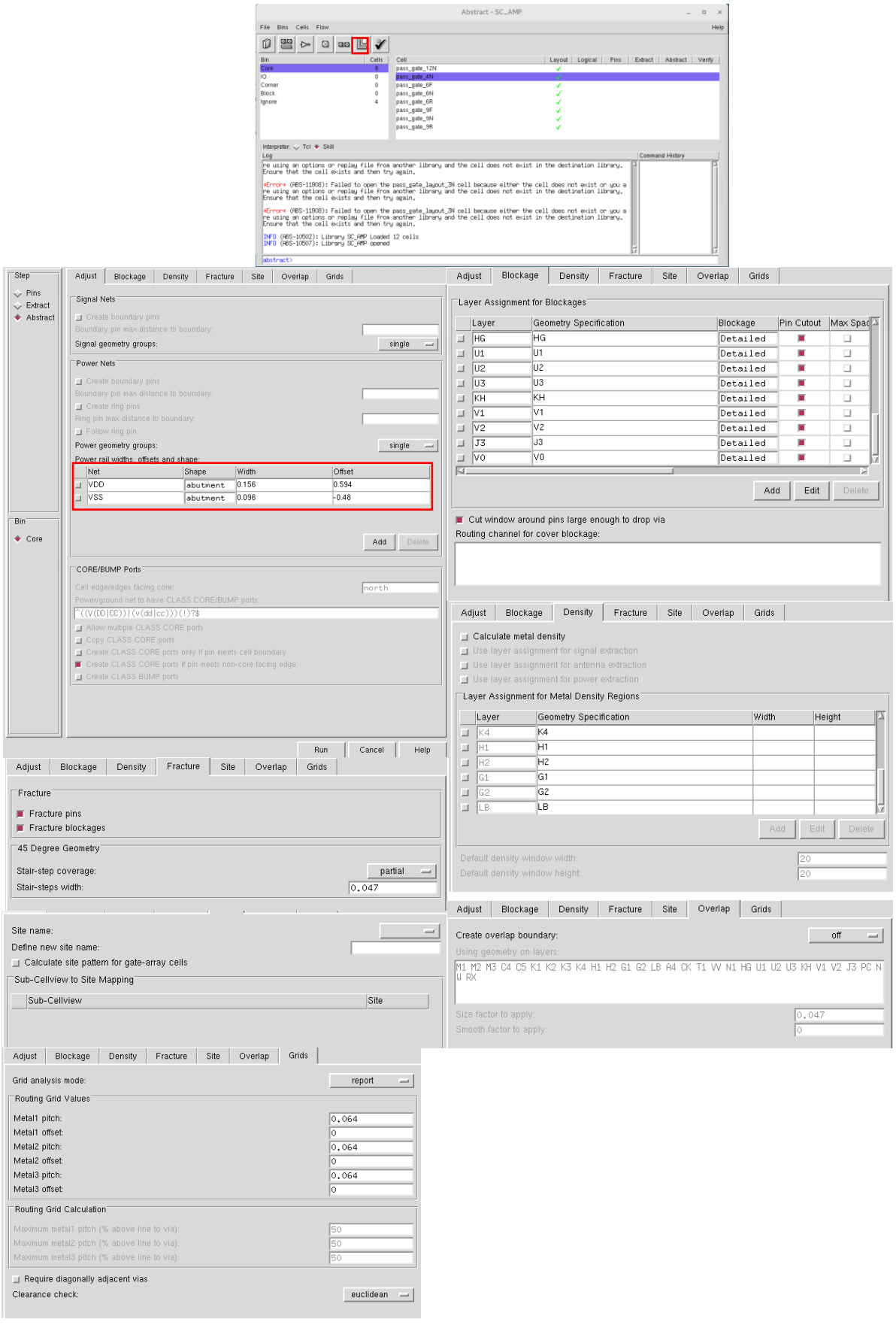




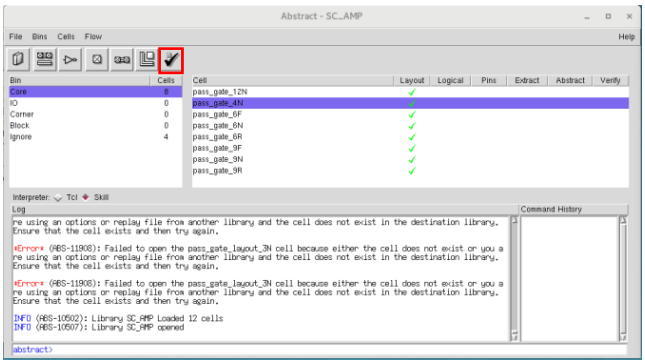


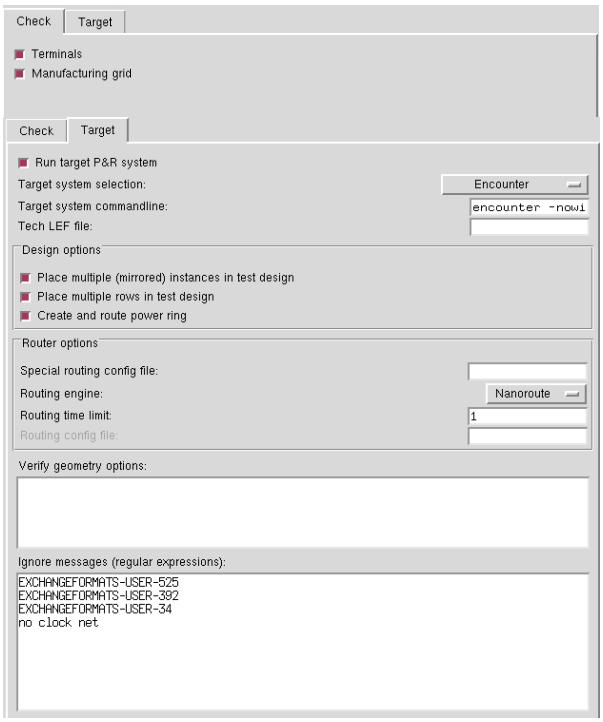


5) Run “Abstract” with the following settings (mostly default, but everything is shown below):

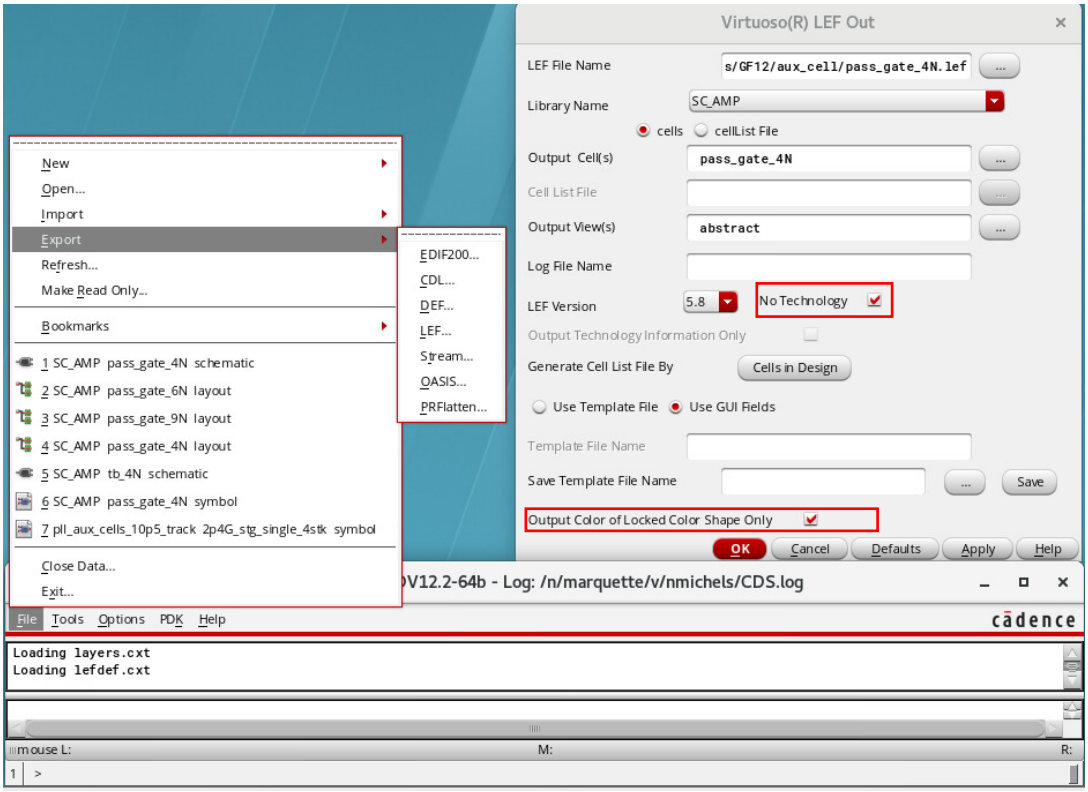


6) Don’t have to run verify, but here are the settings (will probably get some errors messages):





7) From Virtuoso CIW -> File -> Export -> LEF:



8) Open .lef file to make manual modifications to complete
   a. Delete PROPERTYDEFINITIONS
   b. Add SITE custom…
   c. May need to manually change output pins to be OUTPUT if not specified during pins
   d. Probably better way of doing this so it generates it correctly initially

```
1  VERSION 5.8 ;
2  BUSBITCHARS "[]" ;
3  DIVIDERCHAR "/" ;
4
5  PROPERTYDEFINITIONS
6    MACRO CatenaDesignType STRING ;
7  END PROPERTYDEFINITIONS
8
9  SITE custom_10p5mcpp84_12lp
10   CLASS CORE ;
11   SIZE 0.084 BY 0.672 ;
12   SYMMETRY Y ;
13 END custom_10p5mcpp84_12lp
14
15 MACRO pass_gate_4N
16   CLASS CORE ;
17   ORIGIN 0 0 ;
18   FOREIGN pass_gate_4N 0 0 ;
19   SIZE 0.252 BY 0.672 ;
20   SYMMETRY X Y ;
21   SITE CoreSite ;
22   PIN VNW
23     DIRECTION INPUT ;
24     USE SIGNAL ;
25     PORT
26       LAYER NW ;
27         RECT -0.1 0.336 0.352 0.772 ;
28     END
29   END VNW
30   PIN VDD
31     DIRECTION INOUT ;
32     USE POWER ;
33     SHAPE ABUTMENT ;
34     PORT
35       LAYER M1 ;
36         RECT MASK 2 -0.009 0.594 0.261 0.75 ;
37     END
38   END VDD
39   PIN VSS
40     DIRECTION INOUT ;
41     USE GROUND ;
42     SHAPE ABUTMENT ;
43     PORT
44       LAYER M1 ;
45         RECT MASK 1 -0.009 -0.048 0.261 0.048 ;
46     END
47   END VSS
48   PIN X
49     DIRECTION INPUT ;
50     USE SIGNAL ;
51     PORT
52       LAYER M2 ;
53         RECT 0.02 0.087 0.052 0.555 ;
54       LAYER M1 ;
55         RECT MASK 2 0.02 0.08 0.245 0.112 ;
56         RECT MASK 2 0.02 0.08 0.052 0.169 ;
57         RECT MASK 1 0.02 0.53 0.245 0.562 ;
58         RECT MASK 1 0.02 0.473 0.052 0.562 ;
59       LAYER V1 ;
60         RECT 0.02 0.498 0.052 0.53 ;
61         RECT 0.02 0.112 0.052 0.144 ;
62     END
63   END X
64   PIN Y
65     DIRECTION INPUT ;
66     USE SIGNAL ;
67     PORT
68       LAYER M2 ;
69         RECT 0.117 0.136 0.149 0.501 ;
70       LAYER M1 ;
71         RECT MASK 2 0.092 0.427 0.177 0.493 ;
72         RECT MASK 1 0.092 0.144 0.174 0.213 ;
73       LAYER V1 ;
74         RECT 0.117 0.444 0.149 0.476 ;
75         RECT 0.117 0.161 0.149 0.193 ;
76     END
77   END Y
78   PIN clk
79     DIRECTION INPUT ;
80     USE SIGNAL ;
81     PORT
82       LAYER M1 ;
83         RECT MASK 2 0.122 0.25 0.252 0.294 ;
84     END
85   END clk
86   PIN clkb
87     DIRECTION INPUT ;
88     USE SIGNAL ;
89     PORT
90       LAYER M1 ;
91         RECT MASK 1 0.122 0.35 0.252 0.394 :
```

Change direction to INOUT and use to POWER

Change CoreSite to: custom_10p5mcpp84_12lp

```
26   PIN VPW
27     DIRECTION INOUT ;
28     USE GROUND ;
29     PORT
30       LAYER SXCUT ;
31         RECT -0.1 0 0.352 0.336 ;
32     END
33   END VPW
```

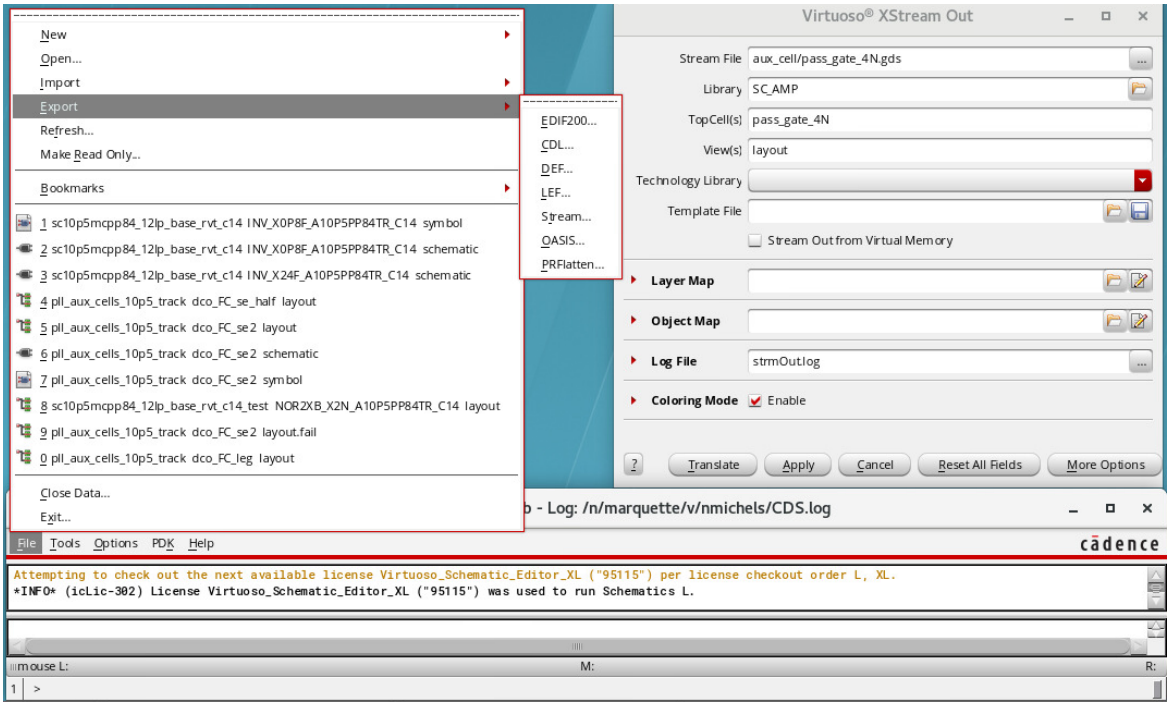
Manually ADD VPW
X-bounds same as VNW
Y-bounds 0 & bottom of VNW

Y should be OUTPUT (in this example) Can specify inputs/outputs during .lef creation as well.

May be line near bottom which needs to be removed:
PROPERTY CatenaDesignType “deviceLevel” ;

.gds:

1) From Virtuoso CIW go to File -> Export -> Stream and load file and translate



.lib:

Setup for this is very manual. Just copy an old .lib file and replace pin names/directions with yours. This file is just placeholder basically to pass through Cadre Flow later.

Great! You've now setup your AUX Cells and are ready to start working in the FASoC side of things. All of these files you've created will go into blocks folder in your block generator directory, but before we move them, let's go ahead and take a look at all the files and what they are used for. You will also need to clean out or modify all of the old files in the directory since you copied them from a different generator. This will all be covered in the next section.

FASoC Block Generator

Before working in your fasoc directory, be sure to source the .tcshrc that was mentioned earlier. The modc12 alias loads the necessary modules.

```
$source .tcshrc
$modc12
```

This section will cover the basic file structure for the fasoc generators, and the steps to make the design.

./

Let's start with the top-level files. The main file here is the include.mk. The main change you need to make to this file is updating the DESIGN_NAME to your block name; The rest can probably be left as default.

./blocks/

Next let's look at the **./blocks** directory. This directory is where you will store all of you aux cell files. The general structure used for the aux cells is ./aux_cell_name/export/rest_of_files. Put all of the aux cell files you generated in the last section into this directory.

./src/

Now let's look at the **./src** directory. This is where you will have your blocks Verilog code for both the top level and the aux cells. The aux cell Verilog blocks are basically black boxes where you just specify the inputs and outputs to the blocks. Note that power/ground shouldn't be specified in any of these Verilog blocks. All power routing is done later. The top-level Verilog block should be your design_name.v. This is where you will specify the details of your block. The general structure of the top-level Verilog should be defining your parameters (general things which change depending on auto generated design) followed by using generate and genvar which essentially generate the rest of the Verilog code depending on the parameters. Refer to other generators as reference.

./scripts/dc/

The next directory is the scripts directory, which is divided into **./scripts/dc** and ./scripts/innovus. The dc portion of scripts is used to get through synthesis, and innovus is used for everything after. Let's focus on the dc part for now:

- **constraints.tcl:** Used to specify clk and set_dont_touch for certain nets/cells you don't want altered.
    - Must have a clk for FASoC flow to work. Can use dummy clk.
- **dc.filelist.tcl:** Specify the Verilog files to be used in design
- **dc.include.tcl, dc.read_design.tcl, dc.setup.tcl, report_timing.tcl:** no change necessary

Once these are setup you can run "make synth" and check the results in **./results/dc/block_name.mapped.v** to see if it is as expected. If you run into errors before the design completes, you can check **./logs/dc/synth.log**. Once you have updated files, use "make bleach_synth" to clear the old design. If still having trouble after reading through log, reach out for assistance.

Assuming you made it through synth, we can now move on to the innovus scripts and the rest of the flow.

./scripts/innovus/

After synth, the rest of the APR stages are as follows: init, place, cts, postcts_hold, route, postroute, signoff. To run each of these stages use the "make [stage]" command. Can also use "make debug_[stage]" command to bring up a GUI to see what has been done and check the results. The files in ./scripts/innovus/ are a combination of general files used for top level and multiple steps (EX: always.tcl, floorplan.tcl) and files which are run before or after each of the stages (EX: pre_[stage].tcl, post_[stage].tcl). You will mostly be concerned with the general files and the files up to pre_place.tcl (rest of stages are mostly default). Use "make bleach_[stage]" to clear any stage and rerun after changes. Also use "make bleach_all" to clear everything (synth and apr).

- **always_source.tcl:** Describes general block sizing and metal/via layers. Main thing to update in this file will be the core_width and core_height of your design. Used in every stage.
- **floorplan.tcl:** Mostly grabs sizing from always_source.tcl, but can add cutrow areas to the floorplan as well as sourcing a custom_place.tcl for your aux cells. Used in init.
- **custom_place.tcl:** This is a file you will have to write python code to generate for automating design. This file is used to specify locations for aux cells. Not necessary to use in theory, but often APR will place things in odd locations if not specified. An example of how a custom_place is written is shown below. The R0 & MX stand for regular orientation and mirrored x-axis. There is also a mirrored y-axis and diagonal.

```
1 #Place pass_gate cells
2 placeInstance pass_input [expr $core_margin_l+17.294] [expr $core_margin_b+20.592] R0 -fixed
3 placeInstance even_final_pass_plus [expr $core_margin_l+17.294] [expr $core_margin_b+21.264] MX -fixed
```

- **setup.tcl:** Grabs info on aux cells. May want to change welltaps intervals depending on design sizing.
- **power_intent.cpf:** Connects global power signals
- **innovus_config.tcl:** Can specify extra rules for innovus such as welltaps spacings.
- **powerplan.tcl:** Used to specify power rings/mesh. Update according to design needs.
- **pre_place.tcl:** Specify your design's pins and locations. Also set_dont_touch your blocks again if necessary.
- Main changes listed above but refer to other **pre/post_[stage].tcl** files to see what they are doing.

Finishing Up

Go through each step using "make debug_[stage]" to see if it works. If you encounter issues, refer to the logs to see the error. Feel free to reach out if having trouble.

Once you get through "make signoff" without any errors, you can check lvs and drc. Also note that if you make changes and just want to rerun the whole design you can use "make design" to go all the way through signoff. When running lvs/drc, can't just use make debug_lvs/drc. You have to run "make lvs" then "make debug_lvs" and same for drc.

Once you are DRC/LVS clean, then congratulations! You now have made a block using the FASoC flow! The next step is to write some python codes that will automatically generate/alter certain files depending on desired specs of your block. This will vary greatly from design to design, so this won't be covered in much detail. Feel free to look at the pymodules used in some other generators to get an idea of what needs to be changed. Good luck!

When everything is finalized, use "make export" to get an export directory needed for top-level tapeout. Some changes will need to be made to export, such as copying block_name_cutObs.lef over the lef file in export.

Before moving on to top-level, you should perform pre/post-PEX simulations on your design. If you find an issue now before starting on top-level, it will save you a lot of time down the road. The basics of how to simulate your design will be covered in the next section.

Post-PEX Simulations

This section will cover the basics of testing your FASoC design. The methods used here are not necessarily the only way to do it, but should serve as a good point to get you started with simulating your design.

PEX Extraction of Design

The first step is going to be to make an extraction directory and a simulation directory. The structure of these directories can be copied from one of the other generators. You will then need to make a python code for performing PEX on your design. You can copy one of the run_pex_flow.py files from another directory and alter it to work for your design. This step can be a bit tricky, so feel free to reach out if having difficulties. Once the extraction directory and run_pex_flow.py are set up, you should be able to generate the PEX netlist of your design.

The file structure for my extraction directory is as follows:

- extraction/
  - layout/
  - ruleFiles/
  - run/
  - runsets/
  - sch/
  - gf12.pex.bashrc

Here is what each of the subdirectories contain:

- **layout:** contains design_name.gds.gz which can be found in ./results/calibre/design_name.merged.gds.gz.
- **ruleFiles:** contains lvs/drc/rcx rules. Just copy these files from another generator.
- **run:** contains results after running PEX extraction. Will copy necessary files from here to simulation directory after running extraction.
- **runsets:** contains pex.runset files for various technology nodes. Copy these files from another generator.
- **sch:** contains cdl netlist after running PEX. Just copies netlist file from ./results/calibre/lvs.
- **gf12.pex.bashrc:** file containing gf12 information. Not used if using method of PEX that I used.

The setup for the simulation directory is simply making an empty simulation directory. It is basically just a separate location for you to store the results of the simulations you run.

Once you have your directories setup, you can work on your python code. The code I used is shown below on the next page. If you are using a similar extraction directory to me, you can use this code and alter the file locations to get it to work. This code is of course geared towards gf12, so if you are working in a different technology then more changes will be necessary. Code should exist for TSMC65 as well, so feel free to reach out to someone for the code for this technology.

Python Code for running PEX extraction:

```python
#!/usr/bin/env python3

#-----------------------------------------------------------------------------
# POST-PEX NETLIST GEN
# IDEA & POSH
# Date: 08/29/2019
#-----------------------------------------------------------------------------
import os
import glob
import shutil
import subprocess as sp

def gen_post_pex_netlist(platform, designName, flowDir, extDir, calibreRulesDir):
    # Copy the cdl netlist to extraction directory
    for file in glob.glob(flowDir+'/results/calibre/lvs/_'+designName+'*.sp'):
        shutil.copy(file, extDir+'/scf/'+designName+'.spi')

    # Copy the merged gds file to extraction directory
    p = sp.Popen(['cp', flowDir+'/results/calibre/'+designName+'.merged.gds.gz', \
                  extDir+'/layout/'+designName+'.gds.gz'])
    p.wait()

    # Clean the space for PEX
    if os.path.isfile(extDir + '/run/svdb/' + designName + '.dv'):
        os.remove(extDir + '/run/svdb/' + designName + '.dv')
    if os.path.isfile(extDir + '/run/svdb/' + designName + '.extf'):
        os.remove(extDir + '/run/svdb/' + designName + '.extf')
    if os.path.isfile(extDir + '/run/svdb/' + designName + '.lvsf'):
        os.remove(extDir + '/run/svdb/' + designName + '.lvsf')
    if os.path.isfile(extDir + '/run/svdb/' + designName + '.pdsp'):
        os.remove(extDir + '/run/svdb/' + designName + '.pdsp')
    if os.path.isfile(extDir + '/run/svdb/' + designName + '.sp'):
        os.remove(extDir + '/run/svdb/' + designName + '.sp')

    if os.path.isdir(extDir + '/run/svdb/' + designName + '.phdb'):
        shutil.rmtree(extDir + '/run/svdb/' + designName + '.phdb',
                      ignore_errors=True)
    if os.path.isdir(extDir + '/run/svdb/' + designName + '.xdb'):
        shutil.rmtree(extDir + '/run/svdb/' + designName + '.xdb',
                      ignore_errors=True)
    if os.path.isdir(extDir + '/run/svdb/' + designName + '.pdb'):
        shutil.rmtree(extDir + '/run/svdb/' + designName + '.pdb',
                      ignore_errors=True)
    if os.path.isdir(extDir + '/run/svdb/' + 'template'):
        shutil.rmtree(extDir + '/run/svdb/' + 'template',
                      ignore_errors=True)

    if platform == 'gf12lp':
        os.environ['TECHDIR'] ='/afs/eecs.umich.edu/kits/GF/12LP/tapo_V1.0_3.1'
        os.environ['BEOL_STACK'] ='13M_3Mx_2Cx_4Kx_2Hx_2Gx_LB'
        os.environ['TECHDIR_LVS'] ='/afs/eecs.umich.edu/kits/GF/12LP/tapo_V1.0_3.1/LVS/Calibre'
        os.environ['TECHDIR_XACT'] ='/afs/eecs.umich.edu/kits/GF/12LP/tapo_V1.0_3.1/PEX/xACT'
        os.environ['PEX_RUN'] = 'TRUE'
    # Configure the PEX rule files

    for file in os.listdir(calibreRulesDir + '/'):
        if not os.path.isdir(calibreRulesDir + '/' + file):
            shutil.copy2(calibreRulesDir+'/'+file, extDir+'/run/')

    with open(extDir+'/runsets/pex.runset.'+platform, 'r') as file:
        filedata = file.read()
    filedata = filedata.replace('design', designName)
    with open(extDir+'/run/pex.runset', 'w') as file:
        file.write(filedata)

    # Run Calibre RCX
    if platform == 'gf12lp':
        p = sp.Popen(['calibre','-gui','-xact','-batch','-runset',
                      'pex.runset'],cwd=extDir+'/run')
        p.wait()
    else:
        p = sp.Popen(['calibre','-gui','-pex','-batch','-runset',
                      'pex.runset'],cwd=extDir+'/run')
        p.wait()

gen_post_pex_netlist('gf12lp', 'sc_amp', '/net/marquette/v/nmichels/fasoc/private/generators/pll-gen/gf12lp/ignore_sc_amp', '/net/marquette/v/nmichels/fasoc/private/generators/pll-gen/gf12lp/ignore_sc_amp/extraction',
                     '/net/marquette/v/nmichels/fasoc/private/generators/pll-gen/gf12lp/ignore_sc_amp/extraction/ruleFiles')
```

Next you will move the three design_name.pex.netlist* files from your extraction/run directory to your simulation directory:

- **simulation/**
    - **design_name.pex.netlist**
    - **design_name.pex.netlist.pex**
    - **design_name.pex.netlist.pxi**

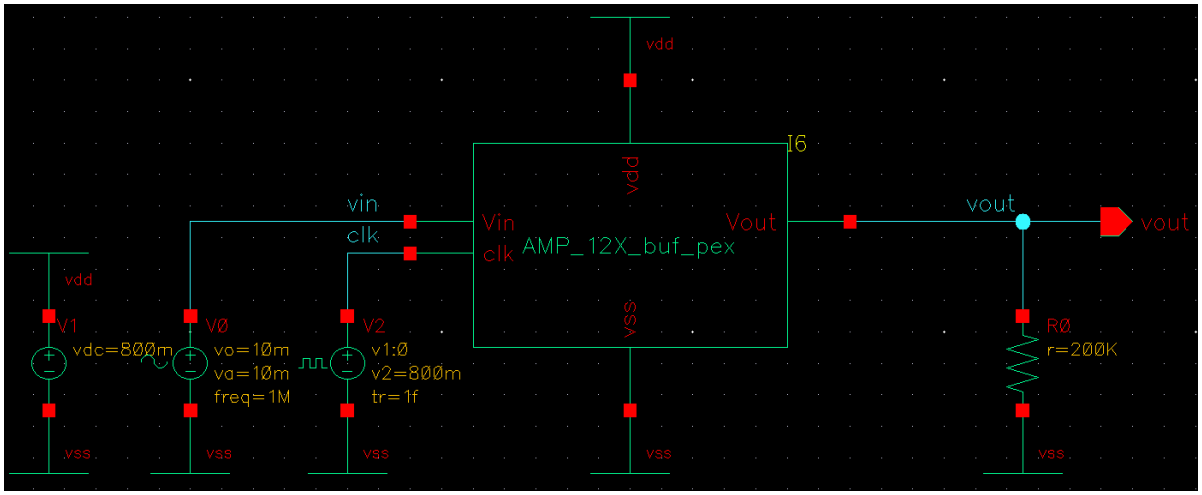
At this point you will need a spice testbench to properly simulate your design. If you are familiar with spice testbench files, you can manually create a testbench or alter an existing generator's testbench. The other option is to generate a testbench using virtuoso, which will be covered in the next section.

Spice Testbench Generation

Note: *Reference .sp AUX cell generation for how to change simulator to HSpice and generate netlist if you are unsure how to do this.*

1) Create a basic testbench of your design in virtuoso:

Example:



2) Launch ADE and change simulator to HSpice.

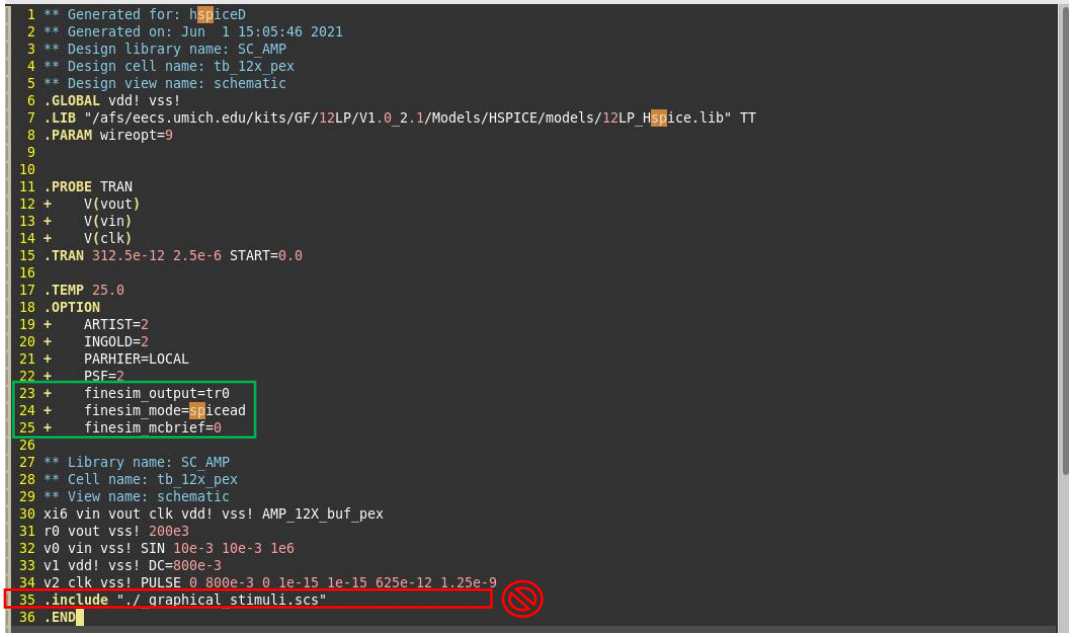
3) Change analysis settings to whatever you need

4) Create netlist

5) Modify netlist as follows
    a. Remove all subcircuit definitions

```
** Generated for: hspiceD
** Generated on: May 31 16:12:09 2021
** Design library name: SC_AMP
** Design cell name: tb_12x_pex
** Design view name: schematic
.GLOBAL vdd! vss!
.LIB "/afs/eecs.umich.edu/kits/GF/12LP/V1.0_2.1/Models/HSPICE/models/12LP_Hspice.lib" TT
.PARAM wireopt=9


.PROBE TRAN
+    V(vout)
+    V(vin)
+    V(clk)
.TRAN 312.5e-12 2.5e-6 START=0.0

.TEMP 25.0
.OPTION
+    ARTIST=2
+    INGOLD=2
+    PARHIER=LOCAL
+    PSF=2

** Library name: sc10p5mcpp84_12lp_base_rvt_c14
** Cell name: INV_X0P6N_A10P5PP84TR_C14
** View name: schematic
.subckt INV_X0P6N_A10P5PP84TR_C14 a vdd vnw vpw vss y
xmmny y a vss vpw nfet m=1 l=14e-9 nfin=3 nf=1 par=1 par_nf=1 asej=1.188e-15 adej=1.188e-15 psej=238e-9 pdej=238e-9 pdevdo
xmmpy y a vdd vnw pfet m=1 l=14e-9 nfin=3 nf=1 par=1 par_nf=1 asej=1.188e-15 adej=1.188e-15 psej=238e-9 pdej=238e-9 pdevdo
.ends INV_X0P6N_A10P5PP84TR_C14
** End of subcircuit definition.

** Library name: SC_AMP
** Cell name: CAP_UNIT_800f
** View name: schematic
.subckt CAP_UNIT_800f bot top
xc0 top bot mimcap w=6.305e-6 l=6.305e-6 nrows=1 ncols=1 slots=0 dtemp=0
.ends CAP_UNIT_800f
** End of subcircuit definition.
```

b. Remove .include and add finesim options (also add .include for your design_name.pex.netlist)



```
** Generated for: hspiceD
** Generated on: Jun  1 15:05:46 2021
** Design library name: SC_AMP
** Design cell name: tb_12x_pex
** Design view name: schematic
.GLOBAL vdd! vss!
.LIB "/afs/eecs.umich.edu/kits/GF/12LP/V1.0_2.1/Models/HSPICE/models/12LP_Hspice.lib" TT
.PARAM wireopt=9


.PROBE TRAN
+    V(vout)
+    V(vin)
+    V(clk)
.TRAN 312.5e-12 2.5e-6 START=0.0

.TEMP 25.0
.OPTION
+    ARTIST=2
+    INGOLD=2
+    PARHIER=LOCAL
+    PSF=2
+    finesim_output=tr0
+    finesim_mode=spicead
+    finesim_mcbrief=0

** Library name: SC_AMP
** Cell name: tb_12x_pex
** View name: schematic
xi6 vin vout clk vdd! vss! AMP_12X_buf_pex
r0 vout vss! 200e3
v0 vin vss! SIN 10e-3 10e-3 1e6
v1 vdd! vss! DC=800e-3
v2 clk vss! PULSE 0 800e-3 0 1e-15 1e-15 625e-12 1.25e-9
.include "./_graphical_stimuli.scs"
.END
```

c. Add any additional sources (Ex: VSS) or passives. Modify to whatever you need.

*Note: It is important to make sure your inputs and outputs are listed in the same order as in your design netlist. So, in the example shown above, the “xi6 vin vout clk vdd! vss! AMP_12x_buf_pex” order is important. Double check your netlist if you switch between pre- and post-PEX or if you regenerate the PEX netlist as it can sometimes change.

Simulating Design

To simulate the testbench, use either finesim or HSpice. Finesim will generally run much faster. After running design, use waveviewer to see the result

```
$finesim -log fs.log -np 4 design_name.sp      or      $hspice design_name.sp -mt 20
$wv sc_amp2.tr0 &
```

The next section will cover the top-level of adding your design along with others to the final chip with pads. This is a lot of work, so don’t expect it to be as simple as dropping your design on the chip and doing some routing. Try to have at least a week to get through this portion!

Pre-PEX Simulations

Performing pre-PEX simulations only requires the pre-PEX design netlist. This can be found in calibre/lvs/ or in the extraction/sch/ directory after running PEX. You may have to make some changes to the cdl of the blocks or some other files in terms of how the MOSFETs are written. An example is shown below with pre-PEX on left.

*Note: Pre-PEX and post-PEX can be very different. Don't rely on pre-PEX for determining performance.

```
1  ********************************************************************************
2  * Library Name: SC_AMP
3  * Cell Name:    pass_gate_12NW
4  * View Name:    schematic
5  ********************************************************************************
6
7  .SUBCKT pass_gate_12NW VDD VNW VPW VSS X Y clk clkb
8  *.PININFO X:I clk:I clkb:I Y:O VDD:B VNW:B VPW:B VSS:B
9  XMN0 X clk Y VPW nfet m=1 l=14n nf=4 nfin=3 fpitch=48n cpp=84n ngcon=1 p_la=0
10 + plorient=0 analog=-1.0
11 XMP0 X clkb Y VNW pfet m=1 l=14n nf=4 nfin=3 fpitch=48n cpp=84n ngcon=1 p_la=0
12 + plorient=0 analog=-1.0
13 .ENDS
14
15
```

```
1  ********************************************************************************
2  * Library Name: SC_AMP
3  * Cell Name:    pass_gate_12NW
4  * View Name:    schematic
5  ********************************************************************************
6
7  .SUBCKT pass_gate_12NW VDD VNW VPW VSS X Y clk clkb
8  *.PININFO X:I clk:I clkb:I Y:O VDD:B VNW:B VPW:B VSS:B
9  MN0 X clk Y VPW nfet m=1 l=14n nf=4 nfin=3 fpitch=48n cpp=84n ngcon=1 p_la=0
10 + plorient=0 analog=-1.0
11 MP0 X clkb Y VNW pfet m=1 l=14n nf=4 nfin=3 fpitch=48n cpp=84n ngcon=1 p_la=0
12 + plorient=0 analog=-1.0
13 .ENDS
14
```

Top-Level for Tapeouts

The top-level is where everybody's designs will be added onto one chip and routed to pads. If you have never worked with adding pads before, then you should know that this step is likely more work than you are anticipating. Make sure to set aside an adequate amount of time to get through this. Lots of errors and lots of waiting for designs to finish.

***Note:** Some of the files will change depending on tapeout, and definitely anything related to dates. There are also a couple shared directories where you will need to add your blocks and aux cells for everything to work.*

The first step is going to be to create a new ~/fasoc_tapeout_[date]/ directory for you to work in. Follow the steps used to make the first fasoc directory. Next you will navigate to the tapeout directory, which for the example tapeout I will be using was ~/fasoc_tapeout_[date]/fasoc/private/tests/fasoc_to_gf12_2021/. This is where the main files you will change will be located.

./

The top-level file structure is actually quite similar to the block level file structure. It still uses the include.mk, ./src/, and ./scripts/. One new directory is the ./fasoc_soc/ directory, which will be covered later. For now, let's update the include.mk

- **include.mk:** The main change you need to make here is to make a block subdirectory for your design and import your block from the shared directory being used for this tapeout. For this tapeout the shared directory was located at /afs/eecs.umich.edu/cadre/projects/fasoc/tapeout_gf12_2021/blocks/[block_name]. Follow other designs and copy your block here and update the include.mk.

./fasoc_soc/soc_top/

Next let's look at the fasoc_soc/soc_top files needed to update. There is too much to go into detail here and other files, so just do best to refer to existing file and feel free to ask for assistance.

- **fasoc_pin_mux.sv:** Here all of the pads will be specified and the input/output wires to the pads
- **fasoc_testchip.sv:** Here you will specify pad I/Os and load your block module and specify connections.

./src/ & ./scripts/dc/

Next let's consider the **./src/** files… actually let's not because I didn't use any! Hopefully someone else can update this section in the future. My best guess is that it is used to define connections between multiple blocks on the top level.

Moving on to the **./scripts/dc/** you will find this section is pretty much the same as the block level, so there isn't really much new to discuss here either.

./scripts/innovus/

Lastly is the **./scripts/innovus** which also unfortunately for you does have some new stuff and changes. Possibly the first difference you will notice is the addition of the new powerplan.tcl, io_floorplan.tcl, padring.io, power_intent.cpf files. Let's go over each of these and some other files that will need updating.

- **BlockNamePowerPlan.tcl:** Connects your VDD/VSS to pads.
- **io_floorplan.tcl:** Specify non VSS power pads for your block
- **padring.io:** Place pads in actual locations
- **power_intent.cpf:** Connects internal pad power rings to each other. Refer to the io document for more info.
- **Other:** Other files are similar to block level, but refer to each to see what it is doing.

There is also one more shared directory you will need to copy files to before you can run your design. go to "/afs/eecs.umich.edu/cadre/projects/fasoc/share/aux_lib_gf12lp_10p5_track" and copy your aux-cells there and mimic the file structure of other blocks. Make a directory with today's date and tag that folder as "latest". Tag example(type this in the command line): ln -s 2020_05_19 latest. So the cadre flow goes to this shared directory and grabs the aux-cells for the cdl and stuffs that are needed for LVS.

Verifying Design

Once you have everything setup, use similar make commands as before to run check design. Before running make synth or make design, need to run "make blocks" which grabs all of the blocks from the shared folder. Then can run as normal. When you get to checking lvs/drc, always check lvs first as this is much quicker. For a quick drc check, you can use "make drc_beol" which can be used to see some quick errors. The drc_beol will have some errors that can be ignored, and also won't check all errors, so will need to eventually run make drc. If you have passed all of that, there is an additional "make dfm" which will check manufacturing rules. Note that drc and dfm can be run in parallel if you first run "make gds_top" and wait for it to finish. You will almost certainly have some errors in lvs/drc, so don't hesitate to reach out for assistance if you can't figure it out.

Once all of that is clean, you can work to get your design on GitHub.

Working with GitHub

First a bit of a warning: I am by no means an expert at GitHub. This is basically just to cover the bare minimum commands/steps you will need to know to be able to keep up with changes being pushed to GitHub and how to upload your own changes. If there is anything you would like to add, please let me know and I can update this section.

Merging Changes

Okay! Let's assume that you've verified your design and are ready to upload your change and be done! Oh wait…. it looks like someone just pushed their changes before you ☹! So now it is your job to merge your changes with what was just pushed. To do this, follow these steps:

1) Check that the files you want to commit are ready (git status)
   a) Look to see what files are set to be merged and use "git add" to add files you want to be committed if they aren't already there
2) Commit your changes (git commit -m "message about commit")
3) Pull the most recent changes (git pull)
   a) This will try to resolve conflicts but will likely fail for some files
4) See which files were not merged fully (git status)
5) Go to files and search for HEAD and merge files manually (gvim ./file; /HEAD)
6) Commit again after merging changes (git commit -m "message about commit")
   a) May have "git add" the files again before committing
7) check status again (git status)
   a) Shouldn't be any unmerged

Great! Now you are back to being on track to push your changes! Make sure to remake the design and check lvs/drc/dfm again before uploading your files. Change may have caused new issues. Once your design is back to being clean, you should be ready to push your design. It is good practice to communicate with whoever else is working on the chip before pushing your design just to double check if everything is okay. People may get angry, but everyone is just frustrated from all of the work during tapeout, so don't take too seriously.

Pushing Final Changes

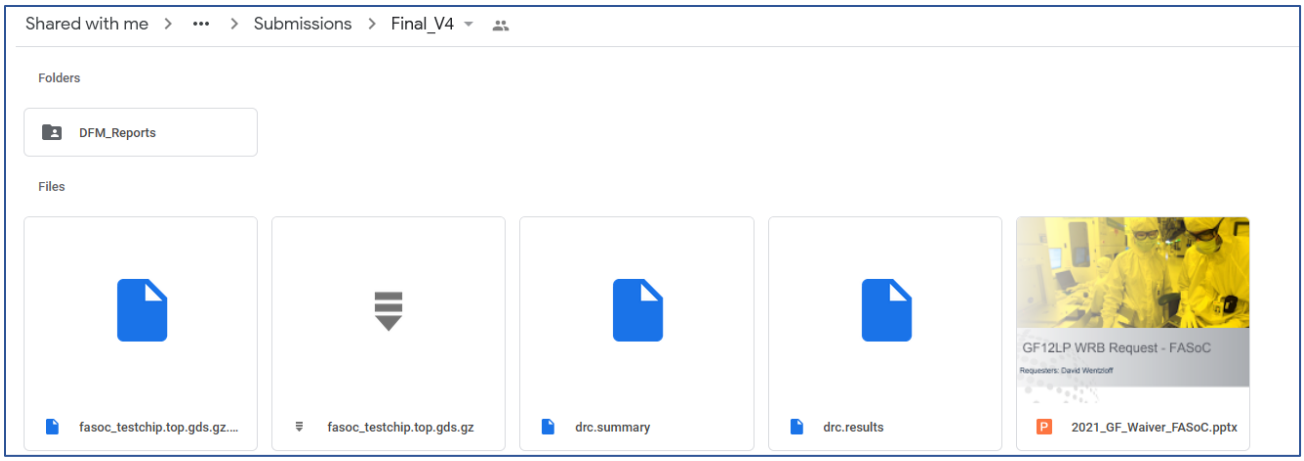
Now that everything is back up and running, we are ready to push our final changes to GitHub by following these steps:

1) cd old_fasoc_5_16/fasoc/private/
2) git push origin master
3) cd ..
4) (old_fasoc_5_16/fasoc/)
5) git add private
6) git commit -m "Updating private to latest"
7) git push origin master

Yay! You've uploaded your block and are free at last! The next section will cover how to do the final upload for the tapeout, but you will likely not be responsible for this if you are a newer student. It is still nice to reference though, so feel free to stick around!

Final Tapeout Upload

For the final upload, there is a google drive shared folder where certain files will need to be uploaded. For this tapeout it was in “2017 IDEA FASoC/Chip Tapeouts/GF12 Testchip 2021/Submissions” at the following URL https://drive.google.com/drive/u/0/folders/14O6i6m5wXRW5sz0uL4YvIbn0DoQHa_E1. The final files which need to be uploaded are shown below



- Copy **Waiver** from other drive submission
- Copy **drc.summary** as“./results/calibre/drc/drc.summary”
- Copy **drc.results** as “./results/calibre/drc/drc.results”
- Copy **DFM_Reports** as “./results/calibre/dfm/DFM_rundir_.../SIGN-OFF/Reports” folder (change name)
- Copy **fasoc_testchip.top.gds.gz** as “./results/calibre/fasoc_testchip.top.gds.gz”
- Make md5sum of top.gds.gz using “md5sum fasoc_testchip.top.gds.gz > fasoc_testchip.top.gds.gz.md5sum”
    - Copy **fasoc_testchip.top.gds.gz.md5sum** to drive as well

If you are doing this all for the first time, please have someone walk through it with you to verify everything is correct.

Okay, now you are really done! In the future additional sections on common problems and solutions may be added, so if you run into anything you want to add, let me know.

Appendix

Old README/Tutorial by Kyumin. Still useful to look at to get a quick glance at overall steps to complete.

```
steps to follow

-prepare
1. design name in include.mk
2. src/
3. scripts/


-synthesis
1. make synth
2. check results/dc/design_name.mapped.v

-APR
"stage": init, place, cts, postcts_hold, route, postroute, signoff
1. make "stage"
2. make debug_"stage"
        - gui will pop up, check the placements/routings

*scripts:
        1. always_source.tcl: used in every step. (ex: core_width, core_height)
        2. floorplan.tcl: used in init
        3. pre_"stage".tcl: used right before "stage"
        4. post_"stage".tcl: used right after "stage"

-calibre
1. make lvs
2. (make drc)

-custom pex
1. use the python code



* debugging: check logs/*/"stage".log
```

I also have some zoom help sessions saved if you are interested in using those for additional assistance.