Aalto University
School of Electrical
Engineering

TheSyDeKick tutorial

Marko Kosunen

Department of Micro and Nanosciences
Aalto University, School of Electrical Engineering
marko.kosunen@aalto.fi

August 3, 2022

Outline

TheSyDeKick project structure

Testing the environment

Creating a new Entity

Simplifying the model to the bone

Documentation with Doctrings

Getting production ready

Congratulations, You are DONE!

Prerequisites

- Project template is available at *https://github.com/TheSystemDevelopmentKit/thesdk_template*.
- Alternatively you should have access to your project's TheSyDeKick repository at *https://bubba.ecdl.hut-fi:81*
- You MUST have ssh keys set up for GitHub and ECD GitLab at bubba.

TheSyDeKick project structure

Directory structures of TheSyDeKick project

- All TheSyDeKick projects look the same

```
TheSyDeKick_project
    init_submodules.sh
    configure
    sourceme.csh
    pip3userinstall.sh
    Entities                    <- All design modules are ''entities''
        thesdk                  <- The SyDeKick core entity
        rtl                     <- rtl entity for rtl simulations
        eldo                    <- eldo entity for eldo simulations
        thesdk_helpers
            shell
                initentity.sh <- Shell script for creating new entities
        inv_sim
        inverter                <- Example entity inverter. All entities look the same
            init_submodules.sh
            configure
            doc
            sv
            eldo
            vhdl
            simulations
                rtlsim
            inverter
                __init__.py   <- Python description of the entity
                controller.py <- Additional entity related Python
```

TheSyDeKick project structure

- All TheSyDeKick projects look the same
- TheSydeKick entities are git submodules initiated in the *init_submodules.sh*
- TheSydeKick entitities are transferable to any TheSyDeKick project.
- TheSydeKick entitities are transferable to any TheSyDeKick project.
- TheSydeKick entitities do not run stand-alone. They need the project.
- Obey the structure. It is not yours to change.
- New entities are initiated with *thesdk_helpers/shell/initentity.sh*

Testing the environment

Testing the environment

- To test TheSyDeKick installation, do the following

```
cd TheSyDeKick_project
./init_submodules.sh
./configure
source sourceme.csh
pip3userinstall.sh
```

- The, **check the Python versions from Thesdk.config** file. Version 3.6 is OK for ECD computing machines.
- Thesdk.config is created and will be overwritten by the configure script. Usually no need to re-run it.

Testing the environment

- Then we test the simulation execution

```
cd Entities/inverter
./configure
python3 inverter/__init__.py
```

- Simulation of an inverter modeled in Python, verilog, vhdl spectre and eldo is executed.
- Press *Return* to close the figures
- This is the elementary way of running simulations
- The ”production way” is

```
./configure && make sim
```

- Try it. If it works, you are good to go for the next step.

Creating a new Entity

Creating a new Entity

- All the Entities are eventually git submodules.
- Go through the following steps and try to think what happens in in term of version control
- The <my_entity> refers to the entity you are creating. *it should be replaced with your entity name*
- By default, the remote points to GitHub, and you do not have push permissions there.

```
cd entities
./thesdk_helpers/shell/initentity -h
./thesdk_helpers/shell/initentity <my_entity>
cd Entities/<my_entity>
#This is just to test the operation
python3 <my_entity>/__init__.py
git remote -v
git remote remove origin
git remote add origin \
    <URL of your TheSyDeKickgroup/<my_entity>.git
git push --set-upstream origin master
```

Converting the new entity to submodule

- Go through the following steps and try to think what happens in in term of git submodules

```
cd TheSyDeKick_project
rm -rf Entities/<my_entity>
git remote remove origin
git remote submodule add  \
    <URL of your TheSyDeKickgroup/<my_entity>.git Entities/
```

- Edit the ./init_submodules.sh script to contain *Entities/<my_entity>*. Then:

```
./init_submosules.sh
```

Working with the submodules

- If you want to edit a submodule *within the master project* this is how it goes

```
cd Entities/<my_entity>
git chekout master # Or your favorite branch
# Do your edits

git add -i '#Add and select the files you want to commit
git commit # You may use -m, but follow the good practices
           # https://chris.beams.io/posts/git-commit/
git push
# Now comes the trick
cd ../
git add <my_entity>
git commit -m''Update <my_entity> submodule''
git push
# To test if everything went as you really wanted
# ../init_submodules.sh
```

Simplifying the model to the bone

The most simple TheSyDeKick model

- The template (<my_entity>) contains features that support python,eldo and rtl simulations.
- Next, we will remove all the parts from the model, and leave only the python model in place.

The target code

The target code

- Edit the *Docstring*

```
1  """
2  =========
3  My Entity
4  =========
5
6  My Entity model template The System Development Kit
7  Used as a template for all TheSyDeKick Entities.
8
9  Current docstring documentation style is Numpy
10 https://numpydoc.readthedocs.io/en/latest/format.html
11
12 This text here is to remind you that documentation is important.
13 However, youu may find it out the even the documentation of this
14 entity may be outdated and incomplete. Regardless of that, every day
15 and in every way we are getting better and better :).
16
17 Initially written by Marko Kosunen, marko.kosunen@aalto.fi, 2017.
18
19 """
20
21 import os
```

The target code

- Edit the *package imports*

```
22 import sys
23 if not (os.path.abspath('../../thesdk') in sys.path):
24     sys.path.append(os.path.abspath('../../thesdk'))
25
26 from thesdk import *
27
28 import numpy as np
29
```

The target code

- Edit the *Class definition*

```
30 class myentity(thesdk):
31     @property
32     def _classfile(self):
33         return os.path.dirname(os.path.realpath(__file__)) + "/"+__name__
34
35     def __init__(self,*arg):
36         self.print_log(type='I', msg='Inititalizing %s' %(__name__))
37         self.proplist = [ 'Rs' ];     # Properties that can be propagated from parent
38         self.Rs =  100e6;             # Sampling frequency
39         self.IOS=Bundle()             # Pointer for input data
40         self.IOS.Members['A']=IO()    # Pointer for input data
41         self.IOS.Members['Z']= IO()
42         self.model='py';              # Can be set externally, but is not propagated
43         self.par= False               # By default, no parallel processing
44         self.queue= []                # By default, no parallel processing
45
46         if len(arg)>=1:
47             parent=arg[0]
48             self.copy_propval(parent,self.proplist)
49             self.parent =parent;
50
51         self.init()
52
53     def init(self):
54         pass #Currently nohing to add
55
56     def main(self):
57         '''Guideline. Isolate python processing to main method.
58
59         To isolate the interna processing from IO connection assigments,
60         The procedure to follow is
61         1) Assign input data from input to local variable
62         2) Do the processing
63         3) Assign local variable to output
64
65         '''
66         inval=self.IOS.Members['A'].Data
67         out=inval
68         if self.par:
69             self.queue.put(out)
70         self.IOS.Members['Z'].Data=out
71
72     def run(self,*arg):
73         '''Guideline: Define model depencies of executions in `run` method.
74
75         '''
76         if len(arg)>0:
77             self.par=True       #flag for parallel processing
78             self.queue=arg[0]   #multiprocessing.queue as the first argument
79         if self.model=='py':
80             self.main()
81
```

The target code

- Edit the *Main script*

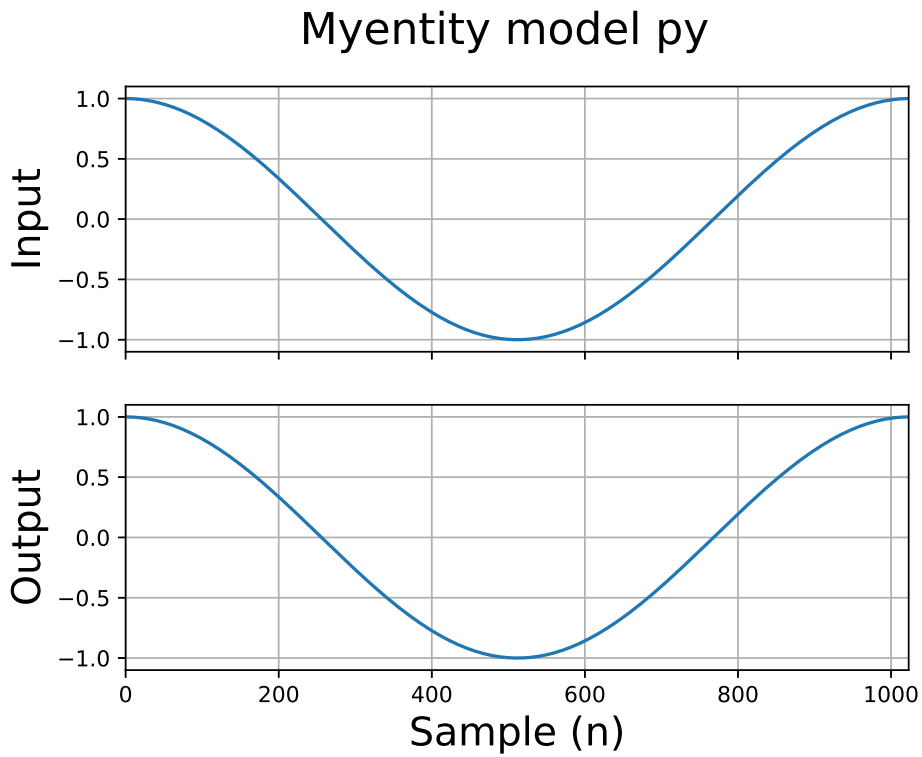
```
82  if __name__=="__main__":
83      import matplotlib.pyplot as plt
84      from  myentity import *
85      from  myentity.controller import controller as myentity_controller
86      import pdb
87      import math
88      length=1024
89      rs=100e6
90      indata=np.cos(2*math.pi/length*np.arange(length)).reshape(-1,1)
91
92      models=[ 'py']
93      duts=[]
94      for model in models:
95          d=myentity()
96          duts.append(d)
97          d.model=model
98          d.Rs=rs
99          d.IOS.Members['A'].Data=indata
100         d.init()
101         d.run()
102
103     for k in range(len(duts)):
104         hfont = {'fontname':'Sans'}
105         figure,axes=plt.subplots(2,1,sharex=True)
106         x = np.arange(length).reshape(-1,1)
107         axes[0].plot(x,indata)
108         axes[0].set_ylim(-1.1, 1.1);
109         axes[0].set_xlim((np.amin(x), np.amax(x)));
110         axes[0].set_ylabel('Input', **hfont,fontsize=18);
111         axes[0].grid(True)
112         axes[1].plot(x, duts[k].IOS.Members['Z'].Data)
113         axes[1].set_ylim(-1.1, 1.1);
114         axes[1].set_xlim((np.amin(x), np.amax(x)));
115         axes[1].set_ylabel('Output', **hfont,fontsize=18);
116         axes[1].set_xlabel('Sample (n)', **hfont,fontsize=18);
117         axes[1].grid(True)
118         titlestr = "Myentity model %s" %(duts[k].model)
119         plt.suptitle(titlestr,fontsize=20);
120         plt.grid(True);
121         printstr="./inv_%s.eps" %(duts[k].model)
122         plt.show(block=False);
123         figure.savefig(printstr, format='eps', dpi=300);
124     input()
125
```

The target code

- Now you are ready to run you model

```
cd Entities/<my_entity>
#This is just to test operation
python3 <my_entity>/__init__.py
```

- The result should look like this:

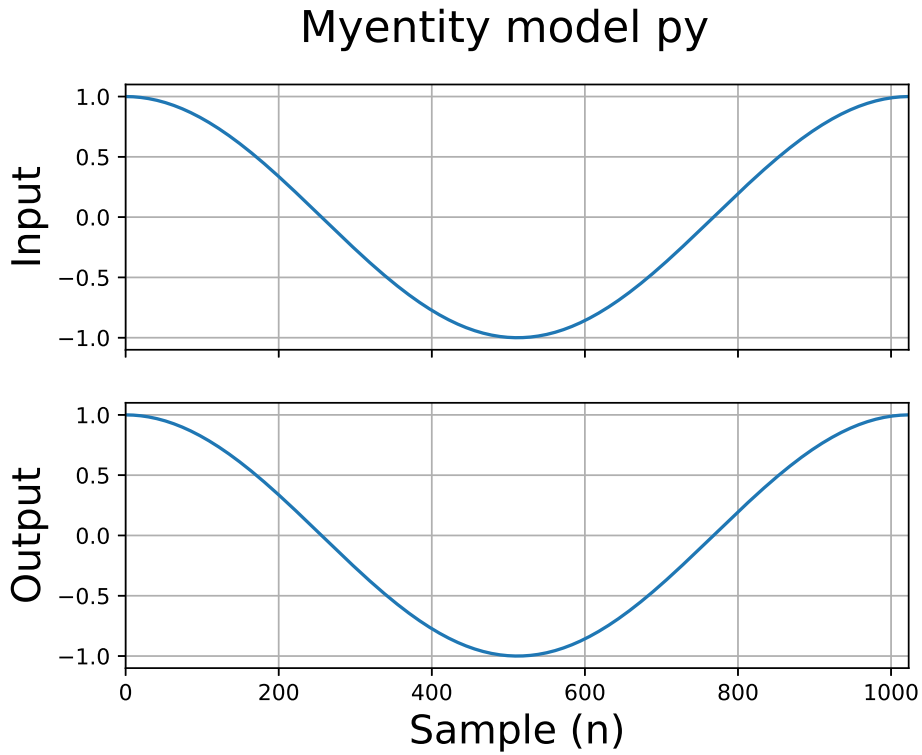


Running with make

- You can now try to run the test with the “Production method”

```
#cd Entities/<my_entity>
./configure
make sim
```

- The result should be the same:



Documentation with Doctrings

Building the documentation

- TheSyDeKick takes also care for you basic documentation needs
- We are using Python Docstrings for that. You may do a web search to figure out what it means.
- Create the documentation or your module with:

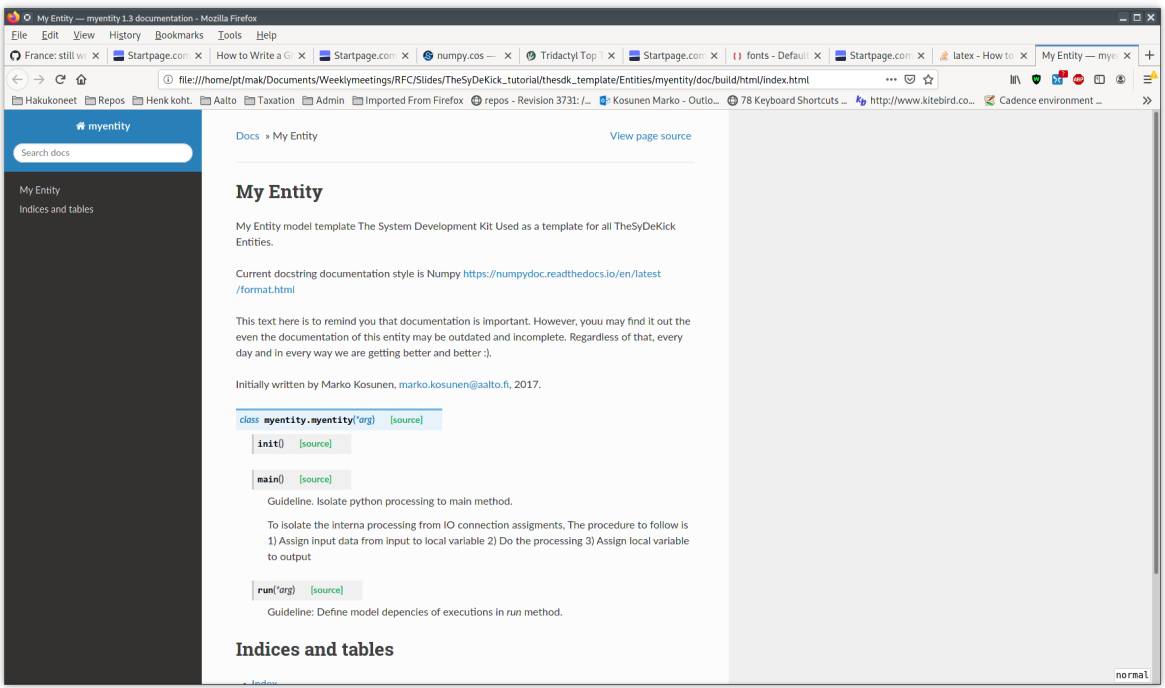
```
#cd Entities/<my_entity>
./configure
make doc
```

Reading the documentation

- You may read the documentation with

```
#cd Entities/<my_entity>
firefox ./doc/build/html/index.html
```

- Compare the documentation to your source code. You may already guess how it is created.



Getting production ready

Production version

- To minimize the need for documentation, TheSydeKick follows the following principles
  - *./init_submodules.sh* gets the submodules
  - *configure* does the configuration and creates the Makefile
  - *make* does the actual work with some functional defaults, and creates the documentation.
- You should be now ready to build you module as it is in production.

```
#cd Entities/<my_entity>
configure && make
```

- Press *Return* close the figure. (NOTE: Normally, you should NOT have the line 'input()' and the end of the self test. You results should be files.)
- This runs the simulation and generates the documentation.
- You may study the structure of *configure*
- You are now ready to release your module:

```
git add -i  #Select the files you have edited
git commit '# Give a nice and clean commit message
git push
```

Working with the submodules, again

- As you are working *within the master project* remember to update it

```
cd Entities/
git add <my_entity>
git commit –m‘‘Update <my_entity> submodule’’
git push
# To test if everything went as you really wanted
# ../init_submodules.sh
```

Congratulations, You are DONE!