TheSyDeKick tutorial

Marko Kosunen

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

September 21, 2023

Outline

- TheSyDeKick project structure
- Testing the environment
- Creating a new Entity
- Simplifying the model to the bone
- The target code
- Documentation with Doctrings
- Getting production ready
- Congratulations, You are DONE!

Prerequisites

- Project template is available at *https://github.com/TheSystemDevelopmentKit/thesdk_template*.
- If you have access to any valid, up-to-date clone of the template, you can use that as well.

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
        spice                 <- spice entity for analog simulations
        thesdk_helpers
            shell
                initentity.sh <- Shell script for creating new entities
        inverter_tests
        inverter_testbench
        inverter              <- Example entity inverter. All entities look the same
            init_submodules.sh
            configure
            doc
            sv
            spice
            vhdl
            simulations      <- Temporary directory for simulation results
                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 entities are transferable to any TheSyDeKick project.
- TheSydeKick entities 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
```

- Then **check the Python versions from Thesdk.config.** Release v1.10 is tested with Python 3.10.
- 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 (icarus), vhdl (ghdl) and ngspice is executed.
- Press *Return* to close the figures
- This is the elementary way of running simulations, i.e.you provide the scriptfile to python shell.
- 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 submodule add \
    <URL of your TheSyDeKickgroup/<my_entity>.git Entitie
```

- 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 practic
           # 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 simplest 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*

```
"""
=========
My Entity
=========

My Entity model template The System Development Kit
Used as a template for all TheSyDeKick Entities.

Current docstring documentation style is Numpy
https://numpydoc.readthedocs.io/en/latest/format.html

This text here is to remind you that documentation is important.
However, youu may find it out the even the documentation of this
entity may be outdated and incomplete. Regardless of that, every day
and in every way we are getting better and better :).

Initially written by Marko Kosunen, marko.kosunen@aalto.fi, 2017.

"""

import os
```

The target code

- Edit the *package imports*

```python
import sys
if not (os.path.abspath('../../thesdk') in sys.path):
    sys.path.append(os.path.abspath('../../thesdk'))

from thesdk import *

import numpy as np

```

The target code

- Edit the *Class definition*

```
class myentity(thesdk):

    def __init__(self,*arg):
        self.print_log(type='I', msg='Inititalizing %s' %(__name__))
        self.proplist = [ 'Rs' ];    # Properties that can be propagated from parent
        self.Rs =  100e6;            # Sampling frequency
        self.IOS=Bundle()            # Pointer for input data
        self.IOS.Members['A']=IO()   # Pointer for input data
        self.IOS.Members['Z']= IO()
        self.model='py';             # Can be set externally, but is not propagated
        self.par= False              # By default, no parallel processing
        self.queue= []               # By default, no parallel processing

        if len(arg)>=1:
            parent=arg[0]
            self.copy_propval(parent,self.proplist)
            self.parent =parent;

        self.init()

    def init(self):
        pass #Currently nohing to add

    def main(self):
        '''Guideline. Isolate python processing to main method.

        To isolate the interna processing from IO connection assigments,
        The procedure to follow is
        1) Assign input data from input to local variable
        2) Do the processing
        3) Assign local variable to output

        '''
        inval=self.IOS.Members['A'].Data
        out=inval
        if self.par:
            self.queue.put(out)
        self.IOS.Members['Z'].Data=out

    def run(self,*arg):
        '''Guideline: Define model depencies of executions in `run` method.

        '''
        if len(arg)>0:
            self.par=True      #flag for parallel processing
            self.queue=arg[0]  #multiprocessing.queue as the first argument
        if self.model=='py':
            self.main()

if __name__=="__main__":
    import argparse
    import matplotlib.pyplot as plt
```

The target code

- Edit the *Main script*

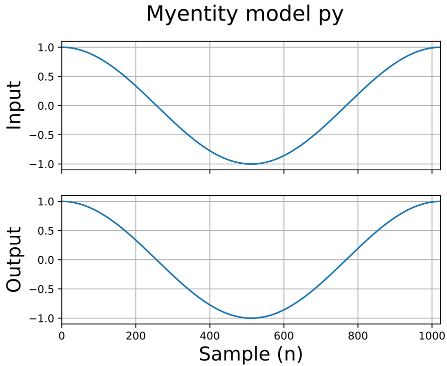
```
from  myentity import *
from  myentity.controller import controller as myentity_controller
import pdb
import math
# Implement argument parser
parser = argparse.ArgumentParser(description='Parse␣selectors')
parser.add_argument('--show', dest='show', type=bool, nargs='?', const = True,
        default=False,help='Show␣figures␣on␣screen')
args=parser.parse_args()

length=1024
rs=100e6
indata=np.cos(2*math.pi/length*np.arange(length)).reshape(-1,1)

models=[ 'py']
duts=[]
plotters=[]
for model in models:
    d=myentity()
    duts.append(d)
    d.model=model
    d.Rs=rs
    d.IOS.Members['A'].Data=indata
    d.init()
    d.run()

for k in range(len(duts)):
    hfont = {'fontname':'Sans'}
    figure,axes=plt.subplots(2,1,sharex=True)
    x = np.arange(length).reshape(-1,1)
    axes[0].plot(x,indata)
    axes[0].set_ylim(-1.1, 1.1);
    axes[0].set_xlim((np.amin(x), np.amax(x)));
    axes[0].set_ylabel('Input', **hfont,fontsize=18);
    axes[0].grid(True)
    axes[1].plot(x, duts[k].IOS.Members['Z'].Data)
    axes[1].set_ylim(-1.1, 1.1);
    axes[1].set_xlim((np.amin(x), np.amax(x)));
    axes[1].set_ylabel('Output', **hfont,fontsize=18);
    axes[1].set_xlabel('Sample␣(n)', **hfont,fontsize=18);
    axes[1].grid(True)
    titlestr = "Myentity␣model␣%s" %(duts[k].model)
    plt.suptitle(titlestr,fontsize=20);
    plt.grid(True);
    printstr="./inv_%s.eps" %(duts[k].model)
```

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:

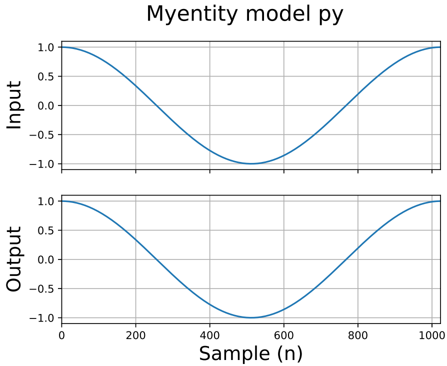


The target code

- 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 ysing Python Docstrings for that. You may do a web search to figure out what it means.
- Create the documentation or your module with:

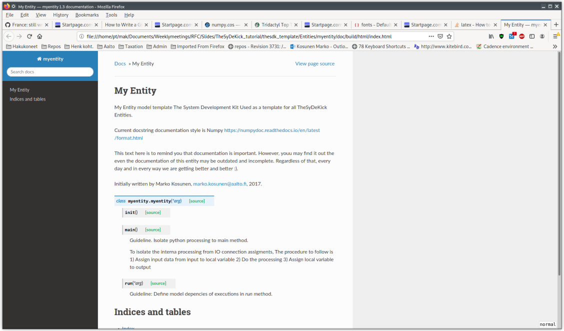
```
#cd Entities/<my_entity>
./configure
make doc
```

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.



My Entity — myentity 1.3 documentation - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///home/pt/mak/Documents/Weeklymeetings/RFC/Slides/TheSyDeKick_tutorial/thesdk_template/Entities/myentity/doc/build/html/index.html
myentity
Search docs
My Entity
Indices and tables
Docs » My Entity
View page source
normal

My Entity

My Entity model template The System Development Kit Used as a template for all TheSyDeKick Entities.

Current docstring documentation style is Numpy https://numpydoc.readthedocs.io/en/latest/format.html

This text here is to remind you that documentation is important. However, youu may find it out the even the documentation of this entity may be outdated and incomplete. Regardless of that, every day and in every way we are getting better and better :).

Initially written by Marko Kosunen, marko.kosunen@aalto.fi, 2017.

class **myentity.myentity**(**arg*) [source]

init() [source]

main() [source]

Guideline. Isolate python processing to main method.

To isolate the interna processing from IO connection assigments, The procedure to follow is 1) Assign input data from input to local variable 2) Do the processing 3) Assign local variable to output

run(**arg*) [source]

Guideline: Define model depencies of executions in *run* method.

Indices and tables

- Index

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 are now ready to build you module for ’production’ use.

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

- Press *Return* close the figure.
- 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 workin *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
```

Next Steps

- Once you understand how to one entity is constructed,please familiarize yoursel to *inverter_testbenchses* and *inverter_tests* enetities.
- By studying them yous should learn how to connect entities together, and how to construct simulations for different simulators.

Congratulations, You are DONE!