Jasper mlib_devel directory structure explained

This is a cut and paste from my correspondence with Jack Hickish via email - thanks Jack! :). I sent the following question to him:

There are no tuts for Jasper. I know there are tuts for Casper :). Do you mind giving me a nice step by step how to in order to use the "Jasper_devel" branch? There are a lot of python files and m files. I just need to know the hierarchy of the project structure and python files.

Jack's response:

There is this --
https://casper.berkeley.edu/wiki/SNAP_Bringup#Using_the_JASPER_Toolflow_with_SNAP - and some test models at https://github.com/jack-h/mlib_devel/tree/jasper_devel/jasper_library/test_models
You should be able to follow casper tut1 if you build it from scratch with the jasper_devel branch. Once you get to compiling (casper_xps) see the above link for the jasper_specific instructions. I'd be amazed if this works first time....

The vague structure of the mlib_devel repo is:
-jasper_library/hdl_sources
  HDL source files for all user IP
-jasper_library/yellow_blocks
 python classes for each yellow block in the simulink xps blockset. These classes contain the python code which tells the tool flow how each yellow block should modify the project's top-level HDL source file and vivado project.
-jasper_library/platforms
 a yaml file specifying information about a specific hardware platform. Mostly this is used to map pythonic constraints - i.e., "connect signal my_led to the board's led[4] pin" - to hardware constraints - i.e. " my_led -> LOC XXX, my_led -> IOSTD XXX". This file also includes source files the platform requires to compile. The source files in jasper_library/hdl_sources/<platform_name>/ are automatically included. jasper_library/hdl_sources/<platform_name>/top.v is used as a starting point for HDL generation.

There's a few matlab scripts in jasper_library which turn a simulink diagram into source/configuration files that the rest of the tool flow can understand.

Then there's the entire mlib_devel/casper_library, which is all the matlab/simulink files for the casper DSP (Digital Signal Processing) libraries.
everything in mlib_devel/xps_base is obsolete. It contains the pcores, which the old casper tool set utilises. Jasper does not make use of this though.
Then there is the entire mlib_devel/xps_library, which contains all the matlab/simulink yellow block files for the casper XPS (Xilinx Platform Studio) libraries.

If you want to add your BSP in, there are two ways to go.

1) (easiest [maybe])
Just make your top level HDL file, including all interfaces (HMC, QSFP, whatever) a template file for jasper. Then in simulink you can design your user IP, and that will get instantiated within the template. You'll need to add some lightweight yellow blocks in simulink to pass signals from the simulink design to the rest of the top level code. e.g., if you wanted to send a simulink signal over 40GbE, you'd need to make a yellow block, with associated python class, to connect the simulink signals to the 40GbE controller in your BSP.

2) (jasper/casper standard)
Make a template file that includes only cores which don't interact with simulink, and don't depend on the contents of the simulink design (for snap, this is just an infrastructure module which generates sys_clk, sys_clk90, etc.). Don't include any interfaces (HMC, QSFP, etc.) in the template.
Make yellow blocks for each bit of IP, which when instantiated in a simulink design will trigger the relevant HDL insertion in the project. (See, e.g., gpio / swreg yellow blocks and their python classes).