**DMA Linux Lab**

**Overview**

This lab will provide a base design to help a designer get AXI Streaming data in and out of PL or PS DDR. A user should be able to take this design and modify it for something more relevant for their application. This example works with the movement of Memory Mapped data as well as Streaming data.

**Lab Instructions**

The following instructions refer to a zip file containing all of the source files to execute the entire lab. This zip file is labeled 182mover-6-25-2018.zip. The contents of this zip file should be extracted into a clean directory and preferably within Linux. The lab will make use of the primary Xilinx Tools:

* 2018.2 Vivado
* 2018.2 PetaLinux
* 2018.2 SDSoC

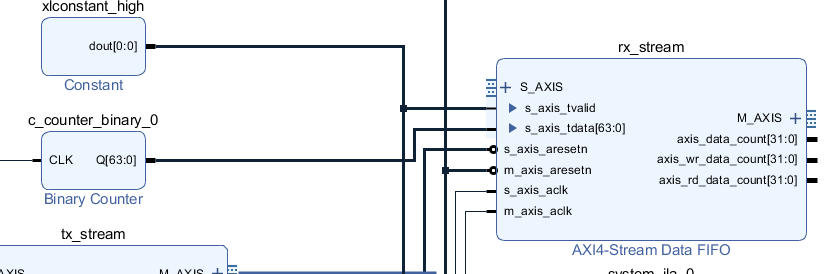
Please ensure to have access to these tools before attempting to start.

**Step 1: Vivado Project**

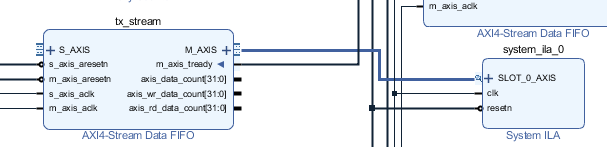
This lab is designed for a user who is already familiar with Vivado and for that reason a Tcl script is used to create the Vivado design. Ensure the zip file is extracted in a clean directory. Then change directory into the “viv” directory at the base directory where the zip is extracted. If it’s not there, create the directory. Within this directory, ensure the vivado tools are sourced and type:

>>*vivado -source ../scripts/vivado\_prj\_create.tcl*

This will create a Vivado project and have two streaming IPs the software from SDSoC will interface with. This is a good time to look at the Vivado project to look at the rx\_stream and tx\_stream IPs; which are AXI4-Stream Data FIFOs. Notice the rx\_stream FIFO has a completely unconnected M\_AXIS interface while it’s constantly receiving counting data as an input:

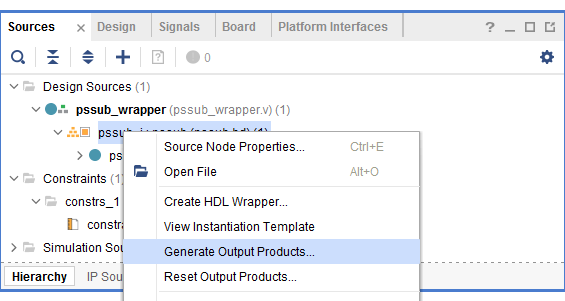


Next, take a look at the tx\_stream IP and notice how the S\_AXIS port is completely unconnected. This port will be interfacing with SDSoC generated software. There’s a system ILA on the other end to monitor the data coming out.



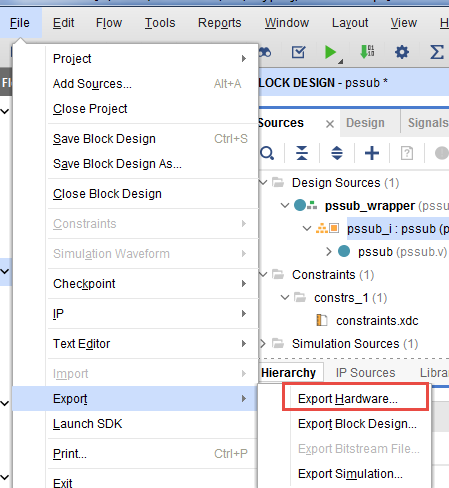
Lastly, notice the MIG core and how we’ve placed a System ILA on the MIG AXI interface. This system ILA can be looked at to see the traffic to the Memory Controller (MIG).

Generate the output products by right-clicking the BD in the sources window and selecting “Generate Output Products…”



Keep the defaults and select “Generate”

While the Design Runs are running, you should be able to Export the HDF. The generation of the design runs should take a couple minutes, but you shouldn’t have to wait till each IP synthesized. Below is how to export the HDF:



Then select the Defaults and export the HDF local to the project. This will export the HDF into the moverlab.sdk directory.

After the HDF is exported, we’re ready for PetaLinux

**Step 2: PetaLinux**

The HDF will be imported into the PetaLinux project utilizing the Template flow. First create the PetaLinux project. We’ll assign the project name of “petamover,” although this is arbitrary.

>>*petalinux-create -t project -n petamover --template zynqMP*

After the project is created, change directory into petamover:

>>*cd petamover*

Now it’s time to import the hdf created from Vivado:

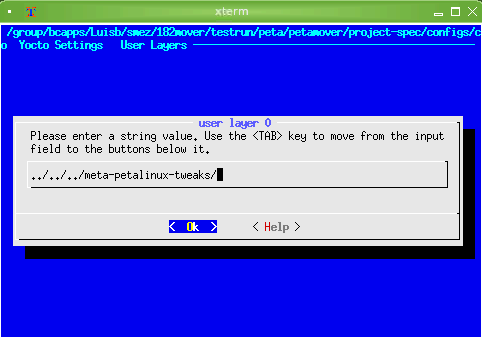
>>*petalinux-config --get-hw-description=../../viv/myproj/moverlab.sdk/*

This command inherently imports the HDF. The menuconfig that pops up after this command will allow us to add a meta layer. This meta layer will add the following settings:

1. The SD card is automatically mounted to the /media/card when it’s plugged into the ZCU102
2. The output from the ‘ls’ command is colorized
3. Some additional utility aliases, such as ‘lh’ which produced a detailed ls listing with human-readable file sized are defined.

Set the user layer 0 to the meta-petalinux-tweaks located in the directory where we extracted files. The setting is located here:

Yocto Settings-> User Layers -> user layer 0



(Make sure to go back three directories since everything is relative to a build directory within the PetaLinux project)

Next, copy the device tree updates into the project directory:

>>*cp ../../device\_tree/system-user.dtsi ./project-spec/meta-user/recipes-bsp/device-tree/files/*

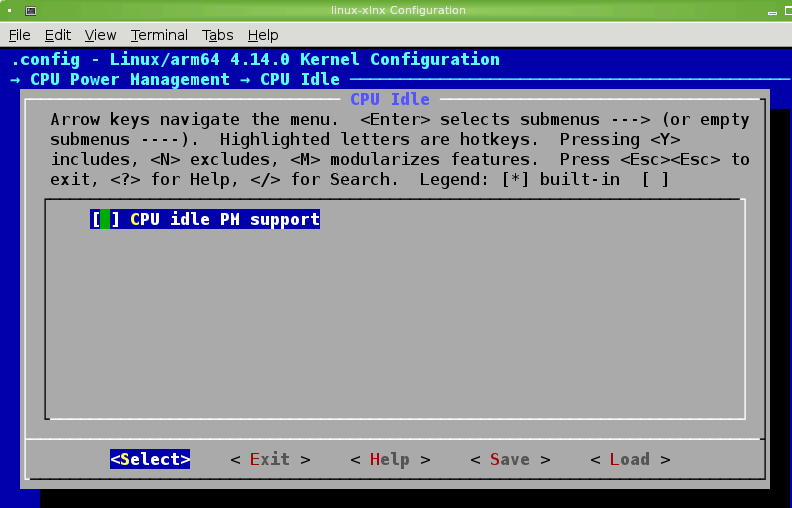
Modify the kernel by disabling some of the power management features and enable the APF drivers. We’re disabling the power management features because we want to debug with the ILA cores. If we keep these Power Management features, the monitoring with the ILA cores will cause a hang on our embedded Linux session.

The APF drivers need to be installed so we can use the framework from SDSoC.

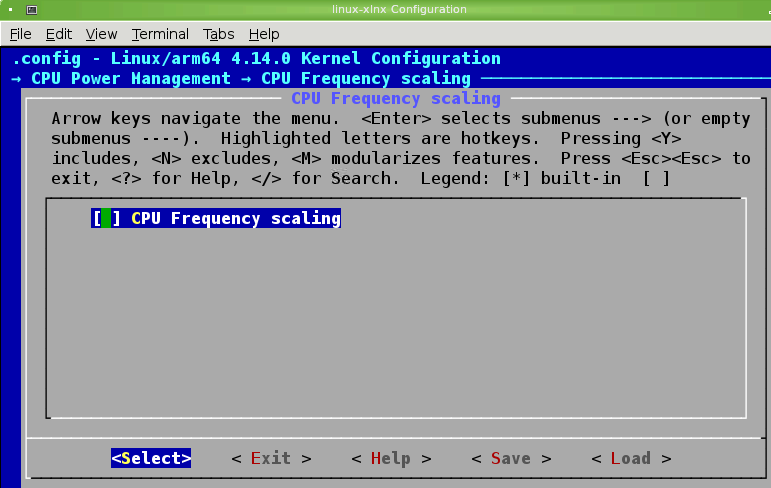
>>*petalinux-config -c kernel*

To disable the power management feature go to:

CPU Power Management->CPU Idle->CPU idle PM support

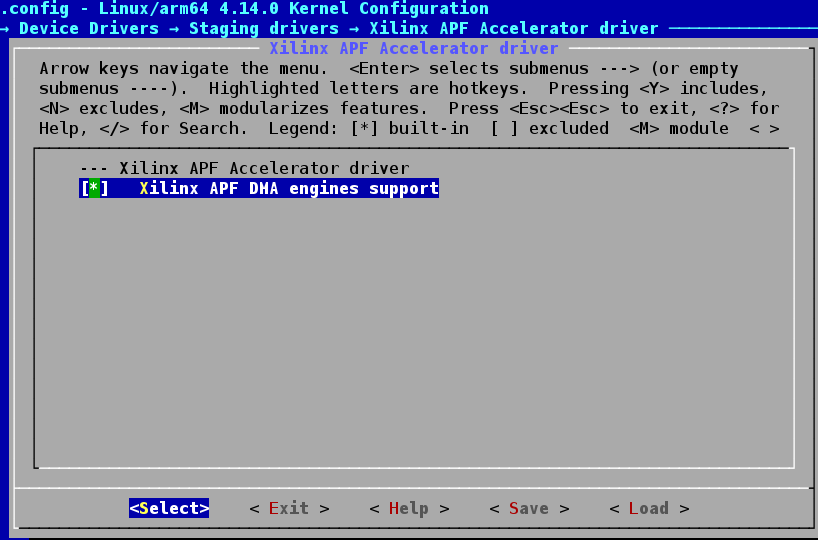


CPU Power Management->CPU Frequency scaling->CPU Frequency scaling



To enable the APF drivers go to:

Device Drivers->Staging drivers->Xilinx APF Accelerator driver->Xilinx APF DMA engines support



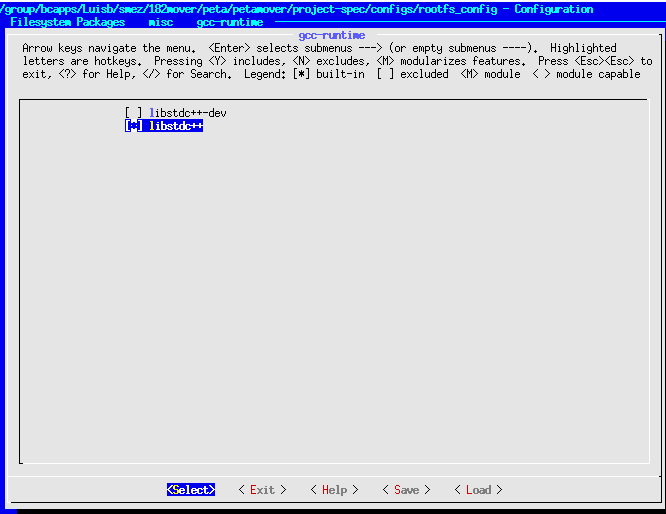
Save the changes to the kernel options.

Next, modify the root file system to add the libstdc++ gcc-runtime libraries.

>>*petalinux-config -c rootfs*

To add the gcc runtime libraries, go to the following:

Filesystem Package->misc->gcc-runtime->libstdc++



Now build the project:

>>*petalinux-build*

((((STOP HERE for the first portion!!!!!)))

After the project builds, change directory to where the zip file was extracted:

>>*cd ../../*

Then source the move\_linuximagefile.sh script to copy (and organize) the relevant files for SDSoC into the sw\_files directory.

>>*source ./scripts/move\_linuximagefile.sh*

**Step 3: Create the DSA**

Within the **Vivado Tcl Console** (with the Vivado project open), source the createdsa.tcl script in the ‘scripts’ directory; as shown below.

>>*source ../scripts/writedsa.tcl*

While it’s building, open the writedsa.tcl script and look at how the commands correlate to the Vivado IPI project.

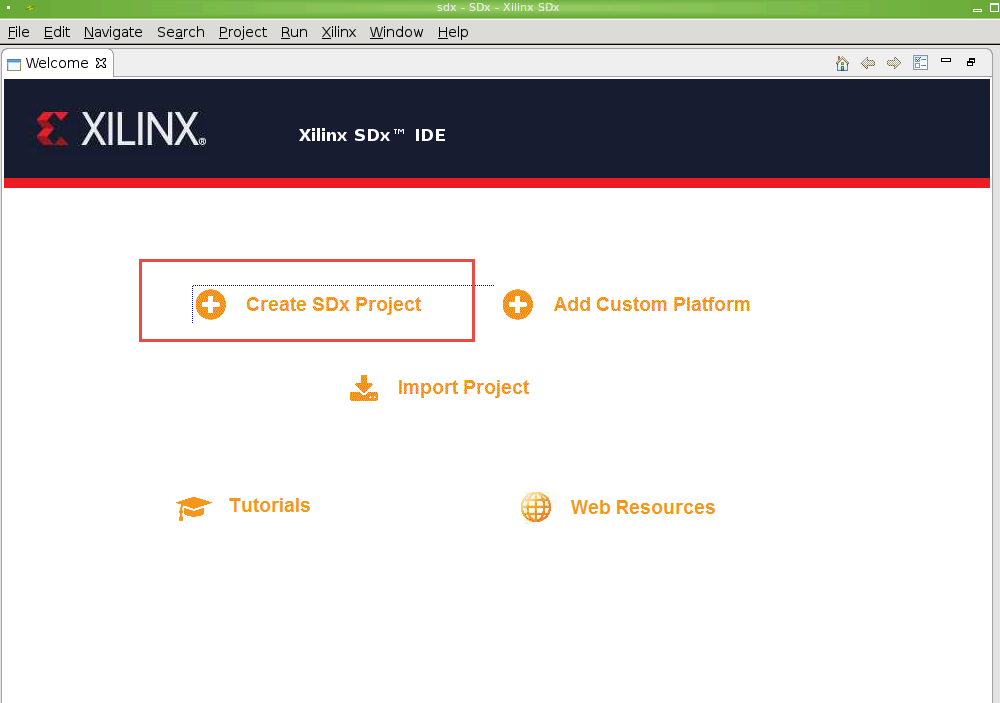
Key takeaway 1: Look at the AXIS\_PORT and how they map to the AXI Stream IPs.

Key takeaway 2: Notice how the AXI\_PORT property enables SDSoC to understand the memory mapped resources available.

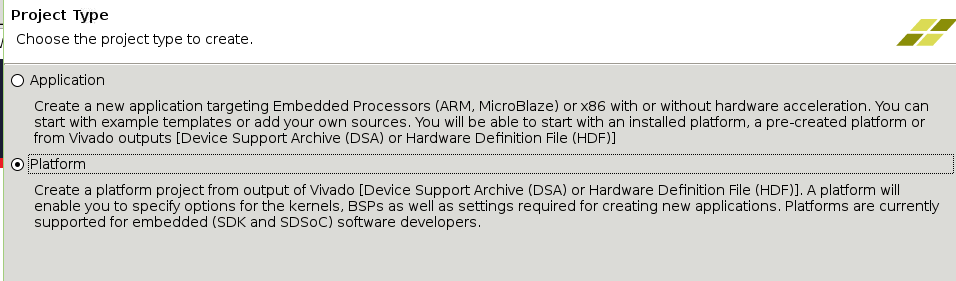
Key takeaway 3: Look at how SDSoC knows about the clocks, resets, and interrupts.

**Step 4: SDSoC - Creating the Platform**

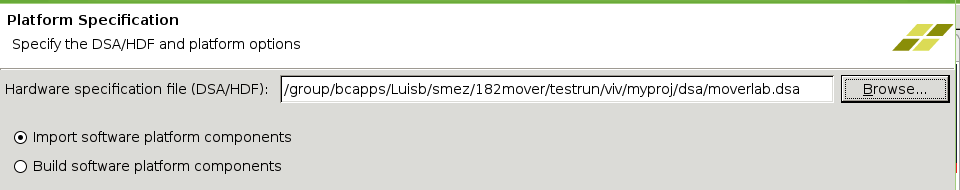
SDSoC is used to create the platform. Launch SDSoC by typing ‘sdx’ in the terminal. The welcome screen should show up and now select the “Create SDx Project” button:



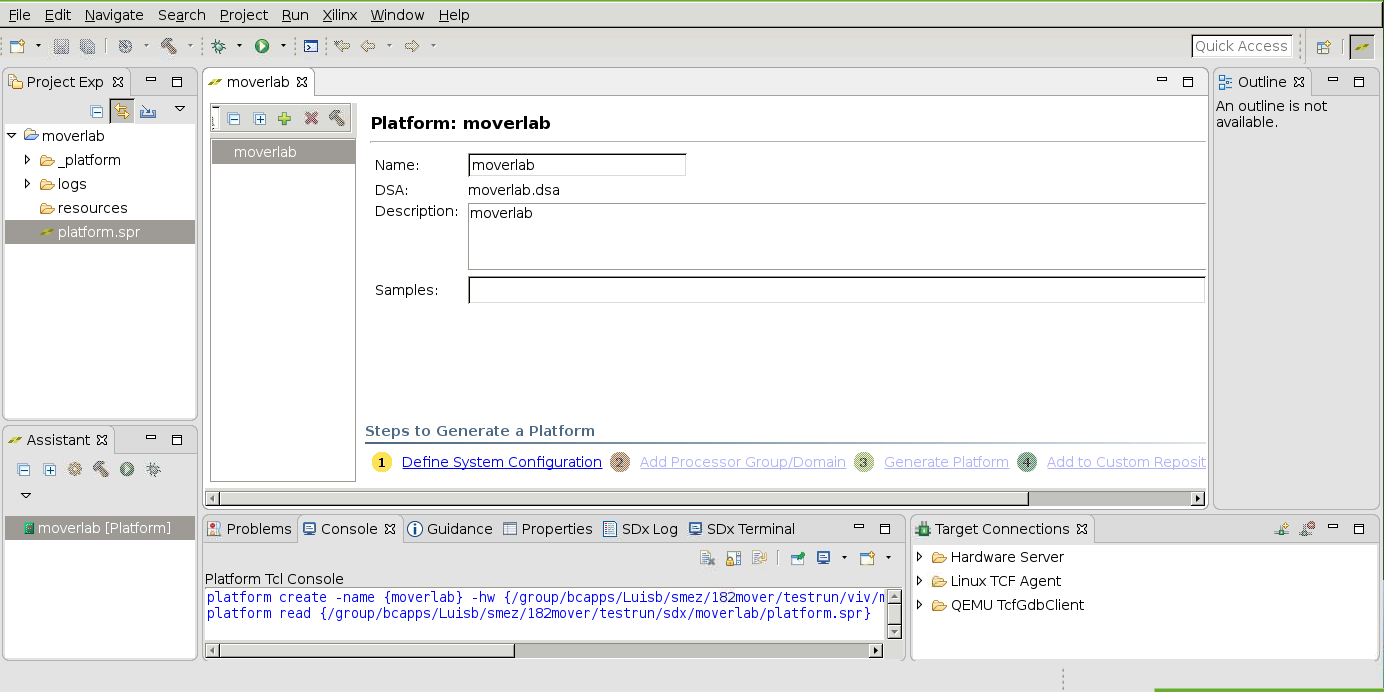
Select the “Platform” option as shown below and select “Next”:



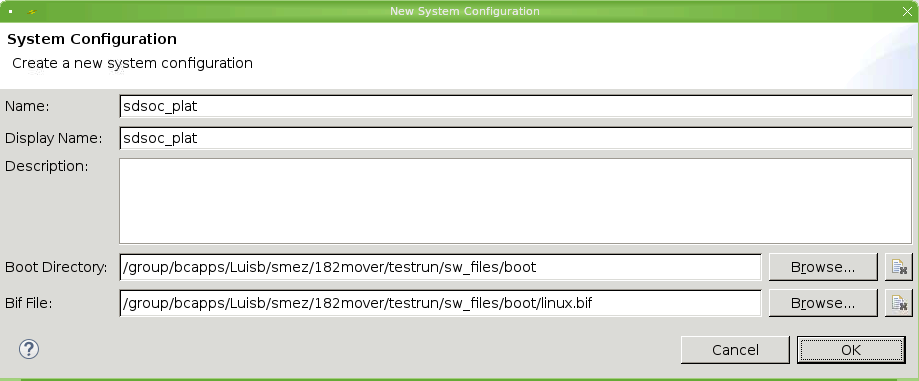
Next, specify the DSA built from Vivado. The DSA is located in the zipExtract/viv/myproj/dsa/ directory; as shown below and select Finish:



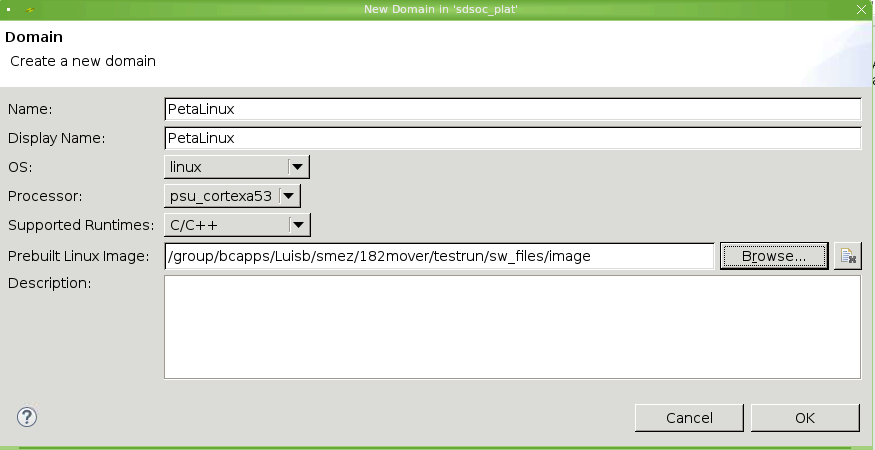
The following screen should be shown after beginning the Platform project:



Select the “Define System Configuration” step to select the boot files created from PetaLinux and also provide a Name. Then select OK.



Now select the “Add Processor Group/Domain” to point to the fit image created from PetaLinux as shown below:

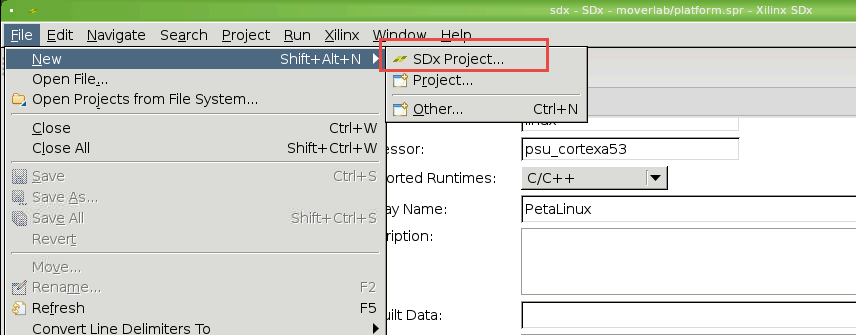


Lastly, select to “Generate Platform.” This will generate the platform and add it local to the project. Next, it’s time to create the application. Do not close SDx yet.

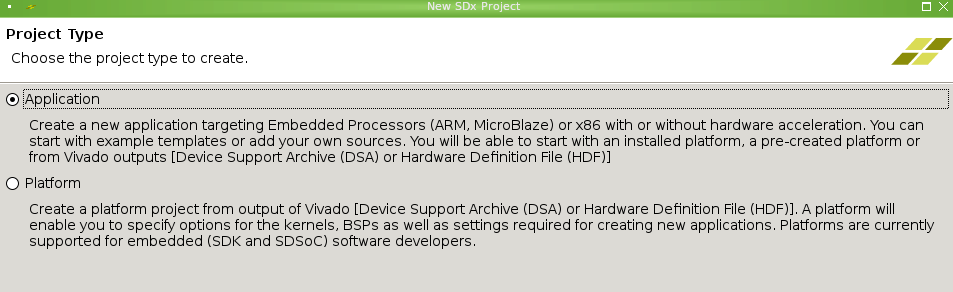
**Step 5: SDSoC - Creating the Linux application**

Now it’s time to create the Linux application with the ability to direct the streaming data in and out of the application. This particular application will take 64-bit input streaming data at 100MHz and send it directly to PL DDR. Then it will take that data from PL DDR into PS DDR. Lastly, it will take the data in PS DDR out to a 64-bit streaming output at 100Mhz.

While SDx is still open, create a new SDx Project by selecting File->New->SDx Project…



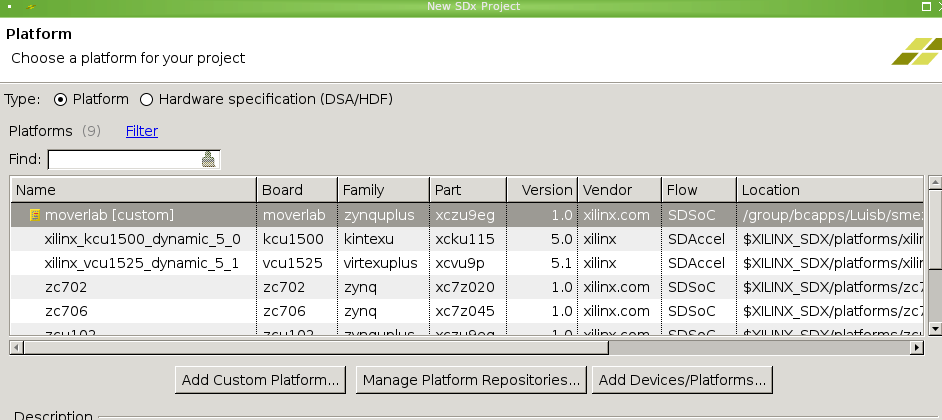
This time, instead of creating a new Platform project, create an Application:



Give the application a name. The name of the application is arbitrary and in this example, it is called “dmalinapp”

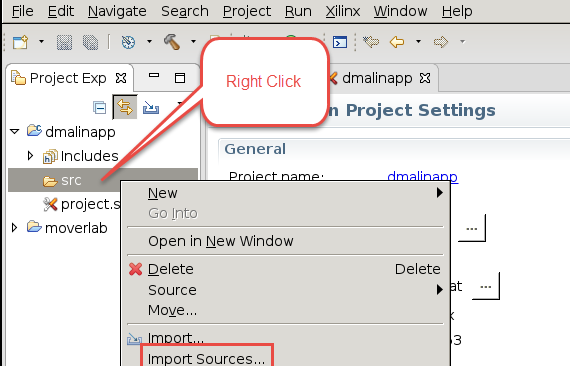


Select the custom platform that was recently create. It’s is called moverlab:

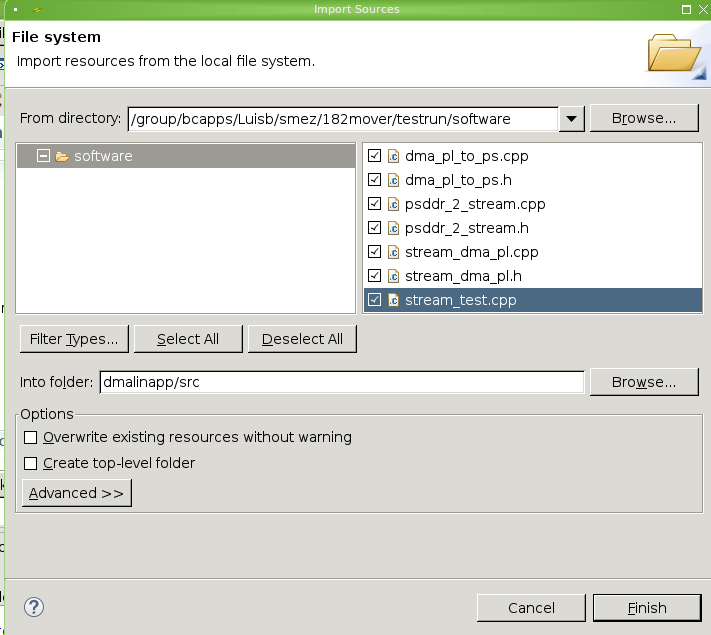


Select the defaults for “System Configuration” as well as the “Templates” and select “Finish”.

Right click the “src” directory in the Project Explorer and select to import sources:

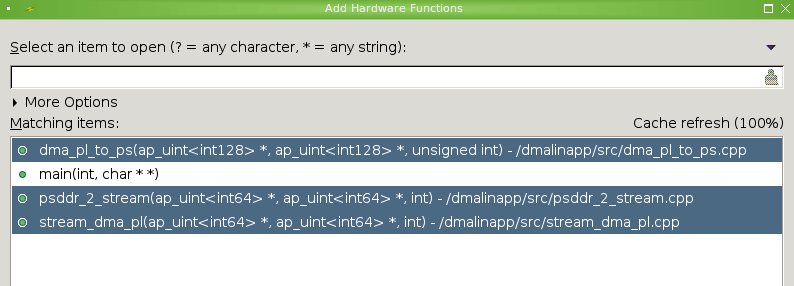


Within the extracted zip file directory, locate the “software” directory and import all of the source files as shown below:

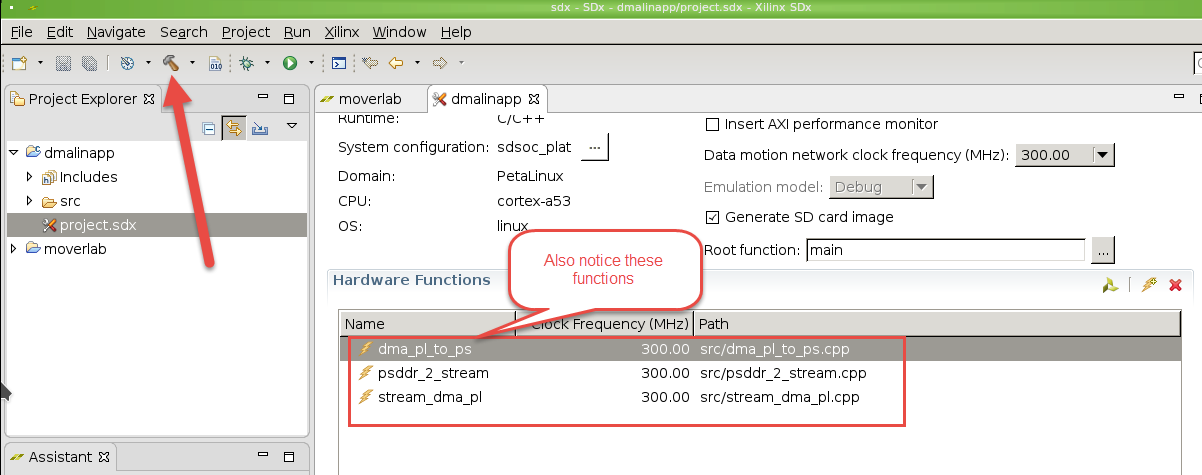


Select/Highlight the following hardware functions to be accelerated(these are the datamovers) as shown below

* dma\_pl\_to\_ps
* psddr\_2\_stream
* stream\_dma\_pl



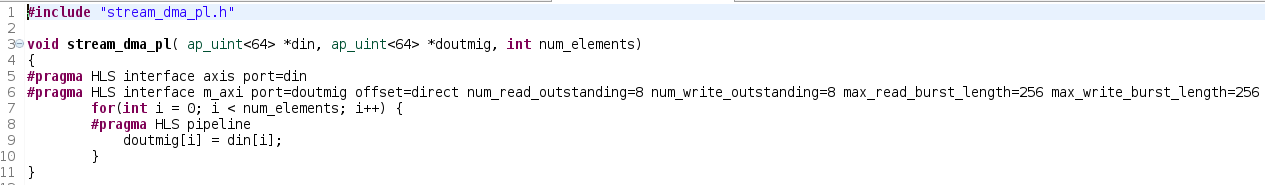
Now it’s time to build the project. Click on the hammer shown below and yell out “It’s hammer time!”



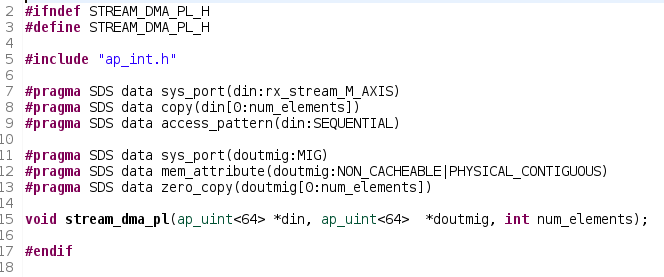
Building this application will take some time. SDx will build new bits during this first build. After the first set up bits are created, it’s very unlikely that the bits will not need to be re-created as long as the hardware functions are not modified. So subsequent builds can be built quickly.

While SDSoC builds

As SDSoC builds the application, this is a good time to look at the source files. First inspect the stream\_dma\_pl.cpp and the stream\_dma\_pl.h. The stream\_dma\_pl.cpp shows the function being done with the stream\_dma\_pl call:

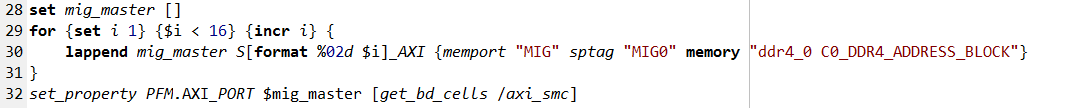


This a simple data mover that takes input data and send it right back out through another pointer. The number of beats is determined at the time the call is made. Notice the HLS pragmas; one is to indicate the port type while the other is a performance enhancement. Now look at the stream\_dma\_pl.h file to look at the special SDS pragmas:



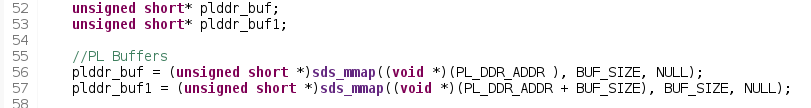
Notice on line 7 the sys\_port pragma is mapping the din pointer to the rx\_stream IP port that is the M\_AXIS stream. This was determined from the script used to create the DSA.

Also notice that sys\_port is used to map the dout pointer to the PL Memory controller. This is also referenced in the dsa.tcl where the memport was labeled MIG:



Lastly notice the zero\_copy pragma is used to tell SDSoC the stream\_dma function will have the dout port as shared memory through an AXI master bus interface.

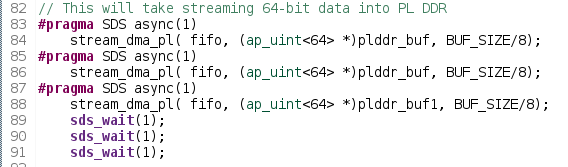
Next look in the stream\_test.cpp file and notice how the plddr\_buf and plddr\_buf1 pointers are memory mapped to the physical MIG addresses:



Also, look at the fifo pointer to see memory is allocated for the streaming source. In theory, this allocation isn’t necessary, but I keep it there since it does no harm.

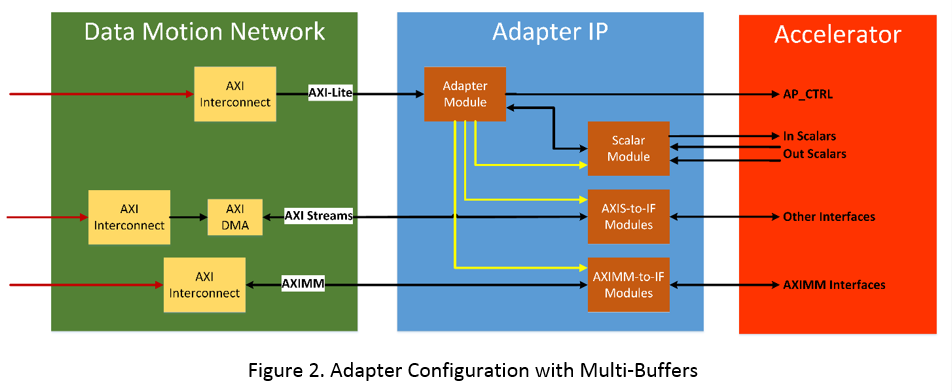


Lastly, look at the call to stream\_dma to see how the streaming data is captured to onto MIG:



First thing to notice is that BUF\_SIZE is divided by 8, this is because steam\_dma\_pl data types were 64-bit wide, which is 8 bytes. If you look at the loop for the data movement in the hardware function, it’s based on beats. BUF\_SIZE was intended for bytes counts; so that’s why it’s divided by 8.

Notice that there are two stream\_dma\_pl calls with the same plddr\_buf passed. This is done to flush out the FIFO while it’s overflowed since the counter has been running for a while now. The async call queues up the transaction in the adapter IP. So the first transaction is essentially thrown out.



The sds\_wait call simply waits for the transactions to complete.

Walk through the other accelerated functions to see how they were implemented. Reference the SDSoC User Guide for details.

Once SDSoC completes

Take the files in the /Debug/sd\_card directory and copy them to your SD card. Now boot Linux and run the executable from /media/card/your\_elf\_name.elf and it should complete successfully. Now read the registers with devmem at address 0x80000000 to see if you have data incrementing every 8 bytes. You’ll have to read them 4 bytes at a time. So you would read from address 0x80000000 and 0x80000004.

Using the ILA:

Within the SDx project, look in the following directory to find the ltx. This ltx will be used to debug.

Debug\\_sds\p0\vivado\prj\prj.runs\impl\_1

With the ILA core, have fun and trigger on AXI Streaming transactions and compare them to what software displays. Remember to be mindful of data types.