Running Jasper tool flow How To

1) OS Required/suggested: Ubuntu 14.04 LTS.

2) This how to assumes that Matlab has already been installed - refer to the Matlab install how to document:
   https://github.com/ska-sa/skarab_docs (master branch)

3) If you are using Vivado as your compile backend then this how to assumes that Vivado has already been installed - refer to the Vivado install how to document:
   https://github.com/ska-sa/skarab_docs (master branch)

4) If you are using ISE as your compile backend then this how to assumes that Xilinx ISE 14.7 has already been installed - refer to the ISE install how to document:
   https://github.com/ska-sa/skarab_docs (master branch)
   NB: SKARAB only needs Vivado, but you may wish to support ROACH platforms

5) If you haven't installed Python, then open a terminal <ctrl +alt + T>. In terminal type:
   "sudo apt-get install python".
   This will install the python package.

6) If you haven't installed Python pip, then in the terminal type:
   "sudo apt-get install python-pip". This will install the python pip package (package manager to install/manage python software packages).

7) If you haven't installed Python yaml, then in the terminal type:
   "sudo pip install pyyaml". This will install the yaml python module.

8) Using the terminal, type the following:
   "*git clone https://github.com/ska-sa/mlib_devel.git (master branch)"*.
   This will clone the mlib_devel repository onto your local machine.

9) Using the terminal, type the following:
   "git status" in the "mlib_devel" folder.
   This should indicate that the master branch has been selected.

10) There are two ways of working with Jasper. You can do this initially with the matlab GUI to compile the front end and then handle the middleware and backend generation using Python or you can run everything in Python. The former stage is more for design and debugging (steps 12-21) and the later stage (steps 22-25) is for the final tested and working design. This How To will cover both methods.
11) It is now time to setup the Xilinx, Matlab and work paths to suit your directory setup. If you don't do this then the below steps will never work. Using the terminal, go to the "mlib_devel" directory and add the correct path to following parameters in "ise_config.sh" and "vivado_config.sh" (if you are not using ISE or Vivado then there is no need to edit that respective script): "XILINX_PATH" (path to your Xilinx script), "MATLAB_PATH" (path to your Matlab script), "MLIB_DEVEL_PATH" (path to your "mlib_devel" folder), "PLATFORM" (this is set to "lin64" for a 64 bit Linux system). Remember to save the file. NB: Please do not commit or push these file changes to the git repo, as this file will constantly change from one user to the next.
12) **Matlab/Python method:** Using the terminal, type the following under "mlib_devel":
   "./startsg" if using Xilinx Vivado to compile the backend or "*./startsg_ise"* if using Xilinx ISE to compile the backend.

It is currently not possible to compile for Altera or Lattice FPGAs. This script will source all the relevant Matlab and Xilinx paths, run matlab and start the system generator. Wait until the Matlab GUI has opened and Matlab is ready.

13) **Matlab/Python method:** In the Matlab command window, type the following:

"simulink".

This will start simulink. Wait until the Simulink window has opened.

14) **Matlab/Python method:** In the Simulink Library Browser, click on the "open model or library" icon in the tab and select where your desired simulink file is (*.slx). There are some test files under "jasper_library/test_models". I use "test_snap.slx" for this How To. Once the file has been selected, click "Open". The "test_snap" design should open in the Simulink window.

15) **Matlab/Python method:** Click the simulink design window ("test_snap") and press the following:

"Ctrl + D".

This will update the simulink model and check for warnings or errors. Make sure there are no errors or warnings. A window should pop up if this is the case.

16) **Matlab/Python method:** In the Matlab command window terminal, type the following:

"jasper".

This will generate the yellow block peripheral file and run the system generator. Wait until the "XSG generation complete. Complete. Run 'exec_flow.py -m …." message is displayed.

17) **Matlab/Python method:** In the Matlab Command Window, cut the following text from it:

"exec_flow.py -m … --middleware --backend --software ... ".
The matlab generation process is now complete and now it is time to switch to Python.

18) **Matlab/Python method:** Open a new terminal <CTRL+ALT+T>, and source the following files from the "mlib_devel" directory:
"source ise_config.sh " if you are using ISE or
"source vivado_config.sh" if you are using Vivado
This is an important step, because the Xilinx and Matlab paths will not be specified properly and "exec_flow.py" will fail to run if this is not done.

19) **Matlab/Python method:** Using the terminal, paste the "exec_flow.py…." command that was cut earlier from Matlab, in the terminal (NB: Make sure the current directory is the "mlib_devel/jasper_library" folder):
"python *exec_flow.py -m … --middleware --backend --software".*
The script will be slightly different for ISE compiles. Remember to add the "python" in front of the "exec_flow.py…" command. Press enter. This command will execute the middleware, which calls the yellow block constructors, creates the top.v file and generates the yaml file, which contains all the parameters needed for the backend to compile. The backend reads the yaml file and builds a list of sources, constraints, generates the constraints file and the tcl file. This tcl file is used by Vivado or ISE to compile the top.v file and all other relevant source files. This generates a bit and binary file, which is used to configure the FPGA. The software reads the binary file and generates a bof and fpg file. The arguments passed to exec_flow.py will be explained in more detail below when dealing with the Python method.

20) **Matlab/Python method:** Using the terminal, wait until the design has finished compiling. Vivado compiles should indicate that there are

no timing violations. ISE compiles will display the timing report. Check the slack times for the setup and hold reports. They should not be negative. If they are then your design is not meeting timing and some changes will need to be made to your design.

21) **Matlab/Python method:** The output directories are generated where the *.slx file sits. In my case, because I was using "test_snap.slx", the directories were generated under "jasper_library/test_models/test_snap". The following directories were created under "test_snap" and should be the same as yours: "sysgen" (contains the system generator files), "outputs" (contains the bof and fpg files) and "myproj" (contains the Vivado or ISE projects files, source files, synth results and implementation results. The bin and bit files are also stored here).

22) **Python method:** Before I explain this method it is important to explain how the "exec_flow" command works and the arguments that are passed to it. The "exec_flow", which stands for "execution flow" can either run the whole flow or just parts of the flow depending on the needs of the user. It is also capable of selecting which Vivado project flow to use i.e. project mode vs non-project mode. It currently defaults to project mode - it should be pointed out that non-project mode is no longer supported in the tool flow. I have already explained the "--middleware", "--backend" and "--software" arguments in step 19) above. There is also a "--perfile" and "--frontend" argument, which is not needed in the Matlab/Python method, but is required for the Python method. The "--perfile" and "--frontend" arguments run the yellow block peripheral file generation and the system generator compile, respectively. It is identical to running "jasper" from the command window in Matlab - see Matlab/Python method above. Below is a list of the "exec_flow" arguments.
    a. "--perfile". Runs the front end peripheral file generation. If not specified, then it won't generate the peripheral file.

b. "--frontend”. This compiles the front end IP, which basically runs the system generator. If not specified, then the compile will not be run.
c. “--middleware”. This runs the tool flow middle process. If not specified, then this process will not be run.
d. “--backend”. This runs the backend compilation i.e. Xilinx Vivado or ISE. If not specified, then this process will not be run.
e. “--software”. This runs the software compilation - generates a *.bof and *.fpg file. If not specified, then this process will not be run.
f. “--be”. This specifies the type of backend to be run. This can either be “--be vivado” or “-be ise”. If this is not specified, then the default is the Vivado backend.
g. “--jobs”. The number of processor cores to run the compile with. If this is not specified, the default is 4. You need to make sure that your processor has at least 4 threads if this is to work.
h. “--nonprojectmode”. NB: This will no longer be supported, as the project mode is good enough for most tool flows, but does work for the SNAP boards. This is only for Vivado compiles. The tool flow can either be project mode (generates a project file) or non-project mode (project is run in memory). Non-project mode is beneficial if there are issues with timing closure. If not specified, the default is project mode. ISE only works using project mode.
i. “-m”. The absolute path and filename of the *.slx file (Simulink model) to compile. If not specified, the default is “/tools/mlib_devel/jasper_library/test_models/test.slx”. I would suggest always specifying this.
j. “-c”. This is the build directory. The default is the same directory as the *.slx file (Simulink model). I don’t normally specify this.

Here are some examples of how to run the command (please make sure you run the command from “/mlib_devel/jasper_library”, where “exec_flow.py” resides:

- This will run the whole process using Vivado in project mode.

  “*python exec_flow.py -m /home/aisaacson/work/git_work/mlib_devel/jasper_library/test_models/test_snap.slx --perfile --frontend --middleware --backend --software”*

- This will run the whole process using Vivado in non-project mode.

  “*python exec_flow.py -m /home/aisaacson/work/git_work/mlib_devel/jasper_library/test_models/test_snap.slx --perfile --frontend --middleware --backend --software --nonprojectmode”*

- This will run the whole process using Xilinx ISE.

  “*python exec_flow.py -m /home/aisaacson/work/git_work/mlib_devel/jasper_library/test_models/test_snap.slx --perfile --frontend --middleware --backend --software --be ise”*

- This will run the front end peripheral file generation and IP compile process using the ISE system generator.

  “*python exec_flow.py -m /home/aisaacson/work/git_work/mlib_devel/jasper_library/test_models/test_snap.slx --perfile --frontend --be ise”*

- This will run the front end peripheral file generation and IP compile process using the Vivado system generator.

"*python exec_flow.py -m /home/aisaacson/work/git_work/mlib_devel/jasper_library/test_models/test_snap.slx --perfile --frontend"*

23) **Python method:** Open a new terminal <CTRL+ALT+T>, and source the following files from the "mlib_devel" directory:
"source ise_config.sh " if you are using ISE or
"source vivado_config.sh" if you are using Vivado
This is an important step, because the Xilinx and Matlab paths will not be specified properly and "exec_flow.py" will fail to run.

24) **Python method:** Using the terminal, run the complete "exec_flow" command from the "mlib_devel/jasper_library" directory:
"python exec_flow.py -m /home/aisaacson/work/git_work/mlib_devel/jasper_library/test_models/test_snap.slx --perfile --frontend --middleware --backend --software" if you are using Vivado or
"python exec_flow.py -m /home/aisaacson/work/git_work/mlib_devel/jasper_library/test_models/test_snap.slx --perfile --frontend --middleware --backend --software --be ise" if you are using ISE

Feel free to add or remove arguments as you wish or need. The design should run through the whole tool flow generation process and when complete the ISE or Vivado compile will of completed without any errors and with a little luck there will be no timing issues. The Vivado compile will determine if timing is met or not and display this to the screen. The ISE compile will display the slack time and report for setup and hold times to the screen. The user will need to monitor the slack time variable to see whether the compile has met timing or not. If the slack time is negative then timing is not met and if the slack time is positive for both setup and hold timing then the design has met the timing requirements.

25) **Python method:** The output directories are generated where the *.slx file sits. In my case, because I was using “test_snap.slx”, the directories were generated under “jasper_library/test_models/test_snap”. The following directories were created under “test_snap” and should be the same as yours: “sysgen” (contains the system generator files), “outputs” (contains the bof and fpg files) and “myproj” (contains the Vivado or ISE projects files, source files, synth results and implementation results. The bin and bit files are also stored here). There is a slight difference to the output directories and files generated for the Vivado project mode and non-project mode. The Vivado project mode creates synthesis and implementation run folders with a project file under “myproj”. The non-project mode creates design check points, reports, bit and binary files all under the “myproj” folder. NB: It does not copy the *.v, *.vhdl source files and add them to this directory. It uses the source files where they are originally situated. This won’t be supported in the future, as previously stated.