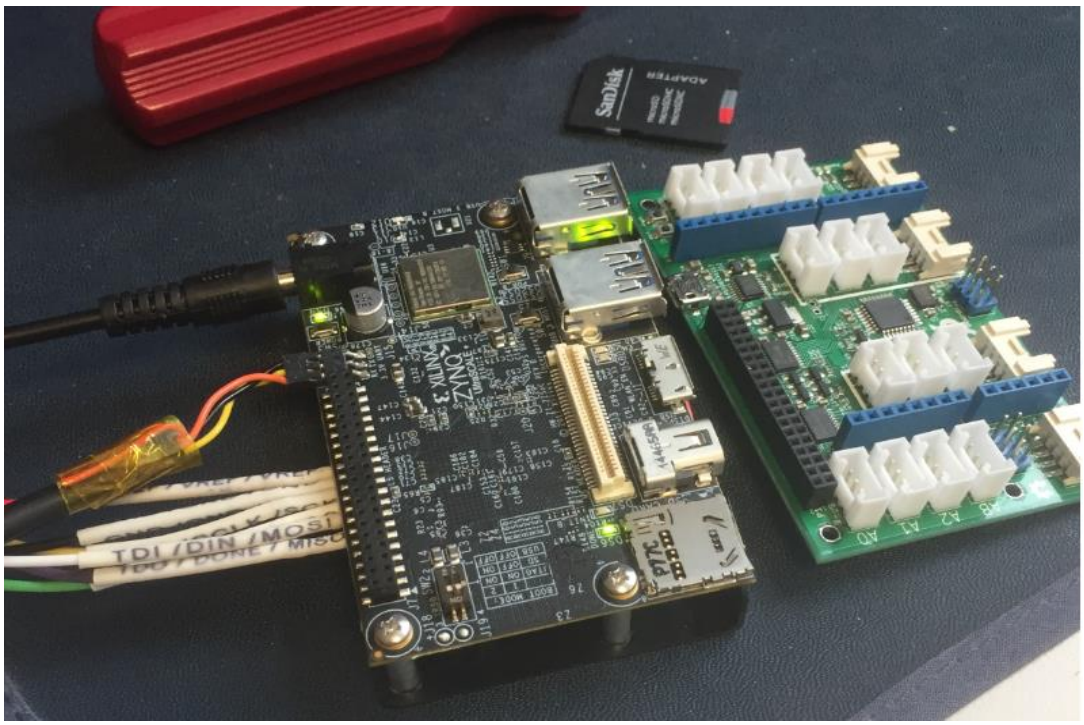
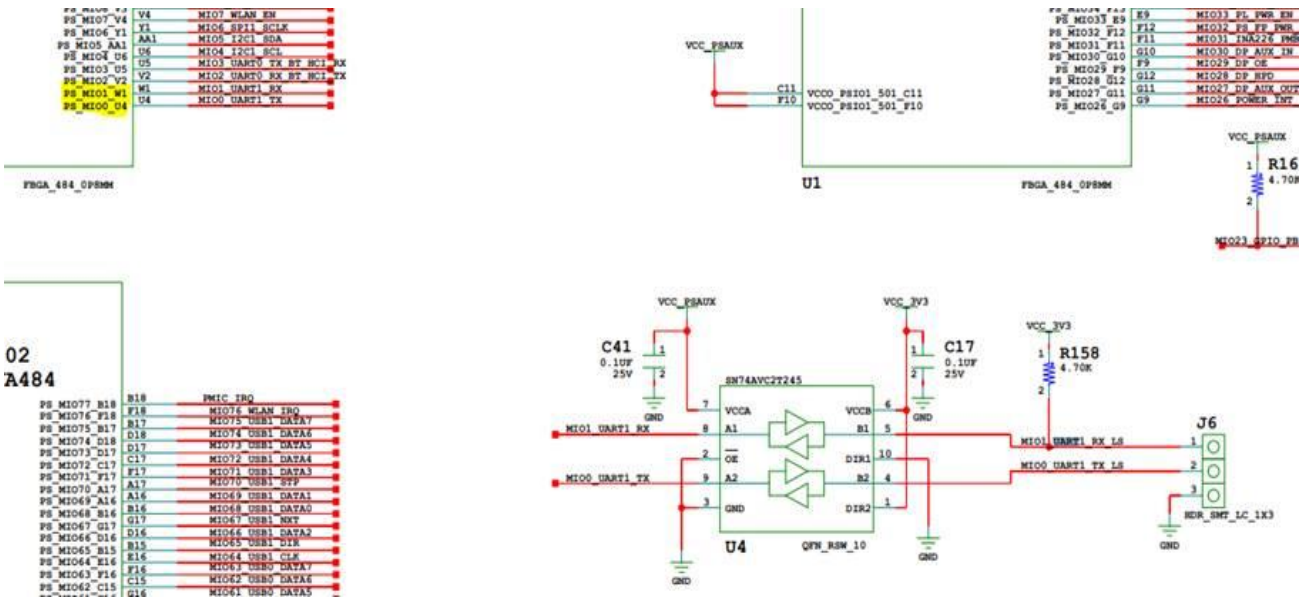
In this demo we shall explore the ZZSOC (Ultra96) board.

Hardware Setup:

In order to debug, I have used fly-leads connected to the JTAG connections via the PMOD on the board. I have also connected a UART via the J6.



Note: there was also a daughter card that I could have used (shown on the right). There is a UART on this that can be used. Users should consult the schematic for their board to determine the MIO pins for each UART. For example:



So, I will be using UART that is connected to MIO 0 .. 1 in the PS.

Vivado Project Creation:

There is a few ways that users can do this. Users can start in Vivado 2018.2, and use the board definition files (BDF) available on GIT to create a Block design. This can be cloned or downloaded.

For example, I just cloned this

- mkdir repo
- cd repo
- git clone https://github.com/Avnet/bdf

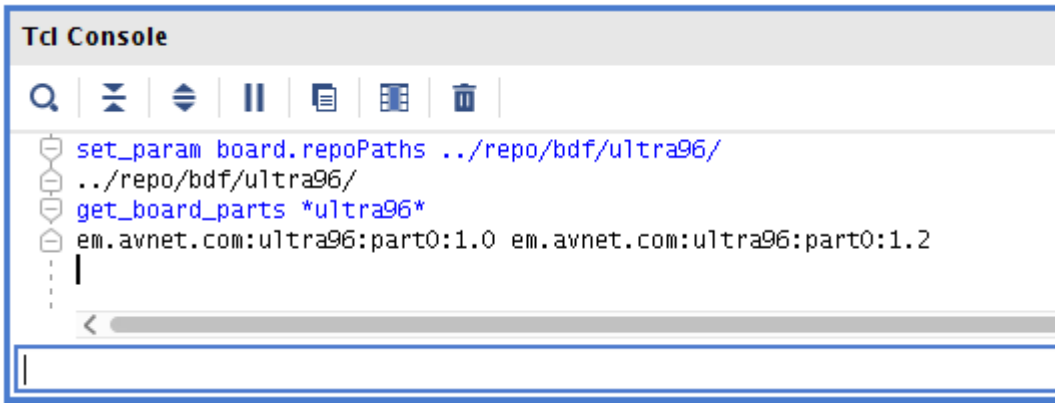
Then add the board files via the TCL console.

Note: do this before opening/creating the project:

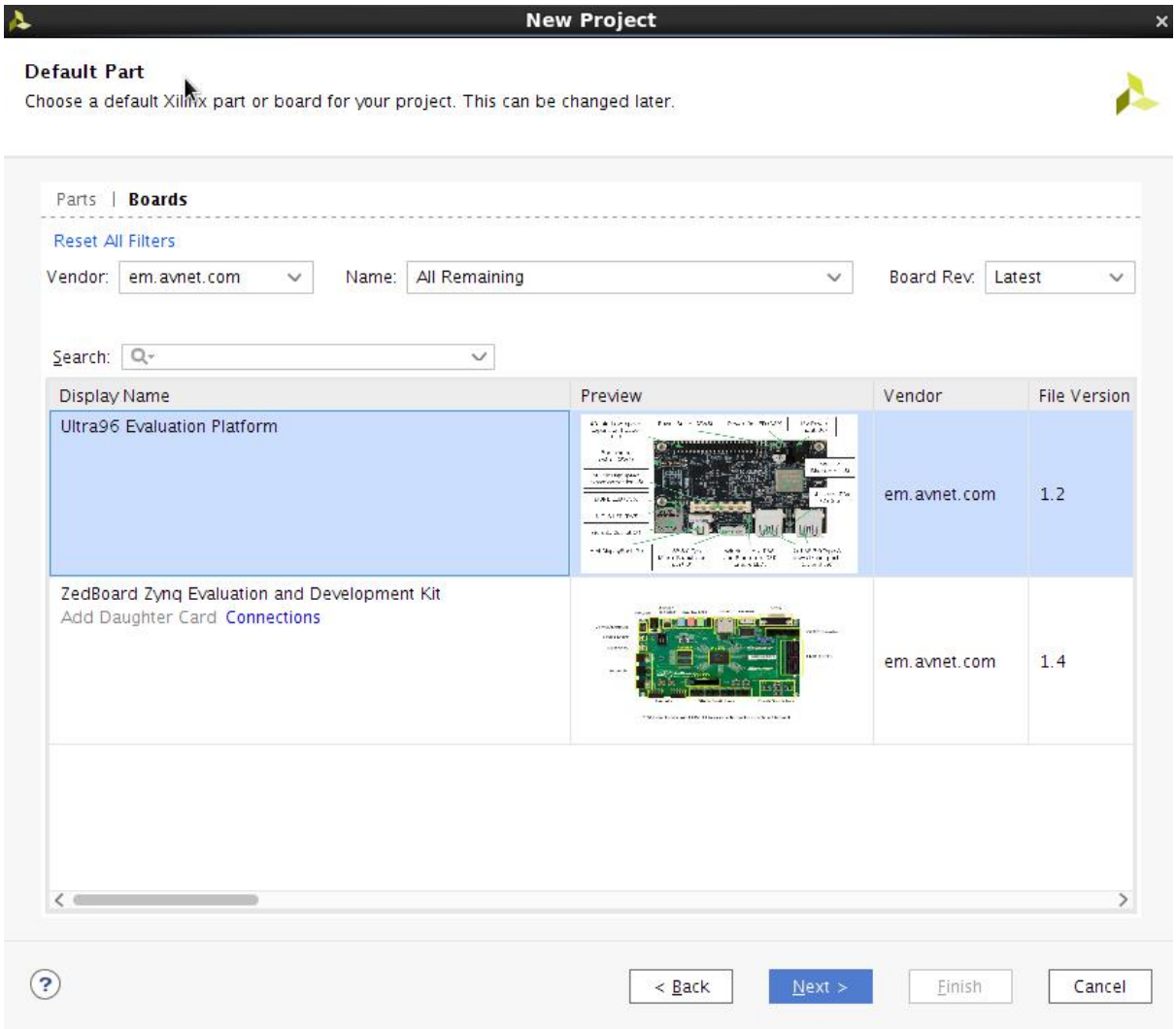
- set_param board.repoPaths <path>/repo/bdf/ultra96

Then use the command beloe to verify:

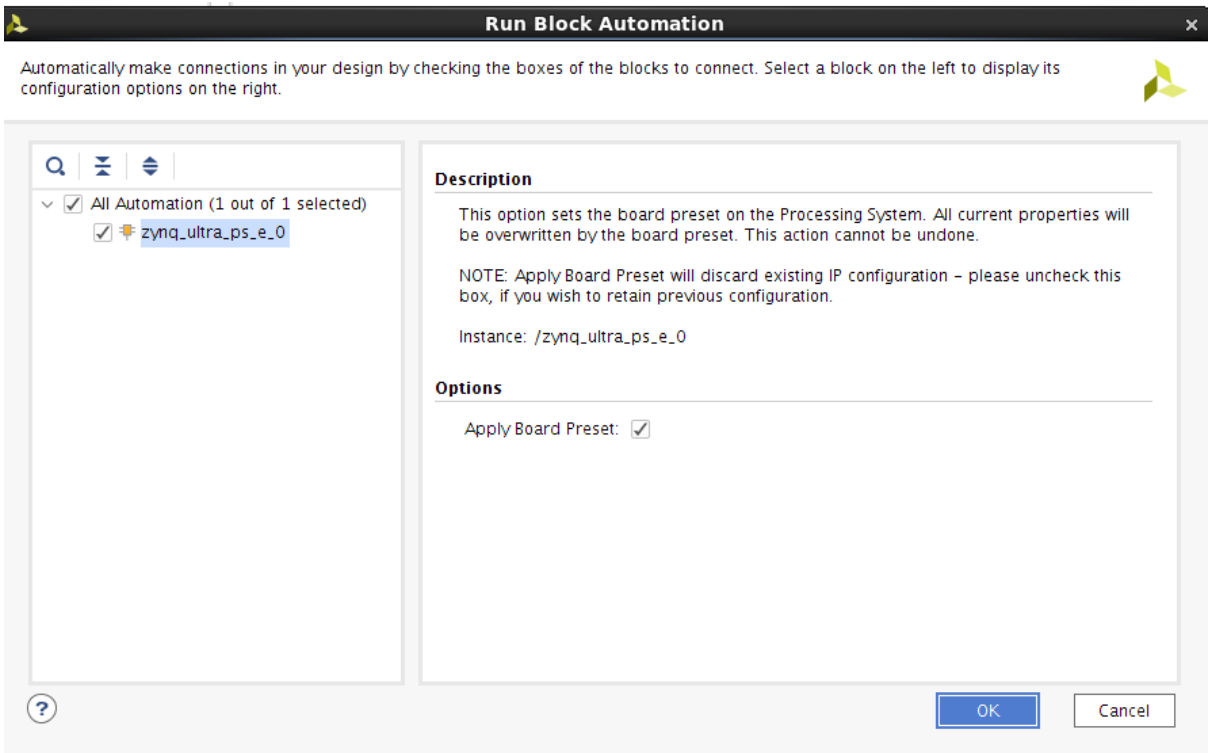
- get_board_parts *ultra96*



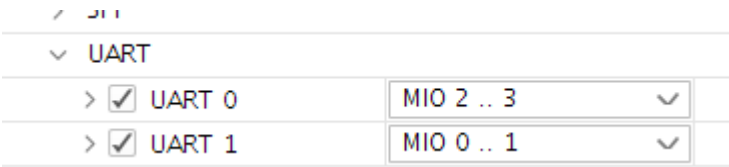
Now, create the Vivado Project and target the Ultra96 board:



If the user adds the Zynq Ultrascale PS from the IP catalog, they tools will allow the user to Run Block Automation. This will populate the PS with the relevant settings for the Ultra96 development board. Such as the DDR settings, clocks, and available IP and MIO pins:

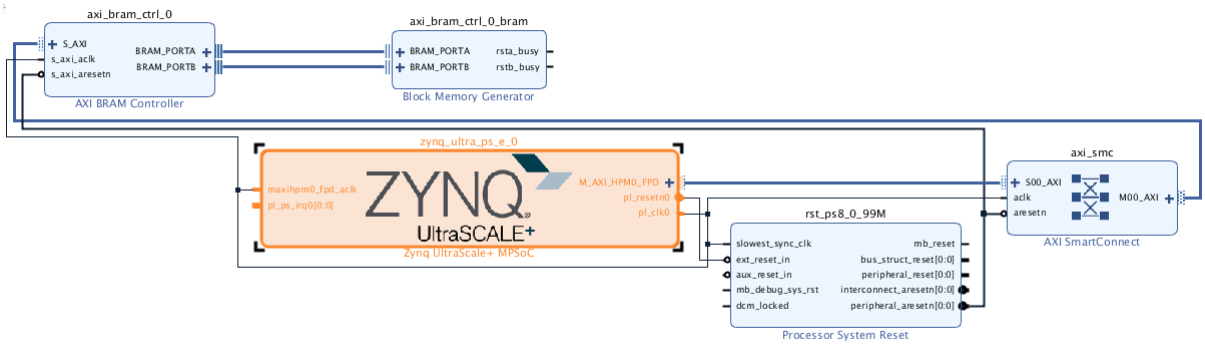


User can double click on the PS in the IPI canvas to view these settings. For, example the UART. Here we can see there are two enabled:



Note: As discovered above in the schematic, MIO 0:1 is connected to J6 on the board. So, UART 0 can be disabled if using the J6 on the board.

User can also add some IP in the PL for testing purposes. For example, the AXI BRAM can be added:



The respective address map can be seen:

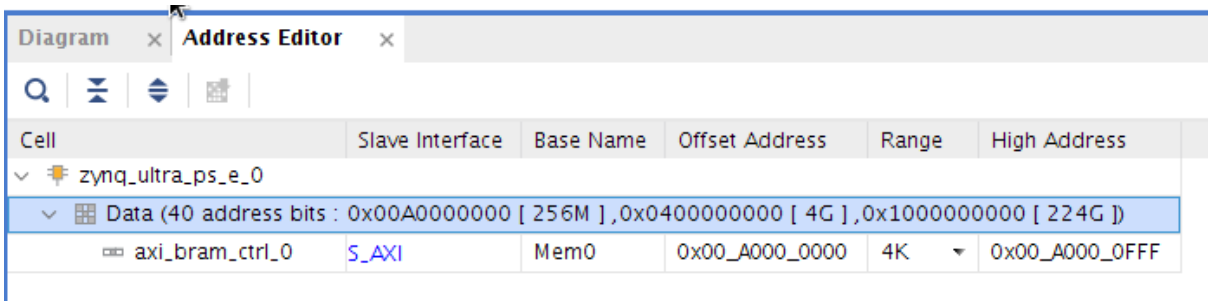


Diagram | Address Editor

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
zynq_ultra_ps_e_0					
Data (40 address bits : 0x00A0000000 [256M] ,0x0400000000 [4G] ,0x1000000000 [224G])					
axi_bram_ctrl_0	S_AXI	Mem0	0x00_A000_0000	4K	0x00_A000_0FFF

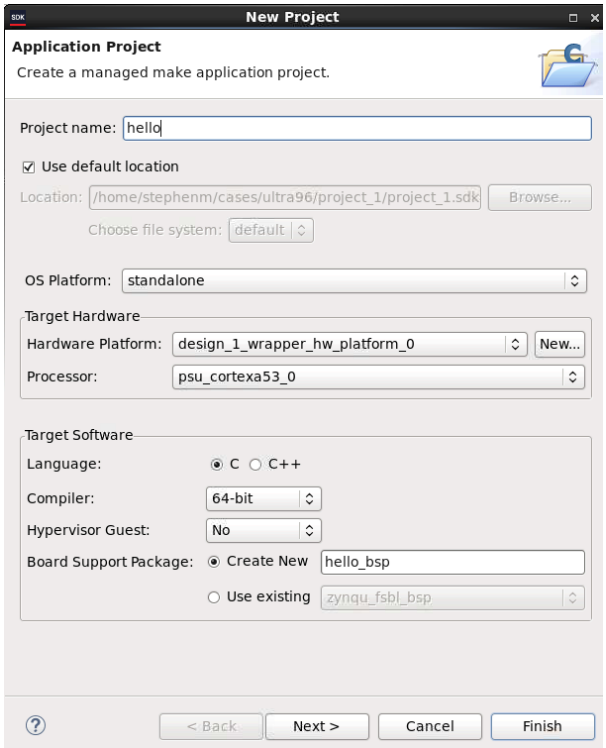
Complete the following steps to export to SDK:

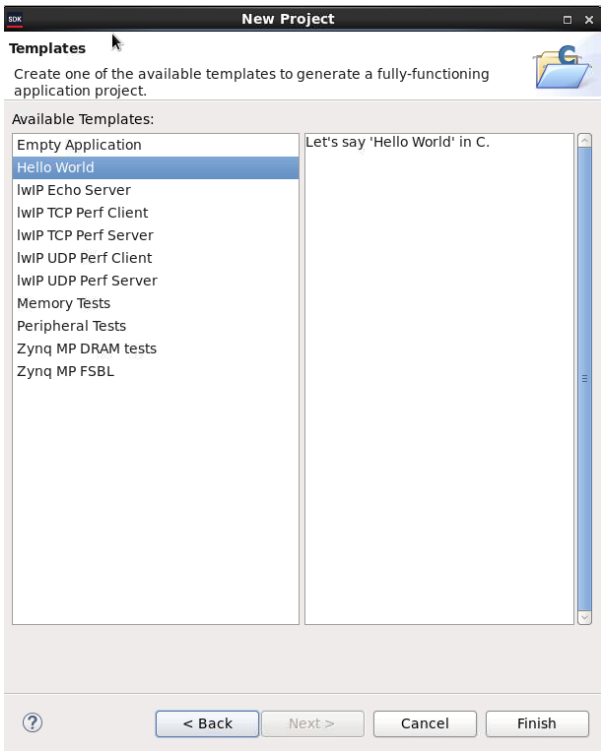
- Generate Output Products
- Create the HDL wrapper
- Generate Bitstream
- File -> Export -> Export Hardware (include bit)
- File -> Launch SDK

Software Application Creation:

User should have launched SDK from the previous step. To test the UART, there is a Hello World template available in the SDK.

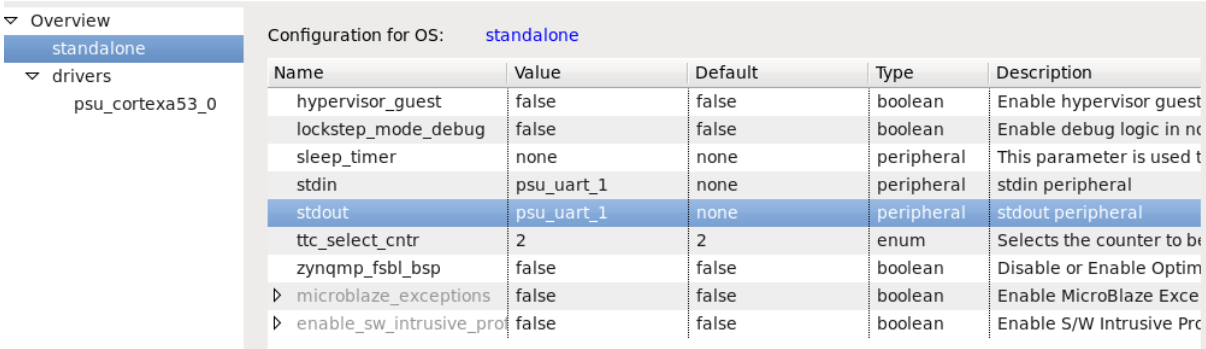
File -> Application Project





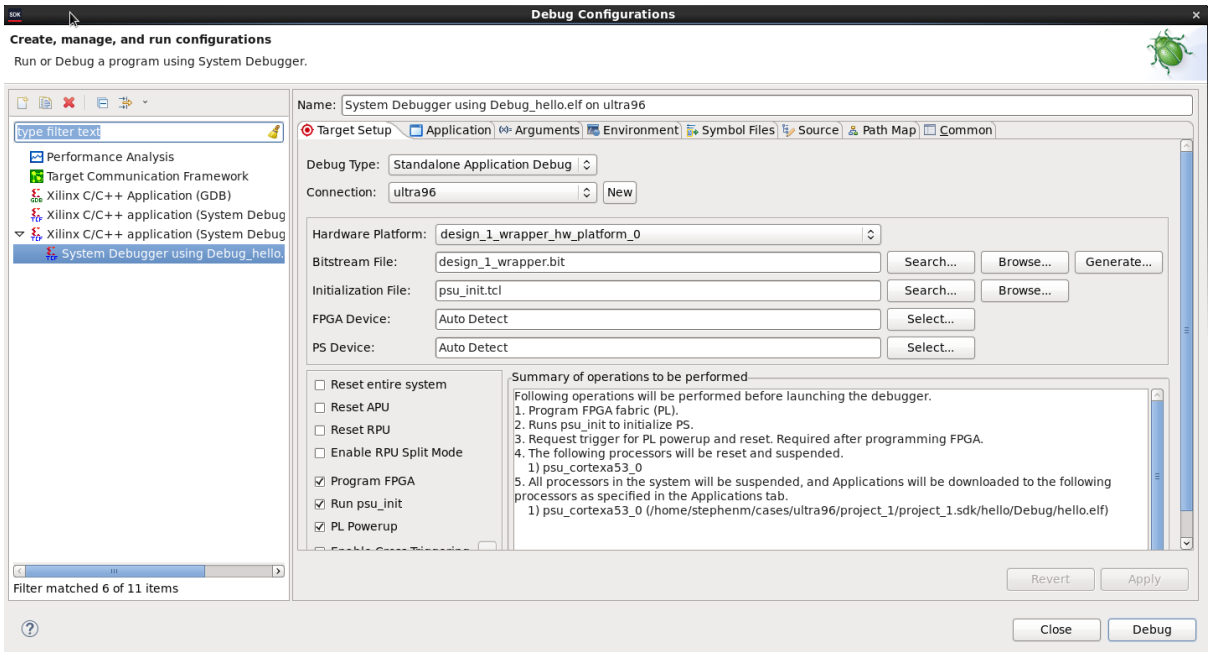
Select Finish. This will create, and automatically build the ELF.

Users may need to change the STDIN/OUT settings in the BSP to make sure that the UART_1 is used. To do this, right click on the newly created hello bsp, and select Board Support Package Settings, and make sure that the UART 1 is used:



Create a Debug Config:

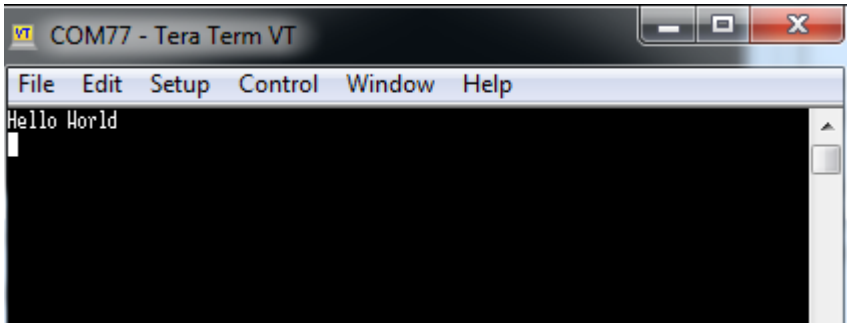
Right click on the hello application, and select Debug As -> Debug Configurations:



Double click on the Xilinx C/C++ application (System Debugger) to create a new debug config. Make sure that the Program FPGA is selected.

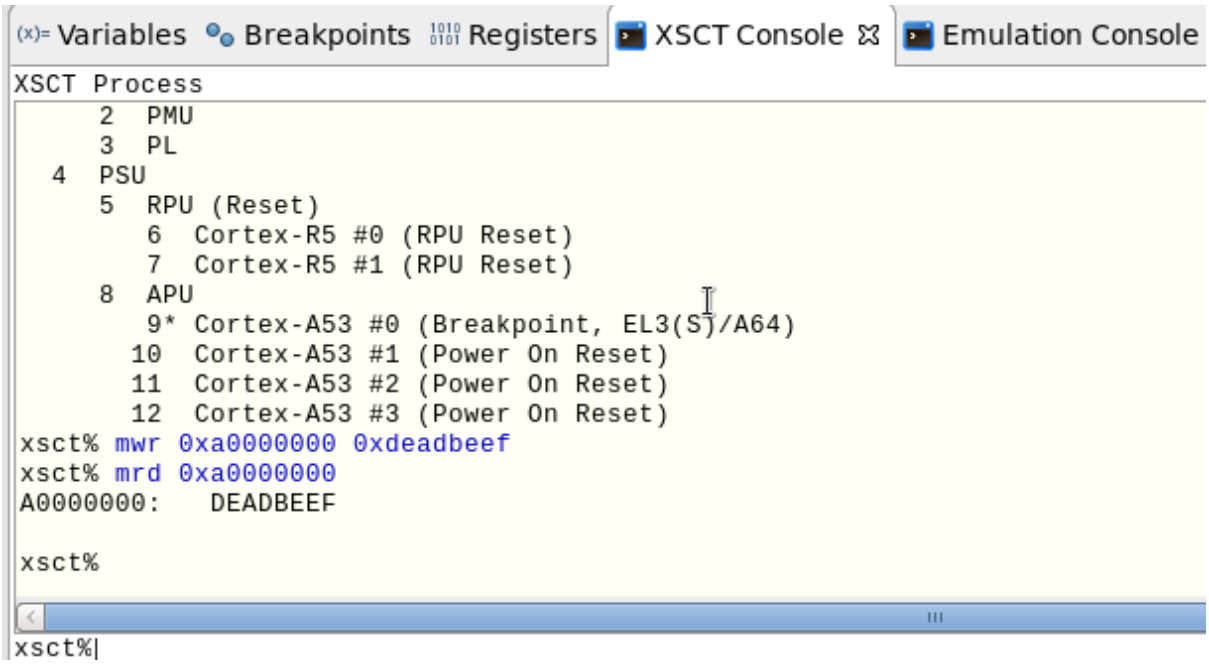
Open a serial port with Baud 115200

This will launch the debug perspective. The debugger will init the PS with the settings in the psu_init.tcl that was created when we exported to SDK. It will download the bitstream, and download the ELF and jump to the main(). User can step through the code, or just resume exit(). The user should see the “Hello World” on the serial port:



User can also do a memory read on the AXI BRAM over the XSCT (Xilinx -> XSCT console)

- connect
- targets 9 (the cortex a53 #0)



Note: User can also do a memory test using the templates available. However, the XSCT is a quick way to debug.

Building a Petalinux Image:

The quickest way for a user to boot linux on the Ultra96 is to use Petalinux and the Ultra96 BSP available from Xilinx.com. There are pre-built images that the user should use to boot from the SD card

MD5 SUM Value : b5d58587eb3e980f0873d419cb3a1409

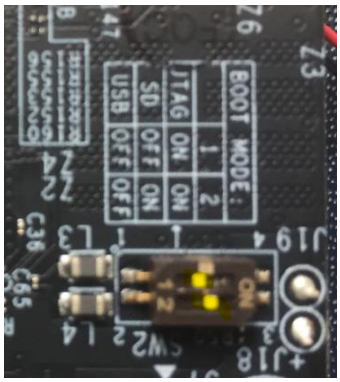
Ultra96 BSP (BSP - 2.76 GB)

MD5 SUM Value : 61a7d6952a4edc52962acf73462ab941

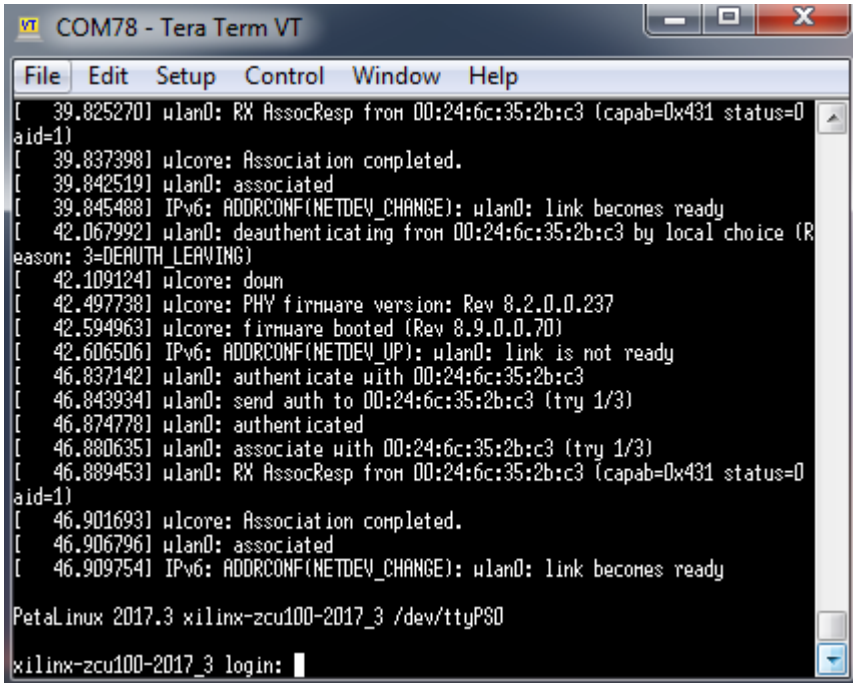
Then in petalinux 2018.2, simply use the command:

- petalinux-build -t project -s <path to the BSP>.bsp

Then simply copy the BOOT.BIN, and image.ub from the pre-built folder onto the micro SD Card. Make sure that the boot mode is set to SD:



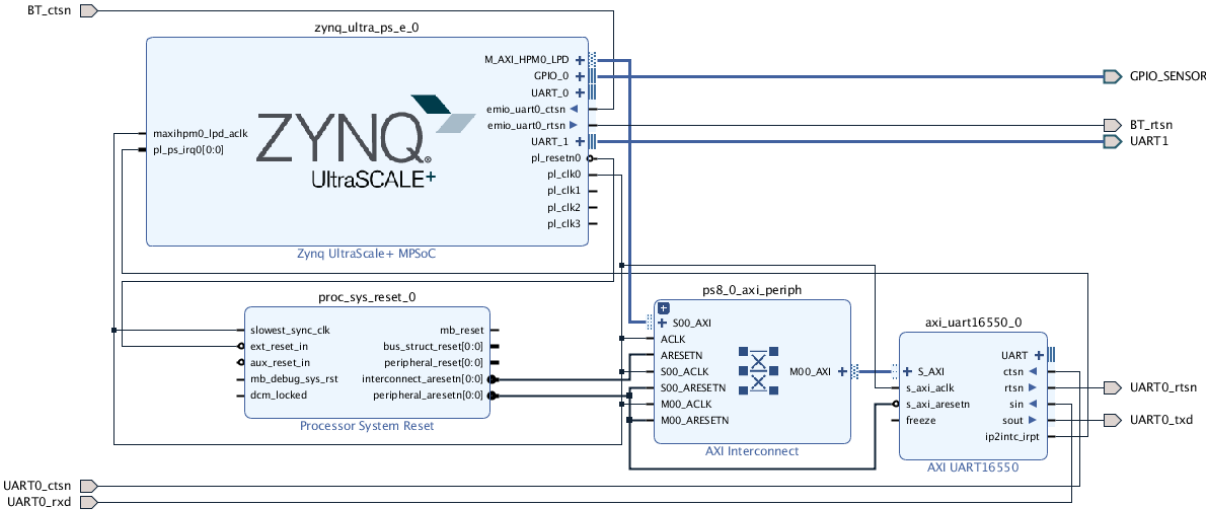
Note: the BSP will use the extension card. So, will be on the micro USB:



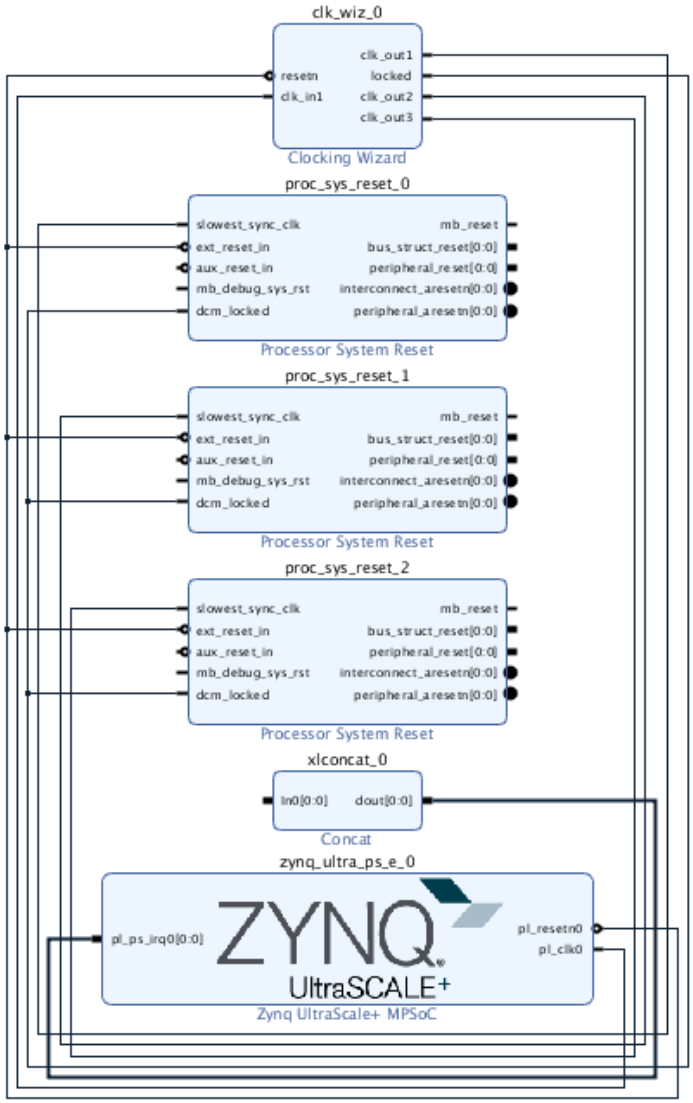
The user can also open the HW design that is used in the BSP in Vivado 2018.2, and they could use this as a basis for future HW designs:

s03) (H:) ▸ cases ▸ ultra96 ▸ xilinx-ultra96-reva-2018.2 ▸ hardware ▸ xilinx-ultra96-reva-2018.2 ▸

Name	Date modified	Type	Size
.Xil	15/06/2018 10:58	File folder	
xilinx-ultra96-reva-2018.2.cache	09/08/2018 16:24	File folder	
xilinx-ultra96-reva-2018.2.hw	09/08/2018 16:24	File folder	
xilinx-ultra96-reva-2018.2.ip_user_files	09/08/2018 16:24	File folder	
xilinx-ultra96-reva-2018.2.runs	09/08/2018 16:24	File folder	
xilinx-ultra96-reva-2018.2.sdk	09/08/2018 16:24	File folder	
xilinx-ultra96-reva-2018.2.sim	09/08/2018 16:25	File folder	
xilinx-ultra96-reva-2018.2.srcs	09/08/2018 16:24	File folder	
vivado.jou	15/06/2018 10:44	JOU File	1 KB
vivado.log	15/06/2018 10:56	Text Document	98 KB
xilinx-ultra96-reva-2018.2.bit	15/06/2018 10:56	BIT File	5,439 KB
xilinx-ultra96-reva-2018.2.hdf	15/06/2018 10:56	HDF File	889 KB
xilinx-ultra96-reva-2018.2.xpr	09/08/2018 16:25	Vivado Project File	17 KB



Users can also build the Linux image from HDF file. For example, I create the HW below:



Note: This is the HW platform that is used in the next section. The TCL file to build this is provided here.

Note: Users will need to add the Ultra96 board files before creating the project:

Preform the following tasks to generate the HDF file:

- Generate Block Design
- Create HDL wrapper
- Generate bitstream (optional)
- File -> Export -> Export Hardware

Follow the steps below to create the Linux image for the HW system above. The BSP can be used as a base project, as this will populate the relevant defconfigs

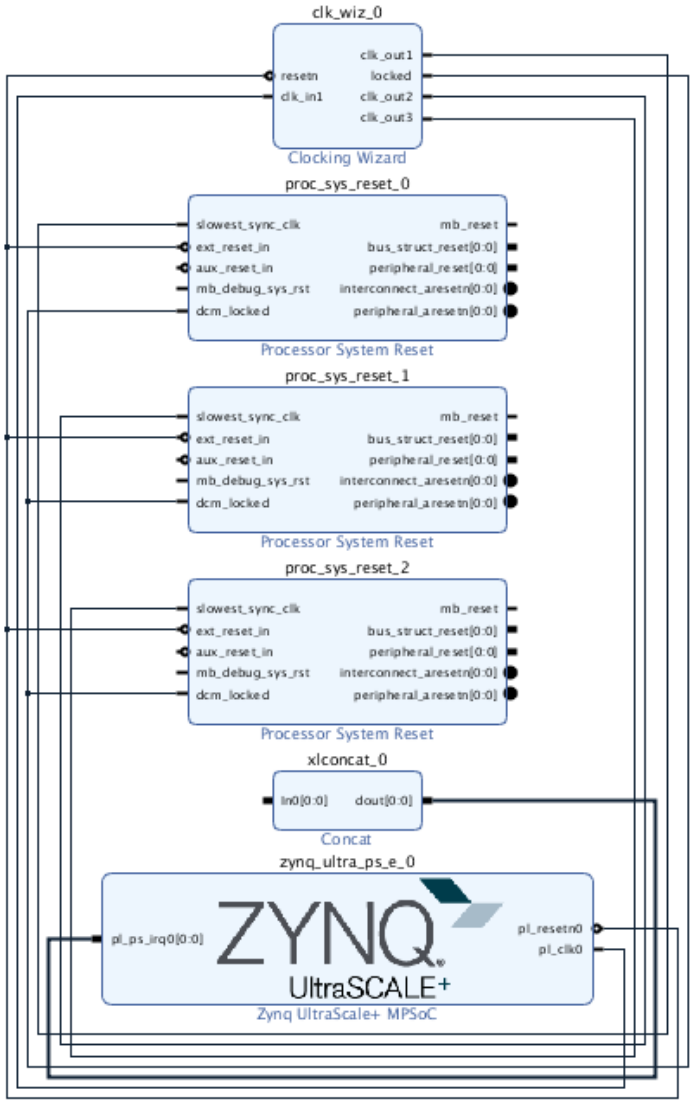
- cd <bsp petalinux project>
- rm -rf components/plnx_workspace
- petalinux-config -get-hw-description=<path to HDF>
- petalinux-config -c kernel
    - Device Drivers -> Generaic Driver Options -> Size in Mega Bytes(1024)
    - Device Drivers -> [*] Staging Drivers -> <*> Xilinx APF Acceleration driver -> [*] Xilinx APF DMA engines support
    - CPU Power Management -> CPU Idle -> [ ] CPU idle PM support
    - CPU Power Management -> CPU Frequency scaling -> [ ] CPU Frequency scaling
- petalinux-config -c rootfs
    - Filesystem Packages -> misc -> gcc-runtime -> [*] libstdc++
- petalinux-build
- cd images/linux
- petalinux-package --boot --fsbl zynqmp_fsbl.elf --uboot

If not using the BSP:

- petalinux-create -t project --template zynqMP --name ultra96_linux
- petalinux-config -get-hw-description=<path to HDF>
- The menu config will auto launch
    - DTG Settings -> (zcu100-revc) MACHINE_NAME
    - U-boot Configuration -> (Xilinx_zynqmp_zcu1000_revC_defconfig) u-boot config target
- petalinux-config -c kernel
    - Device Drivers -> Generaic Driver Options -> Size in Mega Bytes(1024)
    - Device Drivers -> [*] Staging Drivers -> <*> Xilinx APF Acceleration driver -> [*] Xilinx APF DMA engines support
    - CPU Power Management -> CPU Idle -> [ ] CPU idle PM support
    - CPU Power Management -> CPU Frequency scaling -> [ ] CPU Frequency scaling
- petalinux-config -c rootfs
    - Filesystem Packages -> misc -> gcc-runtime -> [*] libstdc++
- petalinux-build
- cd images/linux
- petalinux-package --boot --fsbl zynqmp_fsbl.elf --uboot

SDx Platform creation:

Tip: Users can take existing DSA files and open these in Vivado that can be used as a base project. There will be a <dsa name>_bd.tcl that can be used to create the BD with the PFM properties and this could be a good starting point for users creating a custom platform. The platform used in this demo is based on the HW project used in previous section:



Note: The UART 0 is disabled due to the fact that the 3 pin UART on the board is used.

Note: If using the TCL in the Petalinux section, then the PFM properties are already set. However, this is documented below for demo purposes. The PFM properties can be placed in a TCL file and be source from the TCL console on an opened Block Design in Vivado:

pfm.tcl

```
# Create PFM attributes
set_property PFM_NAME {vendor:lib:ultra96:1.0} [get_files [current_bd_design].bd]

set_property PFM.CLOCK { \
clk_out1 {id "1" is_default "true" proc_sys_reset "/proc_sys_reset_0"} \
clk_out2 {id "2" is_default "false" proc_sys_reset "/proc_sys_reset_1"} \
clk_out3 {id "3" is_default "false" proc_sys_reset "/proc_sys_reset_2"} \
} [get_bd_cells /clk_wiz_0]

set_property PFM.IRQ { \
In0 {} In1 {} In2 {} In3 {} In4 {} In5 {} In6 {} In7 {} \
} [get_bd_cells /xlconcat_0]

set_property PFM.AXI_PORT { \
M_AXI_HPM0_FPD {memport "M_AXI_GP" sptag "" memory ""} \
M_AXI_HPM1_FPD {memport "M_AXI_GP" sptag "" memory ""} \
M_AXI_HPM0_LPD {memport "M_AXI_GP" sptag "" memory ""} \
S_AXI_HP0_FPD {memport "S_AXI_HP" sptag "" memory ""} \
S_AXI_HP1_FPD {memport "S_AXI_HP" sptag "" memory ""} \
S_AXI_HP2_FPD {memport "S_AXI_HP" sptag "" memory ""} \
S_AXI_HP3_FPD {memport "S_AXI_HP" sptag "" memory ""} \
S_AXI_LPD {memport "MIG" sptag "" memory ""}\
} [get_bd_cells /zynq_ultra_ps_e_0]
```

This is then sourced from the TCL command line.

Preform the following tasks below to create the DSA and export to SDK:

- Generate Block Design
- Create HDL wrapper
- File -> Export -> Export Hardware
- write_dsa -force ultra96.dsa
- Launch SDK

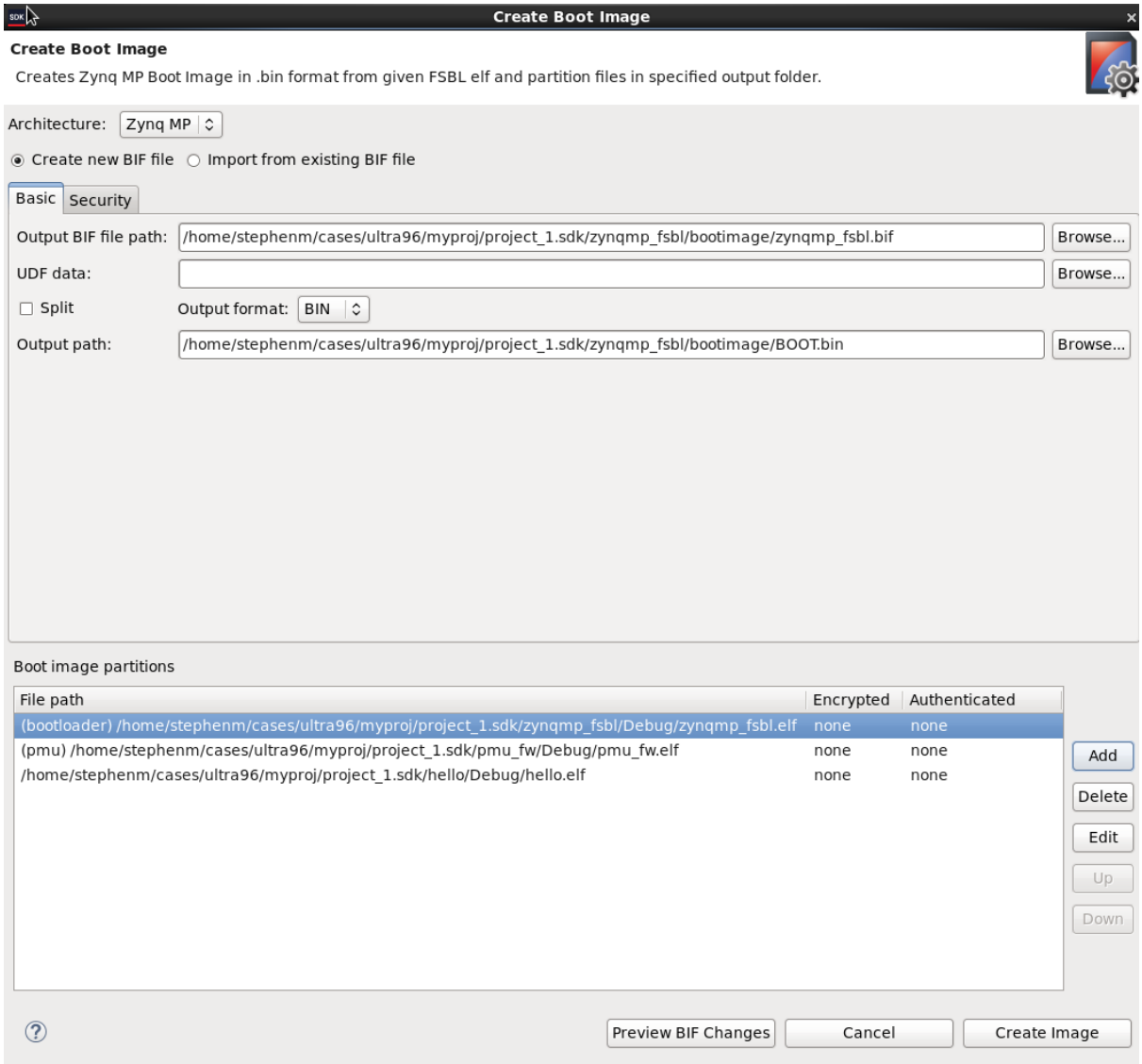
Platform Software file creation (Standalone)

The following files are needed for the Software Platform:

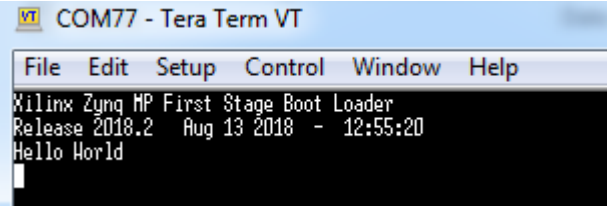
- FSBL
- PMUFW
- Linker Script
- BIF

Launch SDK, and use the templates to create the FSBL and the PMUFW. User can create a Hello world application to generate the Linker script (as this is placed in DDR).

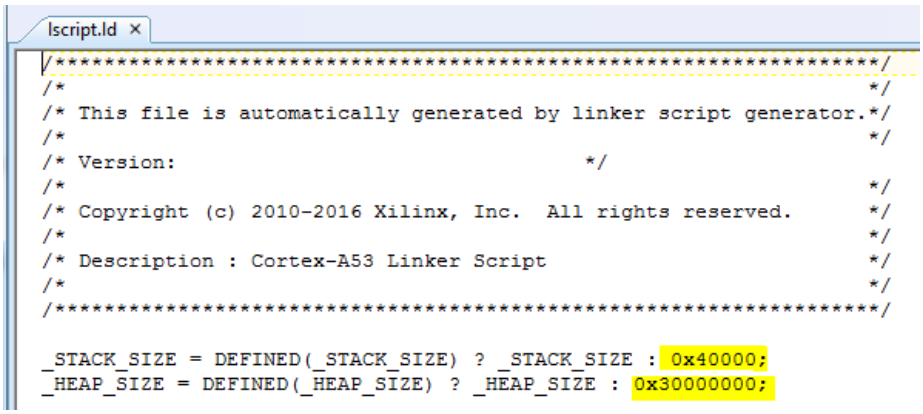
As a sanity check, users can create the bootable image in SDK and test on the board. To do this, right click on the fsbl application in Project Explorer, and select Create Boot Image



Test this on the HW:



The Heap and Stack in the linker should be updated as shown below. Users can read the Software Developers Guide, here for more info on create SDK projects.



```
lscript.ld
/*******************************************************************/
/*                                                                 */
/* This file is automatically generated by linker script generator.*/
/*                                                                 */
/* Version:                                    */
/*                                                                 */
/* Copyright (c) 2010-2016 Xilinx, Inc.  All rights reserved.      */
/*                                                                 */
/* Description : Cortex-A53 Linker Script                          */
/*                                                                 */
/*******************************************************************/

_STACK_SIZE = DEFINED(_STACK_SIZE) ? _STACK_SIZE : 0x40000;
_HEAP_SIZE = DEFINED(_HEAP_SIZE) ? _HEAP_SIZE : 0x30000000;
```

There is a standalone BIF template shown below:

the_ROM_image:

{

[fsbl_config] a53_x64

[bootloader] <zynqmp_fsbl.elf>

[pmufw_image] <pmu_fw.elf>

[destination_device=pl] <bitstream>

<elf>

}

The Linux BIF is shown below:

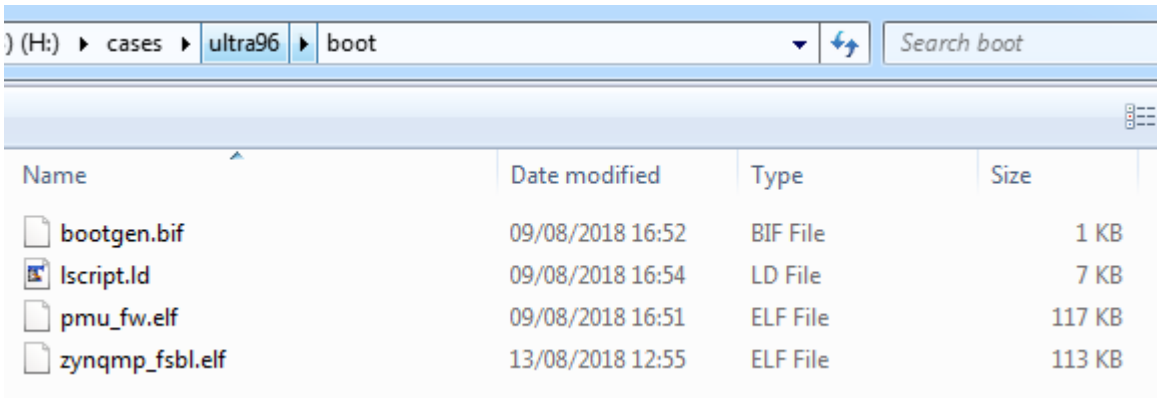
```
the_ROM_image:
{
 [fsbl_config] a53_x64
 [bootloader]<zynqmp_fsbl.elf>
 [pmufw_image]<pmufw.elf>
 [destination_device=pl] <bitstream>
 [destination_cpu=a53-0, exception_level=el-3, trustzone] <bl31.elf>
 [destination_cpu=a53-0, exception_level=el-2] <u-boot.elf>
}
```

These should all be contained in a boot folder.

See chapter 16 in the Software Developers Guide for more info here

The files below should be copied into a boot folder:

- FSBL
- PMU Firmware
- Linker script
- BIF (to create bootable image)



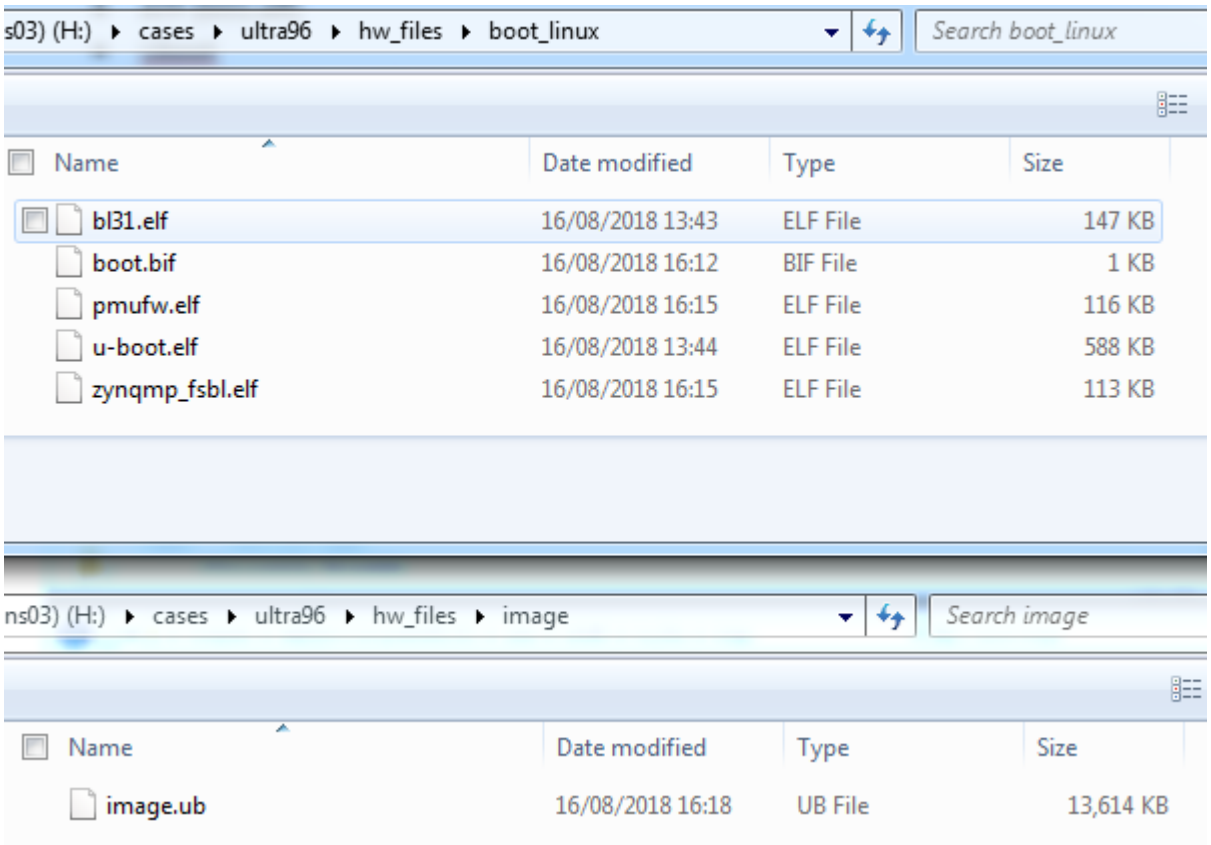
Platform Software file creation (Linux)

The files below should be copied into a boot_linux folder

- FSBL
- PMU Firmware
- ATF (bl31.elf)
- Uboot
- BIF (to create bootable image)

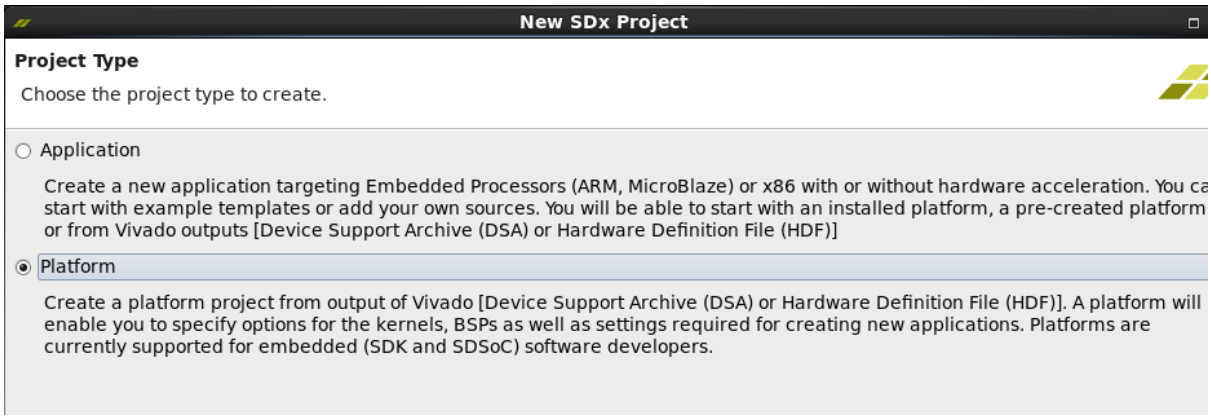
The file(s) below should be copied into an images folder

- Image.ub



Creating the SDx Platform:

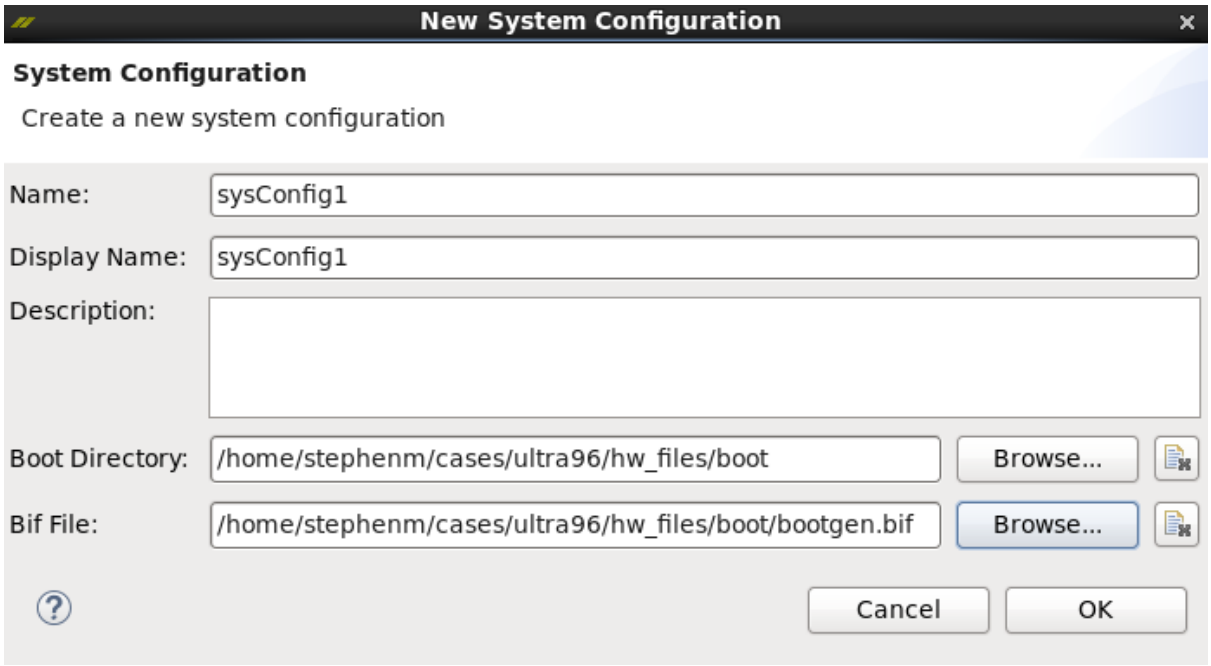
Launch SDSoC 2018.2. File -> New -> SDx Project:



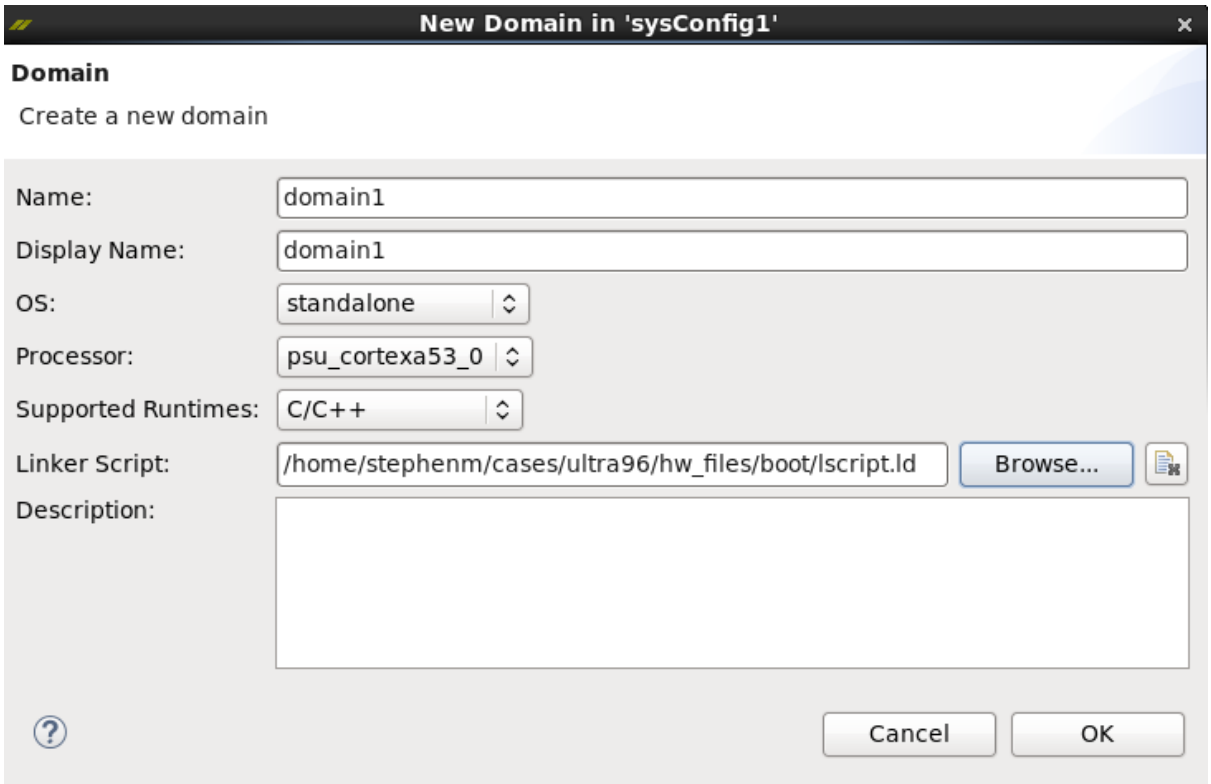
Navigate to the DSA file created in Vivado

Standalone SW Platform:

Select Define System Configuration, and populate with the boot folder and the standalone BIF file:

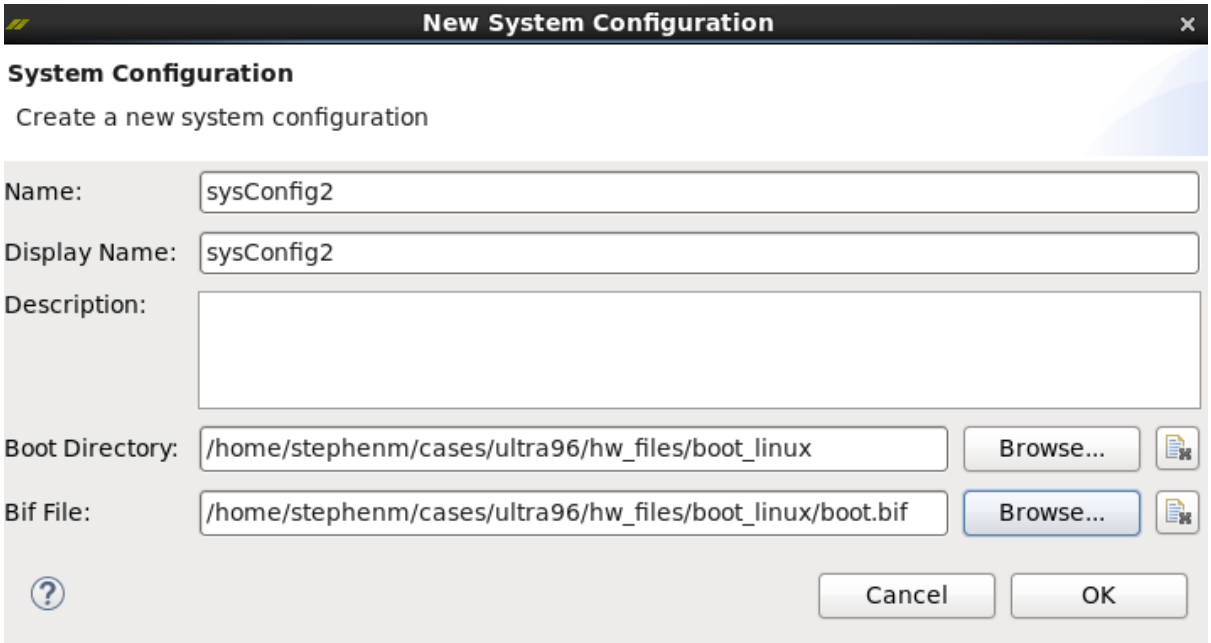


Create a new standalone Domain and populate with the Linker:

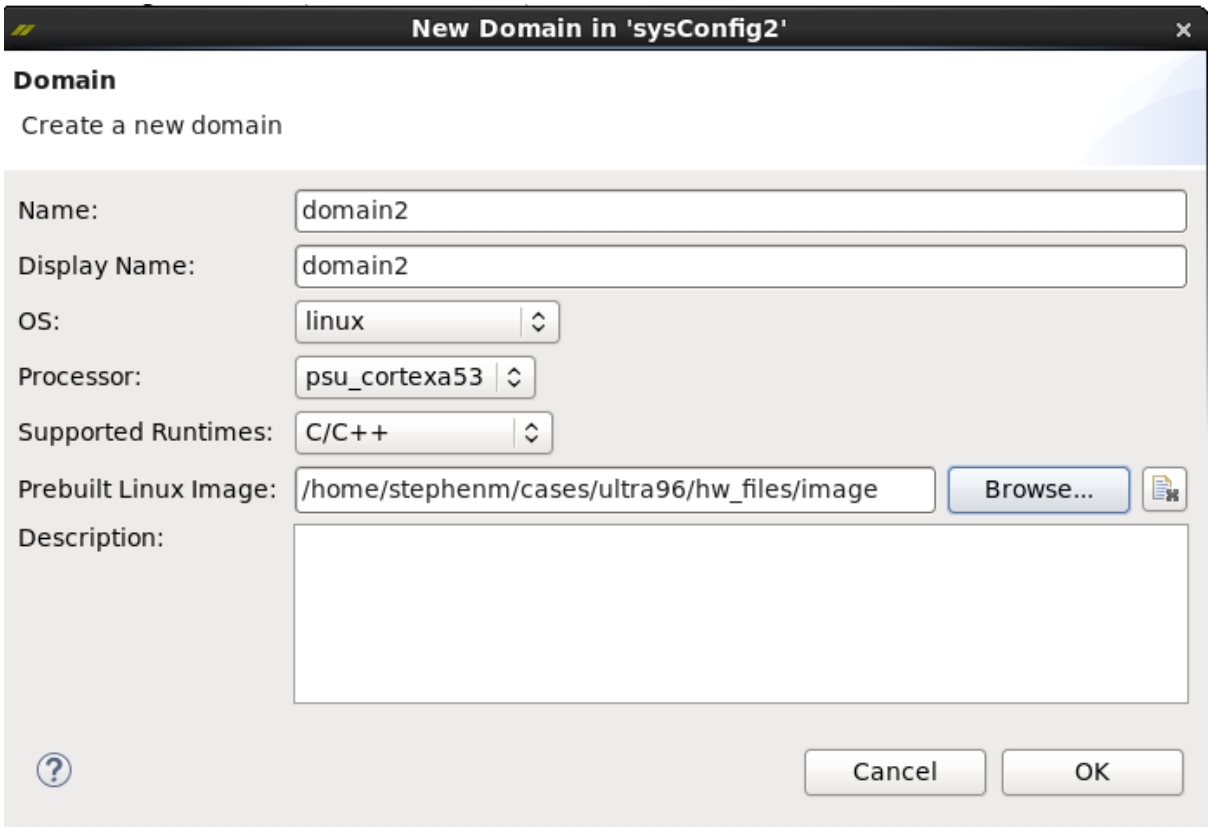


Linux SW Platform:

Select Define System Configuration, and populate this with the boot_linux and the linux BIF file:



Make sure that sysConfig2 is highlighted and select Add Processor Group Domain

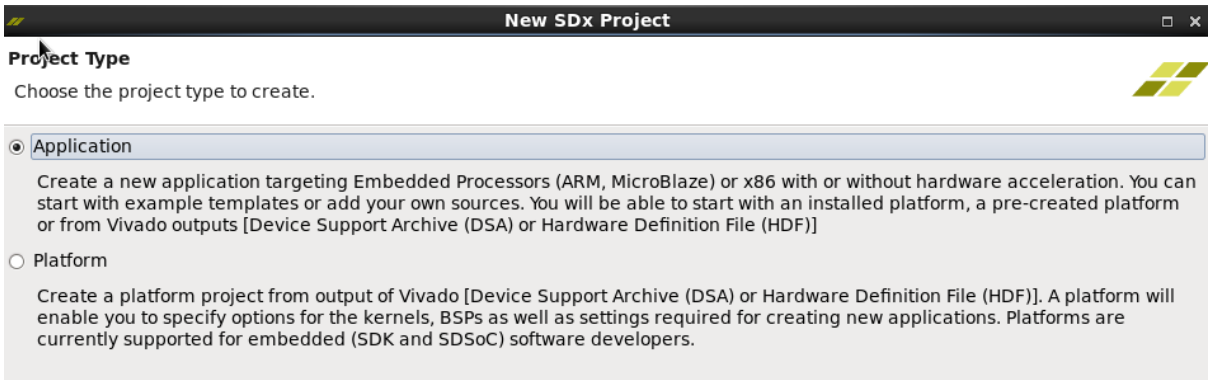


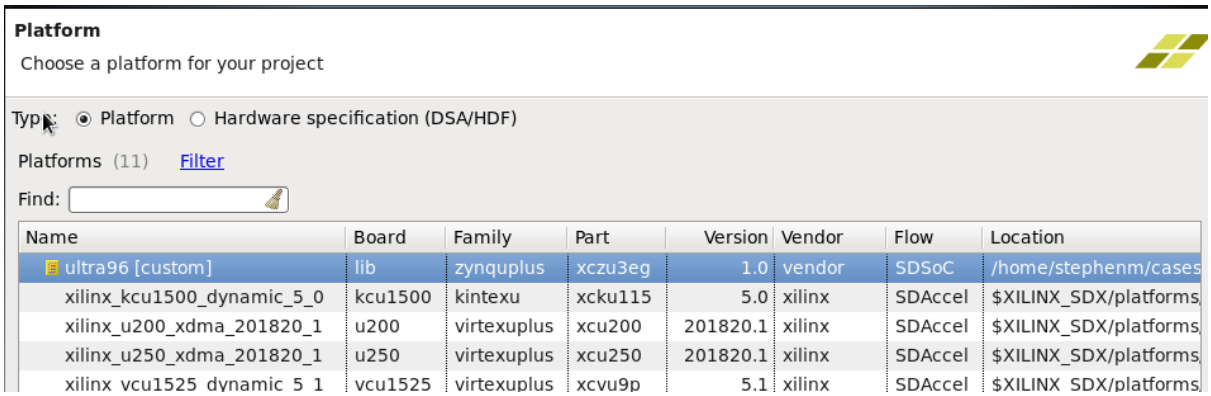
Generate Platform

Add to Custom Repositories

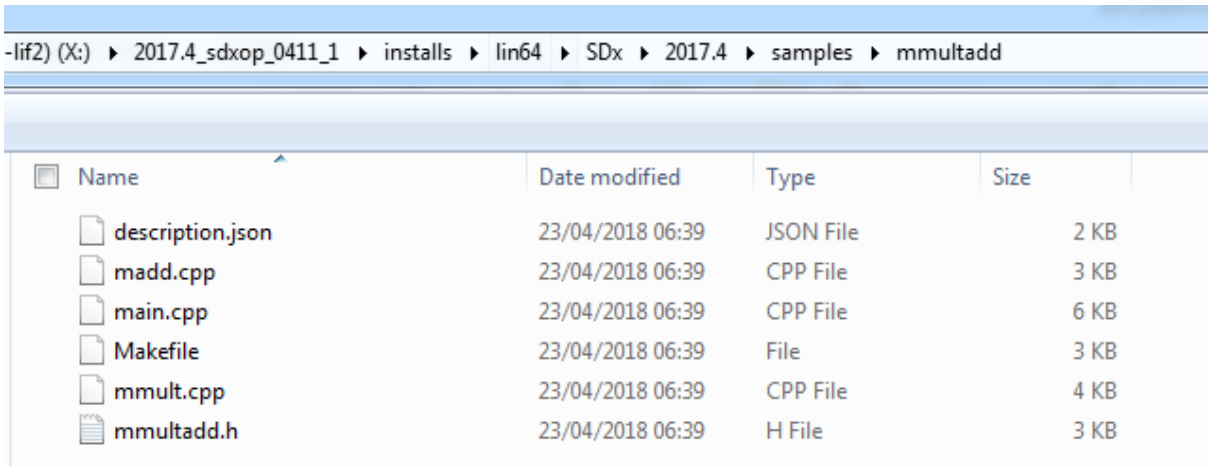
Create a new SDSoC Project:

File -> New -> SDx Project:

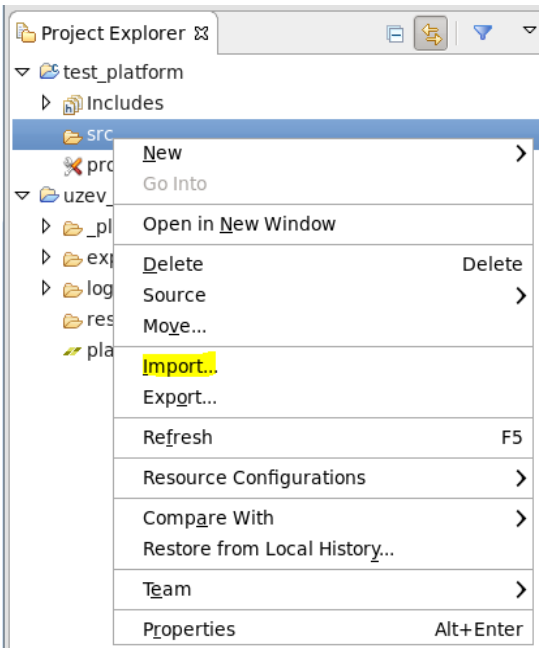




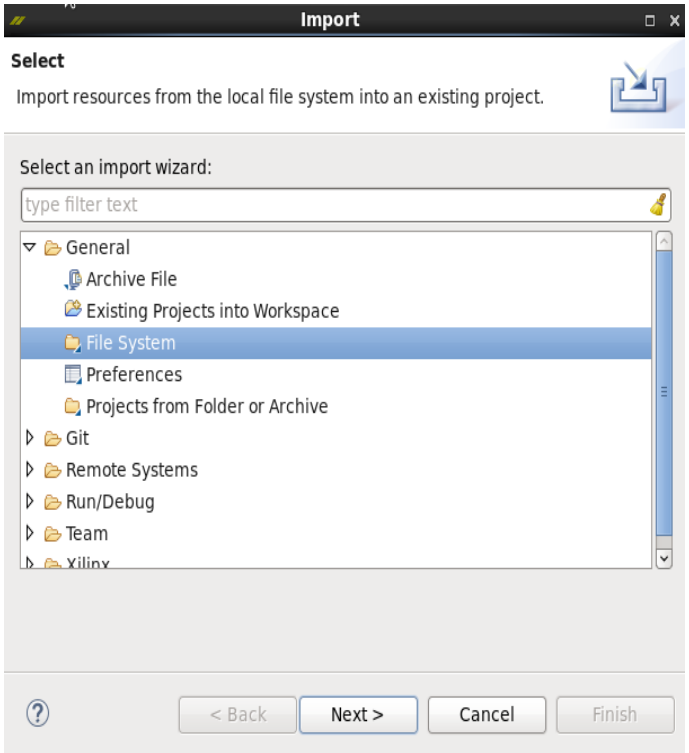
Select Next and select the domain (either standalone or Linux depending on how the platform was setup in previous steps) and Finish. In this case the standalone was used. There will be an Empty Application created. Users can import any of the sample code delivered with SDSoC here. For example, the mmultadd sample code:



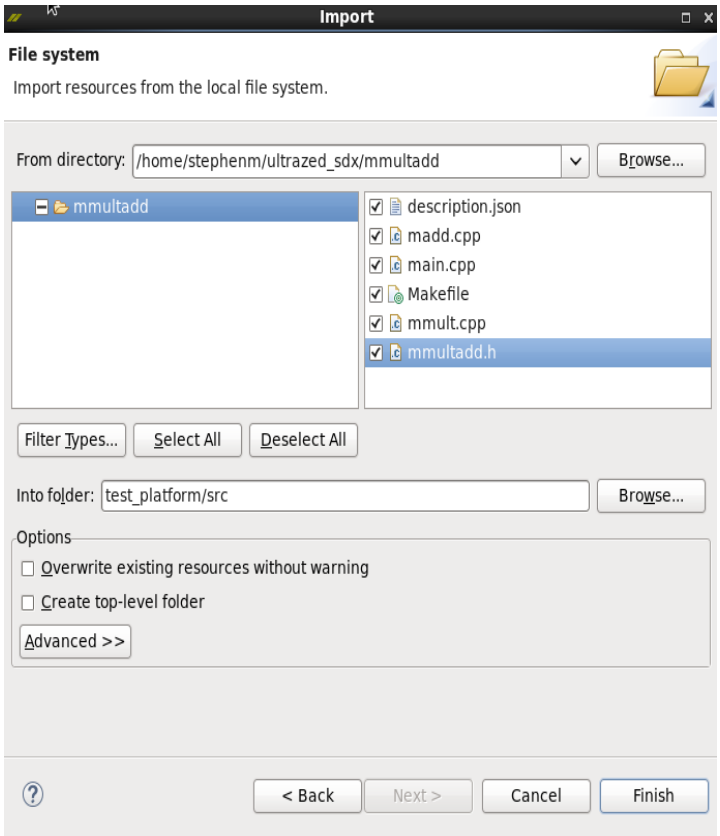
To add this, right click on the *src* directory in the newly create application, and Import:



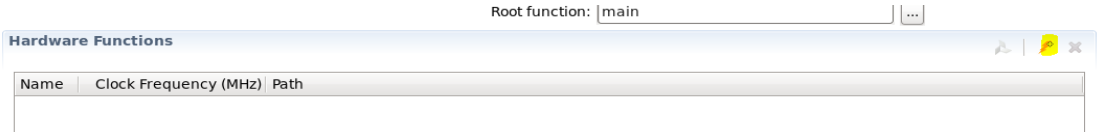
File System:



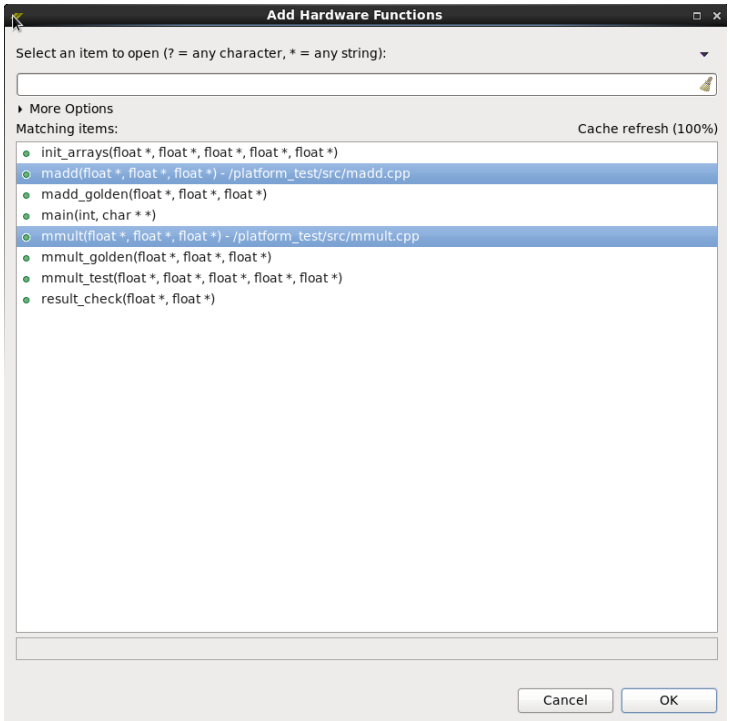
Select all the files:

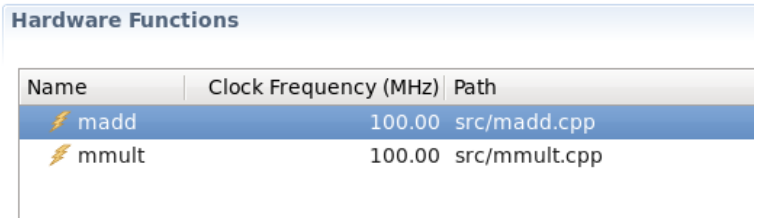


Choose the function to accelerate:

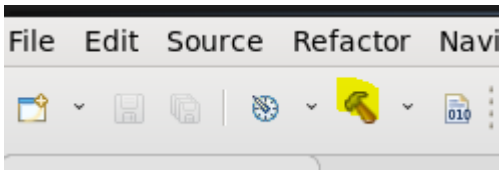


Here, the *madd* and *mmult* functions are selected.



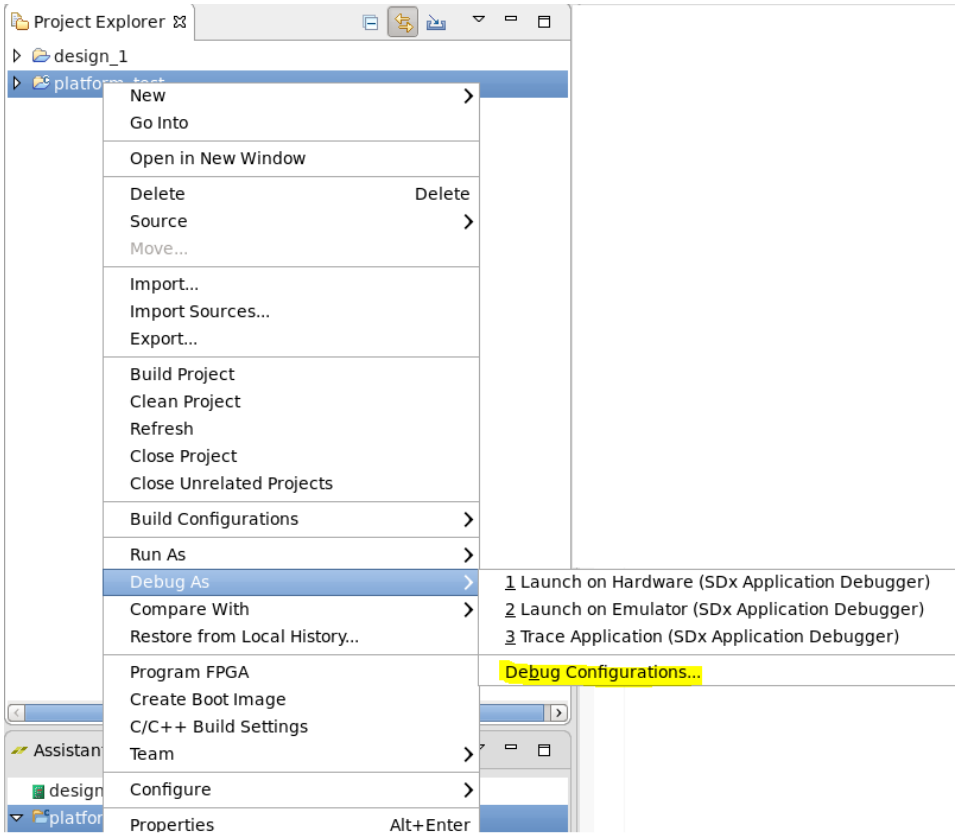


Compile the application:



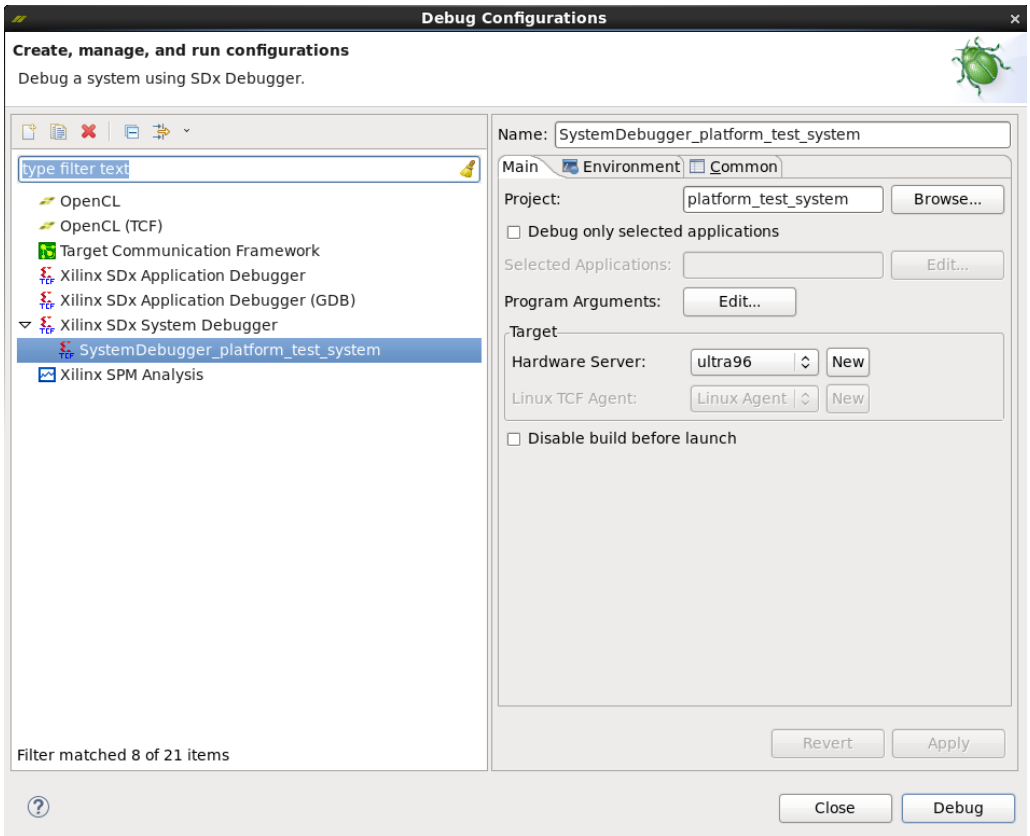
This will take a while to complete as the SDSoC will create HLS IP from the function C code. It will create a datamover based upon the function input/output. It will then build the HW, and the ELF. It will create a bootable image too.

Once, this is complete. User has the option to test on the SD card, or test via the debugger. To test via the debugger, right click on the application in the Project Explorer view, and select Debug As – Debug Configurations:



Debugging the Application in SDx:

Create a new Xilinx SDx System Debugger:



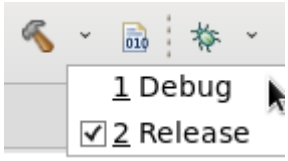
Note: Here I am connected remotely via the Hardware Server. If users are connected locally. Then this would be left as default (Local). Select Debug to launch the debug perspective.

Note: If users can view the SDK log to see what the tools are doing in the background:

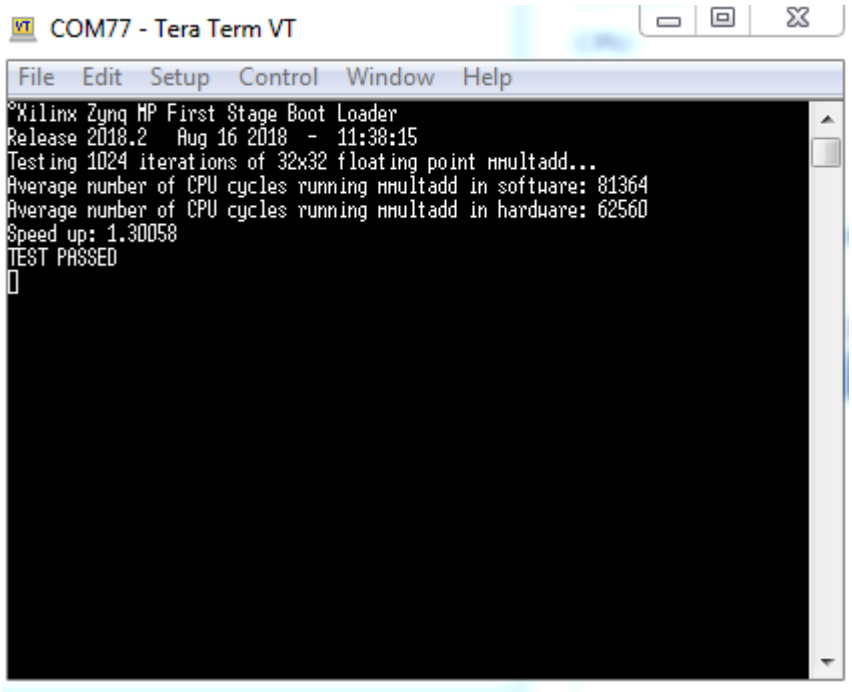
```
11:58:12 INFO   : Jtag cable 'Platform Cable USB II 000015de651301' is selected.
11:58:12 INFO   : 'jtag frequency' command is executed.
11:58:12 INFO   : Sourcing of '/proj/xbuilds/2018.2_0808_1854/installs/lin64/SDK/2018.2/scr
11:58:12 INFO   : Context for 'APU' is selected.
11:58:14 INFO   : System reset is completed.
11:58:17 INFO   : 'after 3000' command is executed.
11:58:17 INFO   : 'targets -set -filter {jtag_cable_name =~ "Platform Cable USB II 000015de
11:58:37 INFO   : FPGA configured successfully with bitstream "/home/stephenm/cases/ultra96
11:58:37 INFO   : Context for 'APU' is selected.
11:58:38 INFO   : Hardware design information is loaded from '/home/stephenm/cases/ultra96/
11:58:38 INFO   : 'configparams force-mem-access 1' command is executed.
11:58:38 INFO   : Context for 'APU' is selected.
11:58:38 INFO   : Sourcing of '/home/stephenm/cases/ultra96/sdx_ws/platform_test/Debug/_sds
11:58:52 INFO   : 'psu_init' command is executed.
11:58:53 INFO   : 'after 1000' command is executed.
11:58:53 INFO   : 'psu_ps_pl_isolation_removal' command is executed.
11:58:54 INFO   : 'after 1000' command is executed.
11:58:55 INFO   : 'psu_ps_pl_reset_config' command is executed.
11:58:55 INFO   : 'catch {psu_protection}' command is executed.
11:58:55 INFO   : Context for processor 'psu_cortexa53_0' is selected.
11:58:56 INFO   : Processor reset is completed for 'psu_cortexa53_0'.
12:00:10 INFO   : The application '/home/stephenm/cases/ultra96/sdx_ws/platform_test/Debug/
12:00:10 INFO   : 'configparams force-mem-access 0' command is executed.
12:00:10 INFO   : ----------------XSDB Script----------------
```

Once the debug perspective launches the user can debug as normal. User should see the following on the serial port

To create a Bootable image, then change to Release and rebuild:



Place the files from the Release/sd_card folder onto the SD card, and boot:



Testing Linux:

Create a new application. However, select the Linux config: