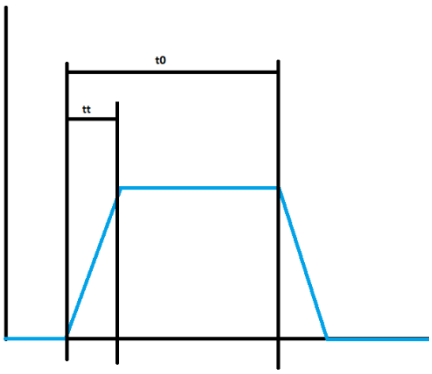
ECE 410/813 Stimulus File Generator

Step 1: Python interpreter – All that is required to run the script is a simple Python interpreter, no additional library set-up is needed. For guaranteed results accessible from any computer, www.online-python.com/ works well. A quick search for "online python compiler" will give a good lineup of alternatives.

Step 2: Running the script – Copy the contents of StimulusGenerator.py into your Python interpreter. For your first time running, give the script a quick run as-is to make sure it runs correctly in your interpreter before making changes

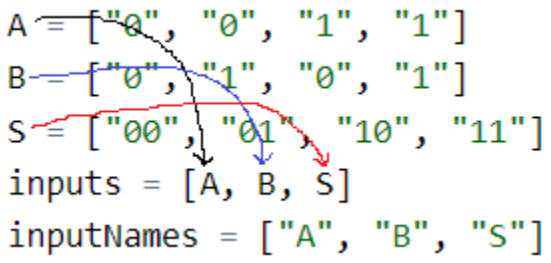
Step 3: Set your timing – The generator creates stimulus files using piecewise-linear curves that have a constant rise time and "clock" time in between switches. The variable t0 controls the time between switches in nanoseconds and tt (transition time) controls the input rise and fall time in picoseconds.

```
t0 = 20 #nanoseconds
tt = 50 #picoseconds
```



Step 4: Enter variable list and names – Setting up your inputs requires several inputs of the script to line up. The variables where you will eventually enter your data points must all be listed in the master list "inputs". The "inputNames" list defines the pin names that will be entered into stimulus file and must therefore correspond to the input pin names in your schematic. These names will be applied sequentially to items in the "inputs" list, so make sure the order of these two variables line up.

*Multi-bit variables will be automatically numbered starting from zero (i.e. S1, S0), so only one entry in “inputs” is required.



Step 5: Enter data – With your variables defined, you can now enter your data that the stimulus file will step through as stings of 1s and 0s. Here are some key points

- Multi-bit data is listed most-significant to least- significant bit (i.e. S1, S0 left to right)
- Each variable does not need to have the same number of entries, variables will hold their last-set value even if other variables continue switching.
- There is no down-side to being explicit about variables that don’t change, the script detects non-switching cycles for variables and ignores them.
- Python lists can be multiplied by constants or added together to easily repeat sequences or keep equal visual spacing (see sample1.py and sample2.py, respectively)

From sample 1, example of using repeating terms

```
A = ["001"]*4 + ["110"]*4
B = ["00", "01", "10", "11"] * 2
S = ["1"]
```

From sample 2, full data path stimulus

```
#       blank          write stage    write push     read stage     read push      write stage    write push
md    = ["00000000"]   +["00001111"]  +["00001111"]  +["00000000"]  +["00000000"]  +["00000000"]  +["00000000"]
rfs   = ["0"]          +["1"]         +["1"]         +["0"]         +["0"]         +["0"]         +["0"]
rfen  = ["0"]          +["0"]         +["1"]         +["0"]         +["1"]         +["0"]         +["1"]
rw    = ["0"]          +["1"]         +["1"]         +["0"]         +["0"]         +["1"]         +["1"]
ada   = ["000"]        +["000"]       +["000"]       +["000"]       +["000"]       +["010"]       +["010"]
adb   = ["000"]        +["000"]       +["000"]       +["001"]       +["001"]       +["000"]       +["000"]
f     = ["000000000"]  +["000000000"] +["000000000"] +["000001000"] +["000001000"] +["000000000"] +["000000000"]
clk1  = ["1"]          +["0"]         +["1"]         +["0"]         +["1"]         +["0"]         +["0"]
clk2  = ["1"]          +["1"]         +["0"]         +["1"]         +["0"]         +["1"]         +["1"]
```

Step 6: Define output – There is only one variable for outputs, since all this information does is apply a 3fF output capacitance to that node so adding more is as simple as copying those lines.

```
outputName = "Y"
outputBits = 1
```

Step 7: Run code and transfer outputs – After running the script with your modified inputs, the output should be the text of a complete stimulus file to copy into a text file on the Linux server for you to select in ADE.

* Check that the end of the file is correct, some interpreters print additional lines after the script finishes, and double check your final text file includes a line break after the last line.

Step 8: Save your generator – Once you have a working stimulus, there may be additional input combinations you want to test during later sessions. If you expect to be working with the same cell, it might be wise to copy the updated generator file, with all your input configurations, as a copy to return to later.