Schematic Generators

This module covers the basics of writing a schematic generator.

Schematic Generation Flow

The schematic generator design flow is slightly different than the layout generator design flow. As described in Module 1, BAG copies an existing schematic template in Virtuoso and perform modifications on it in order to generate human-readable schematic instances. The schematic generation flow follows the steps below:

1. Create schematic in Virtuoso.
2. Import schematic information from Virtuoso to Python.
3. Implement schematic generator in Python.
4. Use BAG to create new instances of the schematic.

CS Amplifier Schematic Generator Example

Lets walk through the common-source amplifier schematic generator example, reproduced below:

```
yaml_file = pkg_resources.resource_filename(__name__, os.path.join('ne
tlist_info', 'amp_cs.yaml'))

class demo_templates__amp_cs(Module):

    def __init__(self, bag_config, parent=None, prj=None, **kwargs):
        Module.__init__(self, bag_config, yaml_file, parent=parent, pr
j=prj, **kwargs)

    @classmethod
    def get_params_info(cls):
        return dict(
            lch='channel length in meters.',
            w_dict='Dictionary of transistor widths.',
            intent_dict='Dictionary of transistor threshold flavors.',
            fg_dict='Dictionary of transistor number of fingers.',
            dum_info='Dummy information data structure',
        )

    def design(self, lch, w_dict, intent_dict, fg_dict, dum_info):
        # populate self.parameters dictionary
        wp = w_dict['load']
        wn = w_dict['amp']
        intentp = intent_dict['load']
        intentn = intent_dict['amp']

        fg_amp = fg_dict['amp']
        fg_load = fg_dict['load']

        # set transistor parameters
        self.instances['XP'].design(w=wp, l=lch, intent=intentp, nf=fg
_load)
        self.instances['XN'].design(w=wn, l=lch, intent=intentn, nf=fg
_amp)

        # handle dummy transistors
        self.design_dummy_transistors(dum_info, 'XDUM', 'VDD', 'VSS')
```

Constructor

Consider the section below:

```
yaml_file = pkg_resources.resource_filename(__name__, os.path.join('ne
tlist_info', 'amp_cs.yaml'))

class demo_templates__amp_cs(Module):

    def __init__(self, bag_config, parent=None, prj=None, **kwargs):
        Module.__init__(self, bag_config, yaml_file, parent=parent, pr
j=prj, **kwargs)
```

First of all, notice the `yaml_file` variable. This variable stores the path to a netlist file that describes instances and connections in the schematic template. This netlist file is generated by BAG when schematic templates are imported from Virtuoso to BAG. This implies that everytime you modify the schematic template, you should re-import the design to BAG in order to update this file.

Then, notice that this class is a subclass of `Module`. Similar to `AnalogBase`, `Module` is the base class of all schematic generators, and it provides many useful functions to modify the schematic template.

Parameters information

Next we have the following class method declaration:

```
@classmethod
def get_params_info(cls):
    return dict(
        lch='channel length in meters.',
        w_dict='Dictionary of transistor widths.',
        intent_dict='Dictionary of transistor threshold flavors.',
        fg_dict='Dictionary of transistor number of fingers.',
        dum_info='Dummy information data structure',
    )
```

This method serves the same purpose as the method with the same name in layout generators.

The Design Method

```
def design(self, lch, w_dict, intent_dict, fg_dict, dum_info):
```

Next we have the `design()` method. Similar to `draw_layout()`, this is where all schematic modification happens. Note that all schematic parameters should be listed as arguments of the `design()` method.

Setting Transistor Parameters

The first several lines of `design()` are quite self-explanatory. Then we have the following:

```
# set transistor parameters
self.instances['XP'].design(w=wp, l=lch, intent=intentp, nf=fg_load)
self.instances['XN'].design(w=wn, l=lch, intent=intentn, nf=fg_amp)
```

All instances in the schematic template will be present in the `self.instances` dictionary, which maps instance names to the corresponding `SchInstance` objects (This is why you should always choose meaningful names for instances in your schematic template). You can modify these instances by called their `design()` method. For `BAG_prim` transistors, their `design()` method takes parameters `w`, `l`, `intent`, and `nf`, which stands for width, length, transistor threshold, and number of fingers, respectively.

Dummy transistors

In the next line we have:

```
# handle dummy transistors
self.design_dummy_transistors(dum_info, 'XDUM', 'VDD', 'VSS')
```

Recall that `dum_info` is the dummy transistor data struct computed by AnalogBase. The `design_dummy_transistors()` method will automatically help you create the corresponding dummy transistors in the schematic, by copying and modifying a transistor instance in the schematic template. For example, here it'll copy and modfy the "XDUM" transistor instance, and use "VDD" as the power supply name, "VSS" as the ground supply name.

SF Schematic Exercise

Now let's try to create a source-follower schematic generator. Before we start writing generator code, we need to create a schematic template for the source-follower:

1. Open the cellview `demo_templates` / `amp_sf` in Virtuoso.
2. Instantiate non-dummy transistors from the `BAG_prim` library, and create proper connections. Use `nmos4_standard` and `pmos4_standard` as the transistors. Leave the dummy transistor alone.
    - Name the amplifying transistor `XAMP` and the bias transistor `XBIAS`.
3. If you're stuck or you want to check your answer, see the schematic for `demo_templates` / `amp_sf_soln`.
4. After completing the schematic, evaluate the following cell, which will update the netlist associated with `amp_sf`.

In [1]:
```
import bag

# obtain BagProject instance
local_dict = locals()
if 'bprj' in local_dict:
    print('using existing BagProject')
    bprj = local_dict['bprj']
else:
    print('creating BagProject')
    bprj = bag.BagProject()

print('importing netlist from virtuoso')
bprj.import_design_library('demo_templates')
print('netlist import done')
```

```
creating BagProject
importing netlist from virtuoso
netlist import done
```

Implement Schematic Generator

Now that you finished the schematic template and imported netlist information into BAG, fill in the missing parts of the following schematic generator. After you finished, evaluate the cell below to run the source-follower amplifer through the flow. If everything works properly, LVS should pass, and you should see DC/AC/Transient simulation plots of the source-follower. If you need to change the schematic template in Virtuoso, remember to re-evaluate the cell above to regenerate the netlist file.

In [2]:
```
%matplotlib inline

import os

from bag.design import Module


# noinspection PyPep8Naming
class demo_templates__amp_sf(Module):
    """Module for library demo_templates cell amp_sf.

    Fill in high level description here.
    """

    def __init__(self, bag_config, parent=None, prj=None, **kwargs):
        # hard coded netlist flie path to get jupyter notebook working.
        yaml_file = os.path.join(os.environ['BAG_WORK_DIR'], 'BAG_XBase_demo'
                                 'BagModules', 'demo_templates', 'netlist_inf

        Module.__init__(self, bag_config, yaml_file, parent=parent, prj=prj,

    @classmethod
    def get_params_info(cls):
        return dict(
            lch='channel length in meters.',
            w_dict='Dictionary of transistor widths.',
            intent_dict='Dictionary of transistor threshold flavors.',
            fg_dict='Dictionary of transistor number of fingers.',
            dum_info='Dummy information data structure',
        )

    def design(self, lch, w_dict, intent_dict, fg_dict, dum_info):
        w_amp = w_dict['amp']
        w_bias = w_dict['bias']
        intent_amp = intent_dict['amp']
        intent_bias = intent_dict['bias']
        fg_amp = fg_dict['amp']
        fg_bias = fg_dict['bias']

        # TODO: design XAMP and XBIAS transistors
        # related code from amp_cs schematic generator are copied below
        # for reference
        # self.instances['XP'].design(w=wp, l=lch, intent=intentp, nf=fg_load
        # self.instances['XN'].design(w=wn, l=lch, intent=intentn, nf=fg_amp)

        # handle dummy transistors
        self.design_dummy_transistors(dum_info, 'XDUM', 'VDD', 'VSS')


import os

# import bag package
import bag
from bag.io import read_yaml

# import BAG demo Python modules
import xbase_demo.core as demo_core
from xbase_demo.demo_layout.core import AmpSFSoln

# load circuit specifications from file
spec_fname = os.path.join(os.environ['BAG_WORK_DIR'], 'specs_demo/demo.yaml')
top_specs = read_yaml(spec_fname)
```