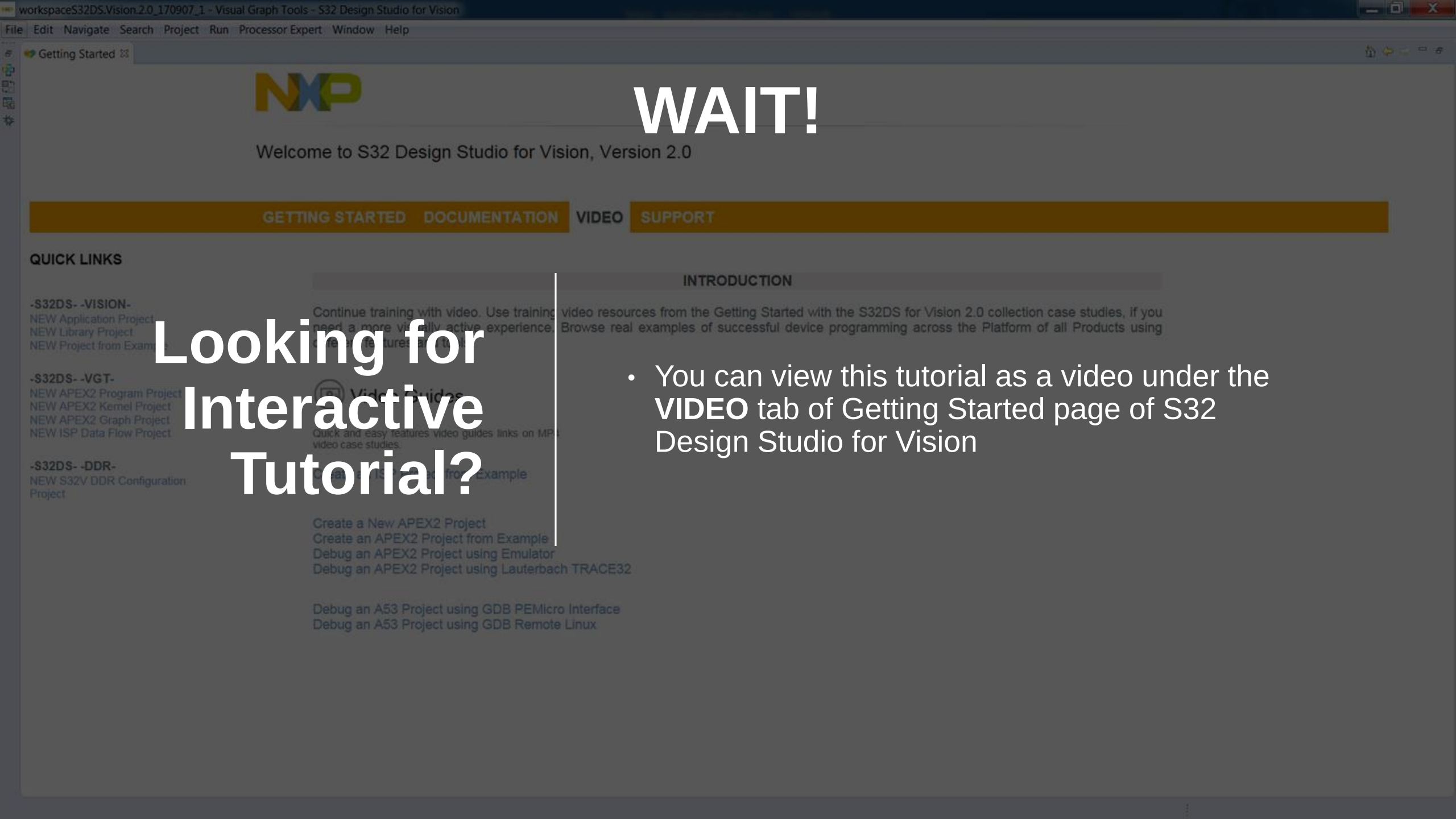
TUTORIAL: S32DS APEX VISUAL GRAPH TOOL

S32 DESIGN STUDIO FOR VISION 2018.R1 or higher

WAIT!

Looking for Interactive Tutorial?

- You can view this tutorial as a video under the **VIDEO** tab of Getting Started page of S32 Design Studio for Vision



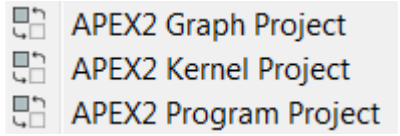
Prerequisite for tutorial

- Knowing the S32V234 SoC
- Have an understanding of the APEX architecture and APEX Core Framework(ACF)
  - Refer **UG-10267-03-14-ACF_User_Guide.pdf** to learn about ACF
    - Path: *s32ds_installation_directory*\S32DS\s32v234_sdk\docs\apex\acf
- Be familiar with the Vision SDK software

Agenda

- Tutorial overview
- Make an APEX Graph (project)
- Make an APEX Program (project)
- Make a Linux application project

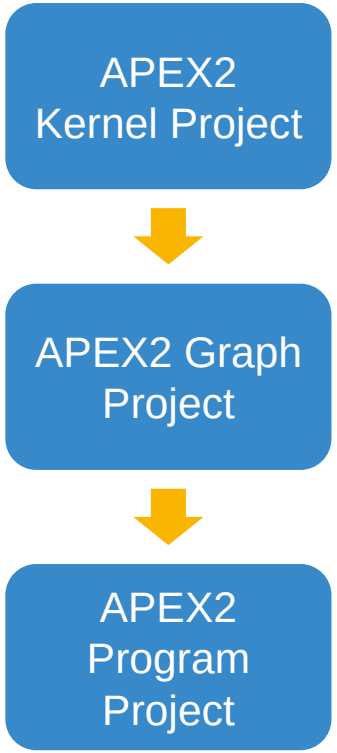
WHAT ARE 3 OPTIONS AVAILABLE IN APEX GRAPH TOOL?



- APEX2 Kernel Project:
  - Kernel project is useful to define user function that works as one of the building blocks of vision pipeline

- APEX2 Graph Project:
  - Here, user can define the vision pipeline using multiple kernels
    - Vision SDK provides a broad library of built-in kernels
    - Moreover, user can build custom kernels using APEX2 Kernel Project.
  - The graph minimizes the data transfer between host and APEX, by launching multiple kernels with single data transfer. In short takes advantages of pipelining and data localization.

- APEX2 Program Project:
  - This project actually generates a source code for an APEX application
  - Once vision pipeline is ready, user can
    - Use multiple graphs to create complete APEX program flow (useful in case of dependencies between graphs)
    - Map different graphs to different APEX engines (useful if user want to run multiple graphs in parallel)
    - Choose from different image buffers (useful to transfer data between host and APEX)



Tutorial Overview:

1. We will make an APEX graph using *APEX2 Graph Project* option
   - Just to make it simple we will use the kernels available in Vision SDK
     - Vision SDK provides many built-in Kernels readily available for user development.
   - User may use the *APEX2 Kernel Project* option to create a brand new kernel as well
2. Moving forward, we will use the graph built above and make an *APEX2 program project*
   - As described previously, using the program project we will specify the image buffer type, select APEX engine and generate the source code for APEX engines.
3. Lastly, we will use this source code into our Linux application program to accelerate the performance using APEX engines.

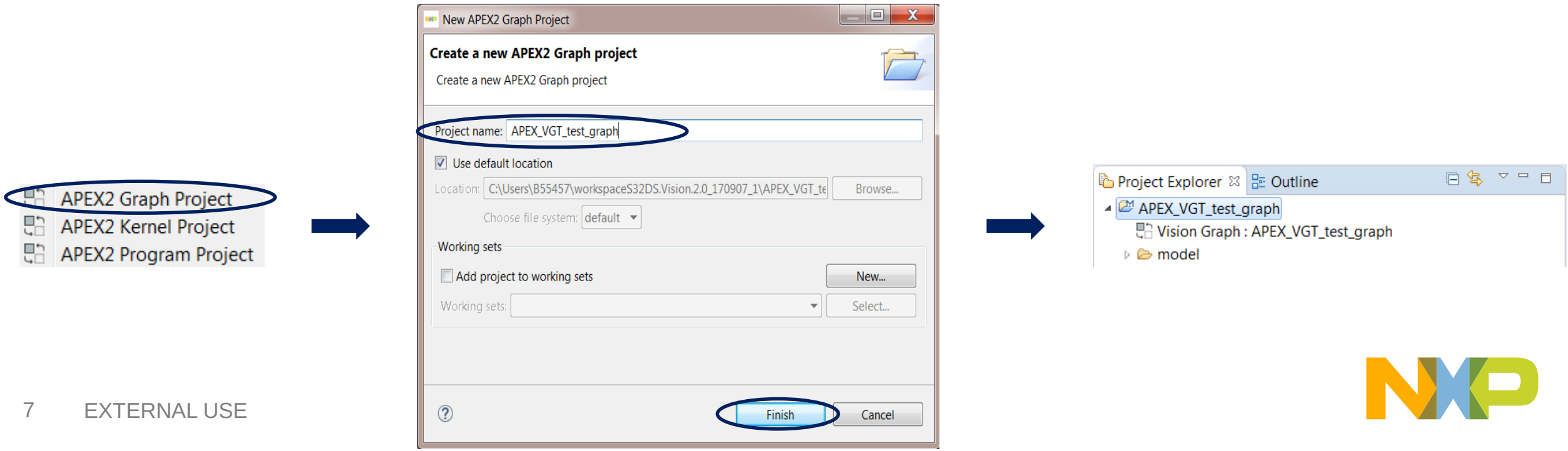
Complete application will take a .png image, upscale and downscale it using APEX engines and return processed images

MAKE AN APEX GRAPH

First of all we will make an APEX graph using Vision SDK kernels

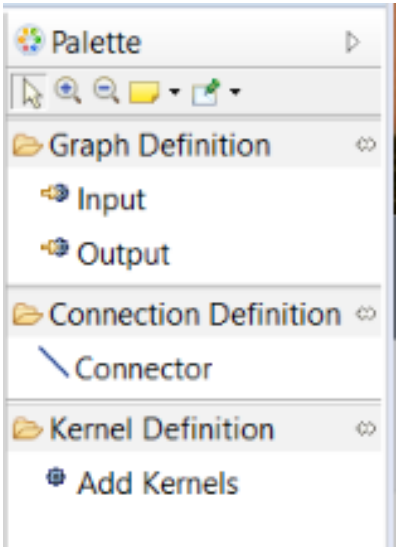
Make an APEX2 Program Project

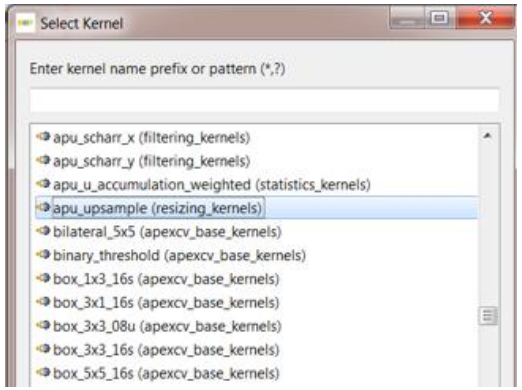
- We will make a simple graph that
  - Grabs an image >> Up-scales and down-scales it by factor of 2 >> Returns processed images
- Make a new **APEX2 graph project** named : **APEX_VGT_test_graph**

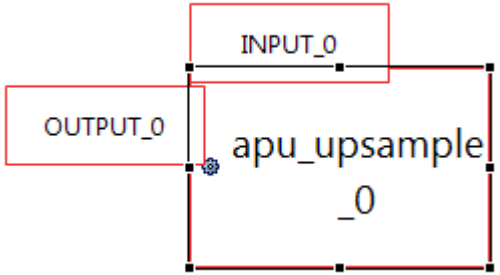


Make a graph

- Let’s start building a graph…
- From the **Palette** window (right side of the S32DS window):
  - Select **Add Kernels** block
  - Click in the workspace to drop the block and it will ask to select kernel for that block
  - Select built-in kernel: **apu_upsample (resizing kernels)**

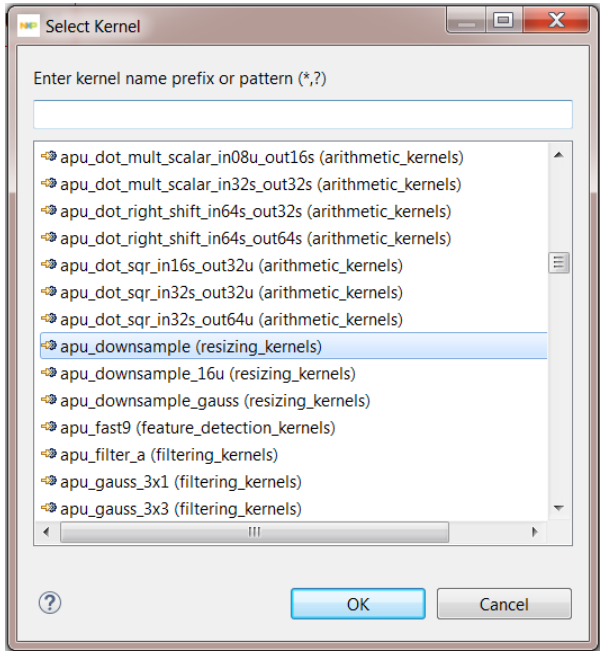


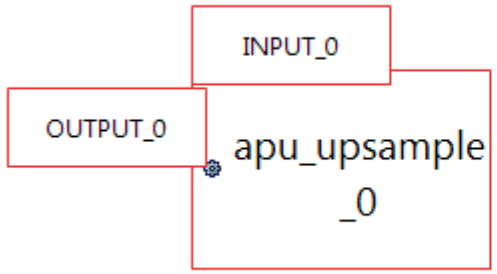


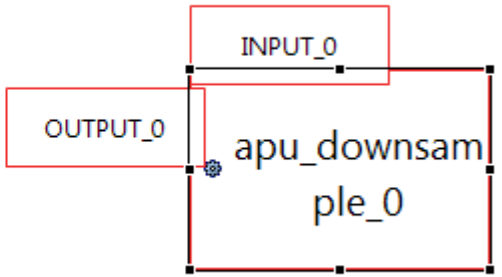


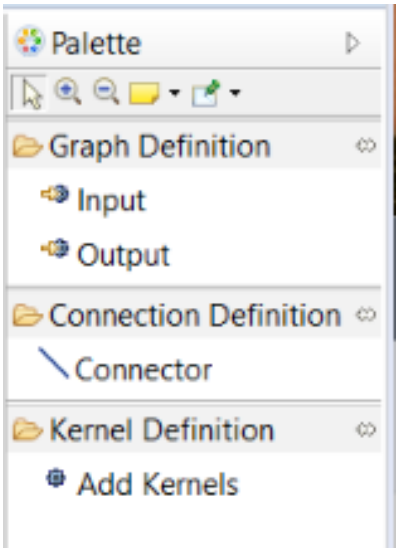
Make a graph

- Similarly, create a block for kernel: **apu_downsample (resizing_kernels)**



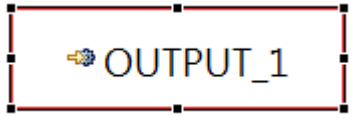


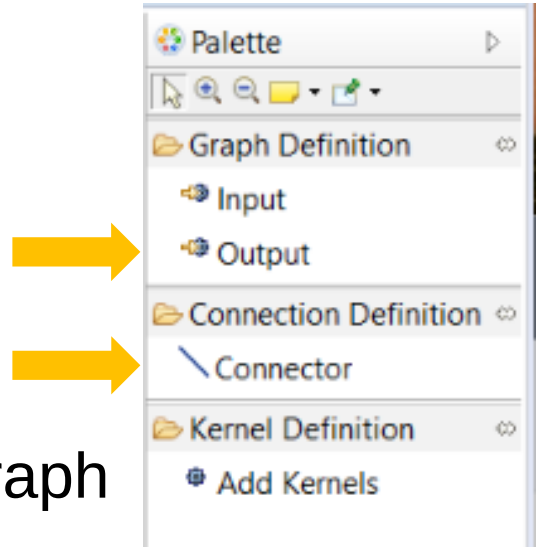




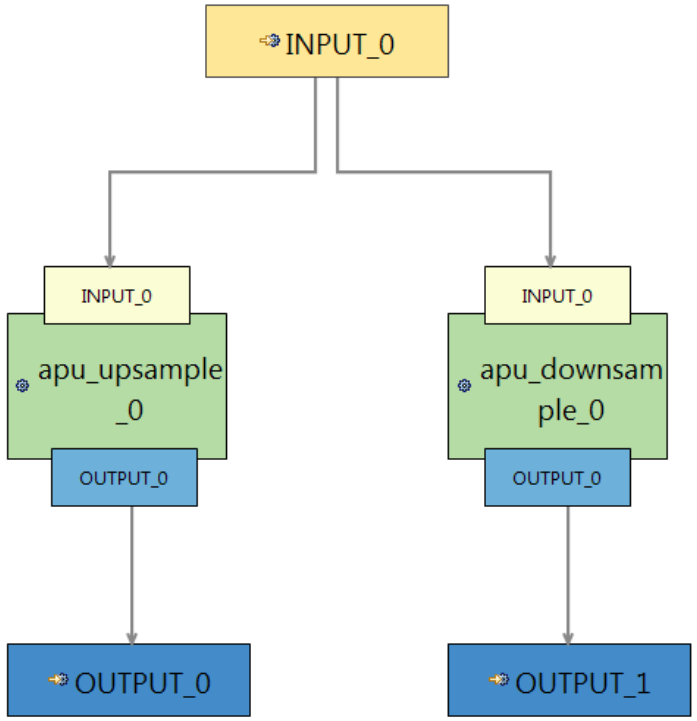
Make a graph

- Select **Output** block and include it into the graph





- Using **Connector**, connect graph blocks to create the following graph



Configure the Block Properties

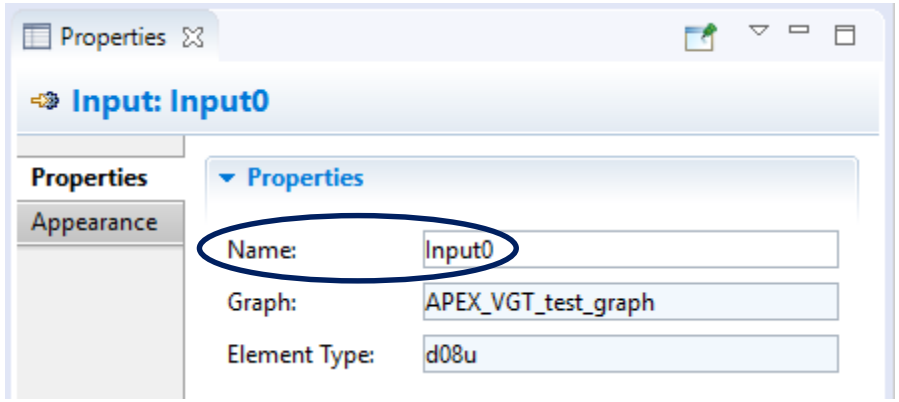
- You do **not** need to configure any properties for the graph!

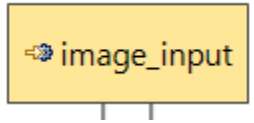
- APEX Core Framework (ACF) will take care of this!
  - e.g. ACF extracts Image size description from image buffer automatically

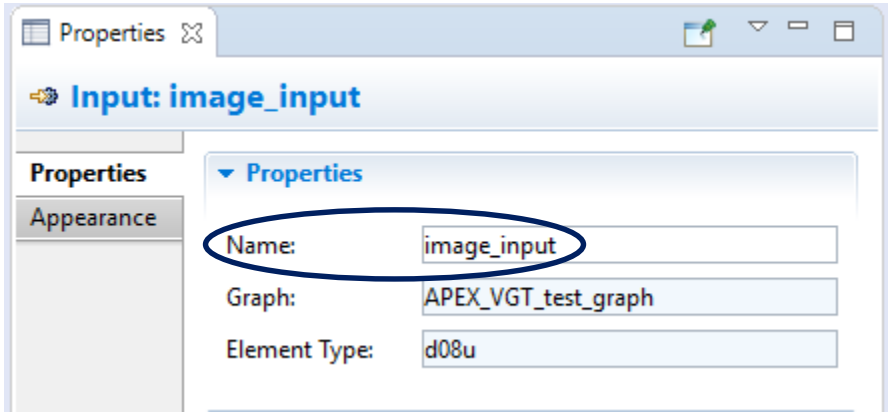
- It is a generic graph and can be used in any application without any application specific modification such as image size description.

Change the Block Name

- For better readability, user may want to change the block names
- Select a **Block** from the graph and look at the **Properties** window (on your left)
- Change the **Name** here, try changing the INPUT block

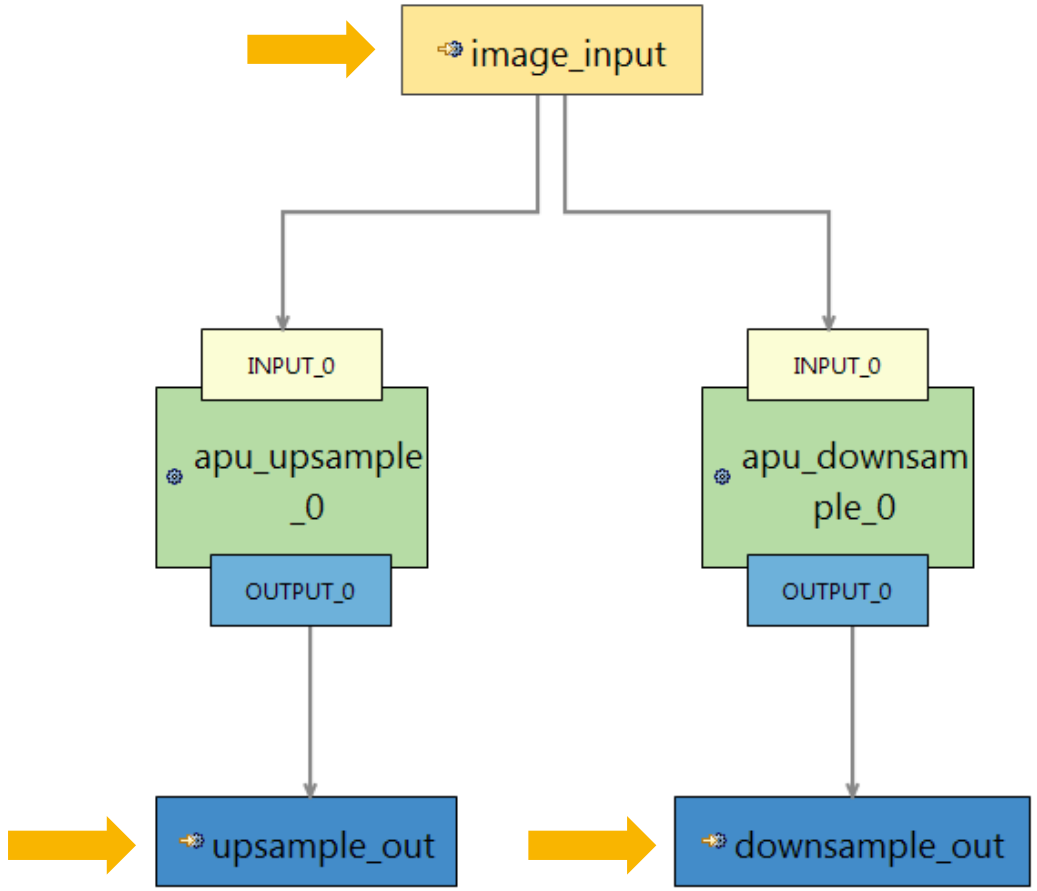






Change the Block Name

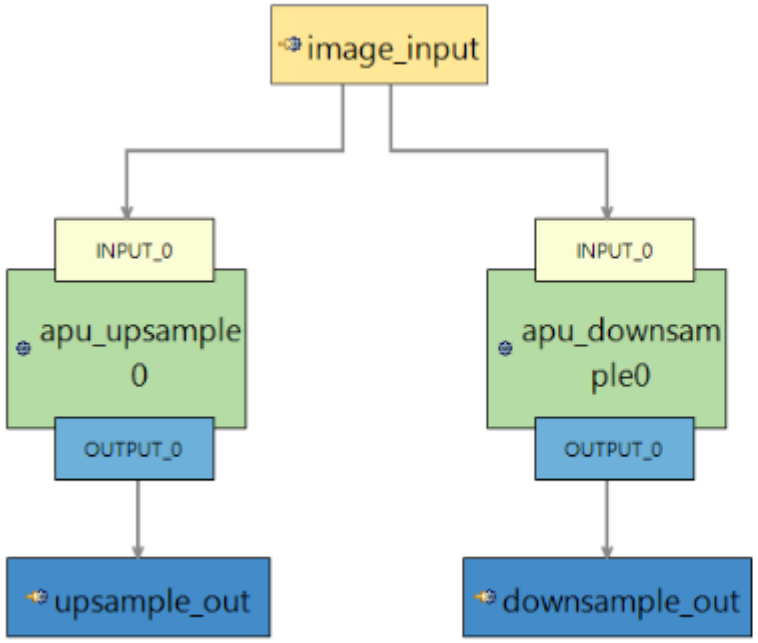
- Let’s do the same for both OUTPUT blocks
- Now graph should look like this…

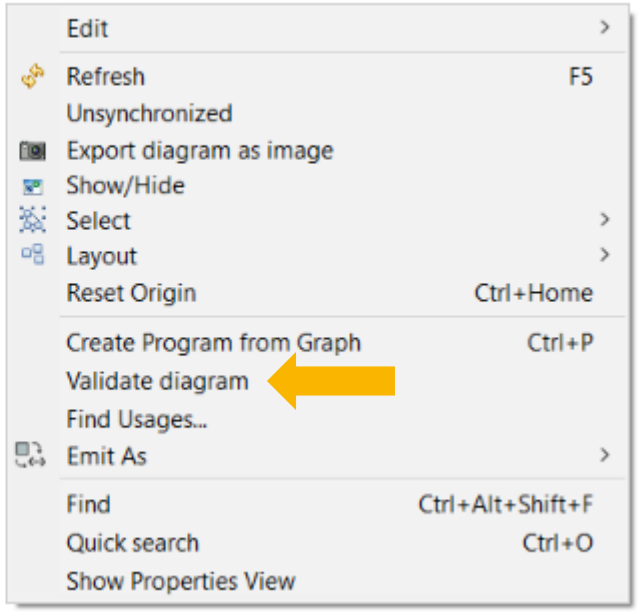


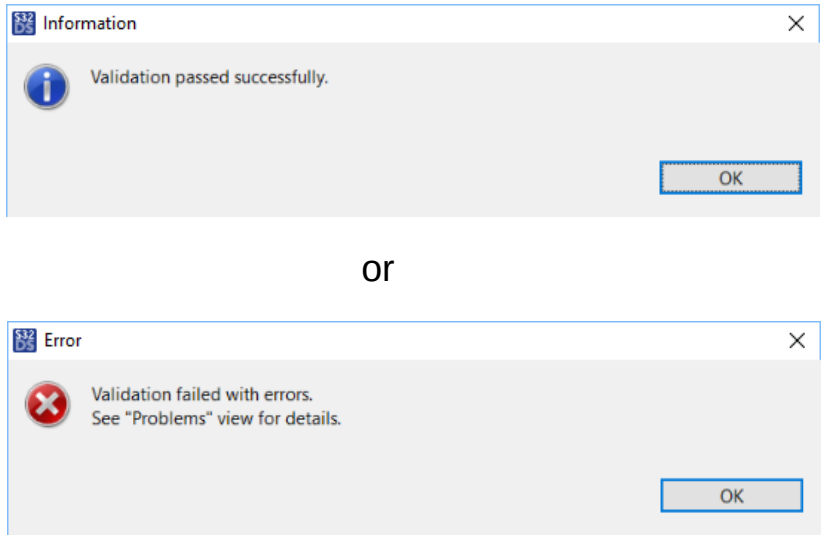
Graph is now READY!

Validate graph for correctness

- **Save** the graph
- **Right Click** anywhere in the white part of the graph
- **Validate** graph
  - You will see a pop-up window showing status of validation.

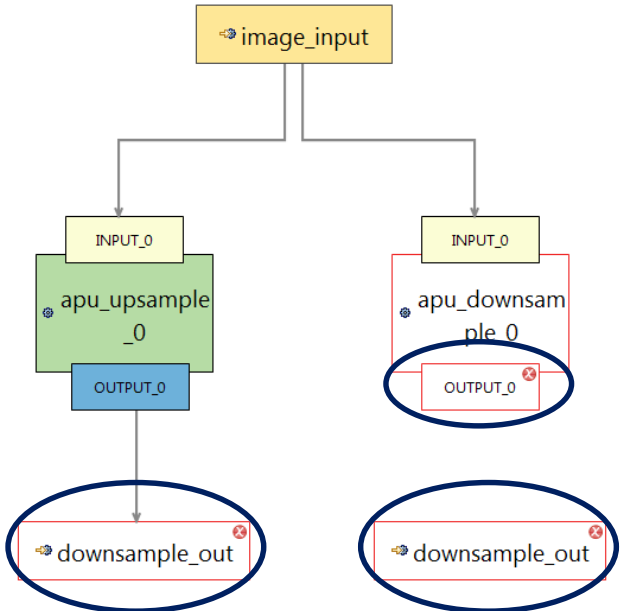






Validation Error

- Error will be indicated by red cross on the block and description can be seen in the **Problems view**
- Since there are no parameters to configure, there are only two type of errors
  - Missing connections
  - Duplicate names
- Block color will remain white in case of errors
- **Correct** the error, **Save** the graph and try to **Validate** again



Problems | Console

4 errors, 0 warnings, 0 others

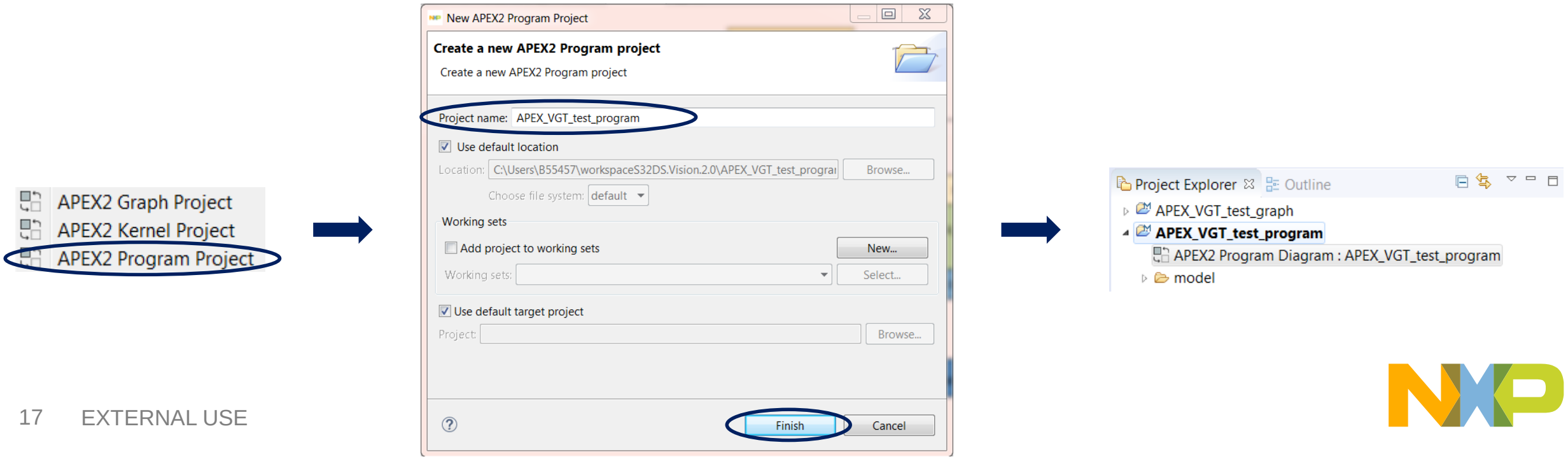
Description	Resource	Path	Location	Type
Errors (4 items)				
The 'Duplicate Input/Output name "downsample_out" in Graph "APEX_VGT_test_graph"' constraint is violate	APEX_VGT_test...	/APEX_VGT_test_grap...	APEX_VGT_test...	Sirius diagram editor Plugin problems
The 'Duplicate Input/Output name "downsample_out" in Graph "APEX_VGT_test_graph"' constraint is violate	APEX_VGT_test...	/APEX_VGT_test_grap...	APEX_VGT_test...	Sirius diagram editor Plugin problems
The 'Must have a consumer' constraint is violated on 'APEX_VGT_test_graph::apu_downsample_0::OUTPUT_0'	APEX_VGT_test...	/APEX_VGT_test_grap...	APEX_VGT_test...	Sirius diagram editor Plugin problems
The 'Must have a producer' constraint is violated on 'APEX_VGT_test_graph::downsample_out'	APEX_VGT_test...	/APEX_VGT_test_grap...	APEX_VGT_test...	Sirius diagram editor Plugin problems

MAKE AN APEX PROGRAM

Once APEX graph is ready, we will use this graph to make an APEX program. Here, we will map this graph to one of the APEX engines, define the image buffers and generate source code.

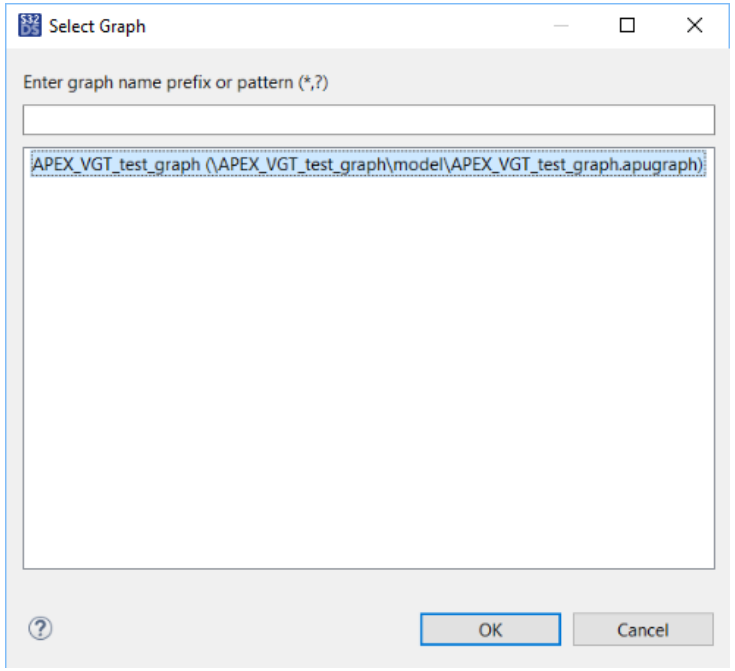
Make an APEX2 Program Project

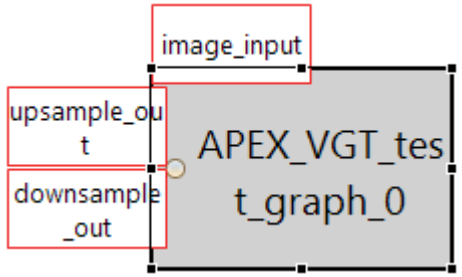
- We will make a program project and specify
  - graph we want to use and its mapping on specific APEX engine
  - image buffer type
- Make a new **APEX2 Program project** named: **APEX_VGT_test_program**

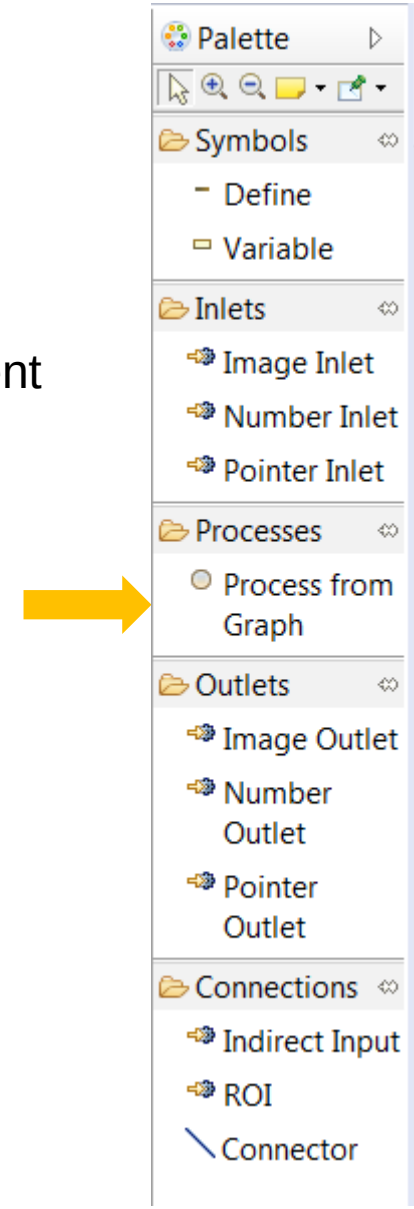


Make an APEX program

- From the **Palette** window (right side of the S32DS window):
  - Select **Process from Graph** block
  - Click in the workspace to drop the block and it will ask to select graph available in the current workspace
  - Select **“APEX_VGT_test_graph”** that we just created

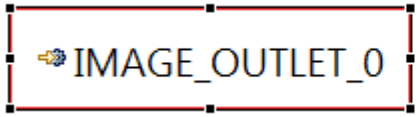


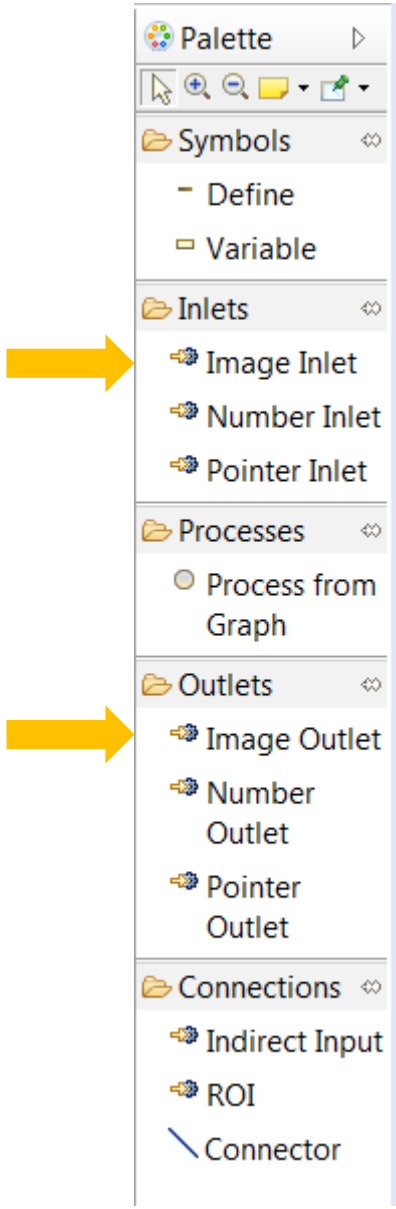




Make a program graph

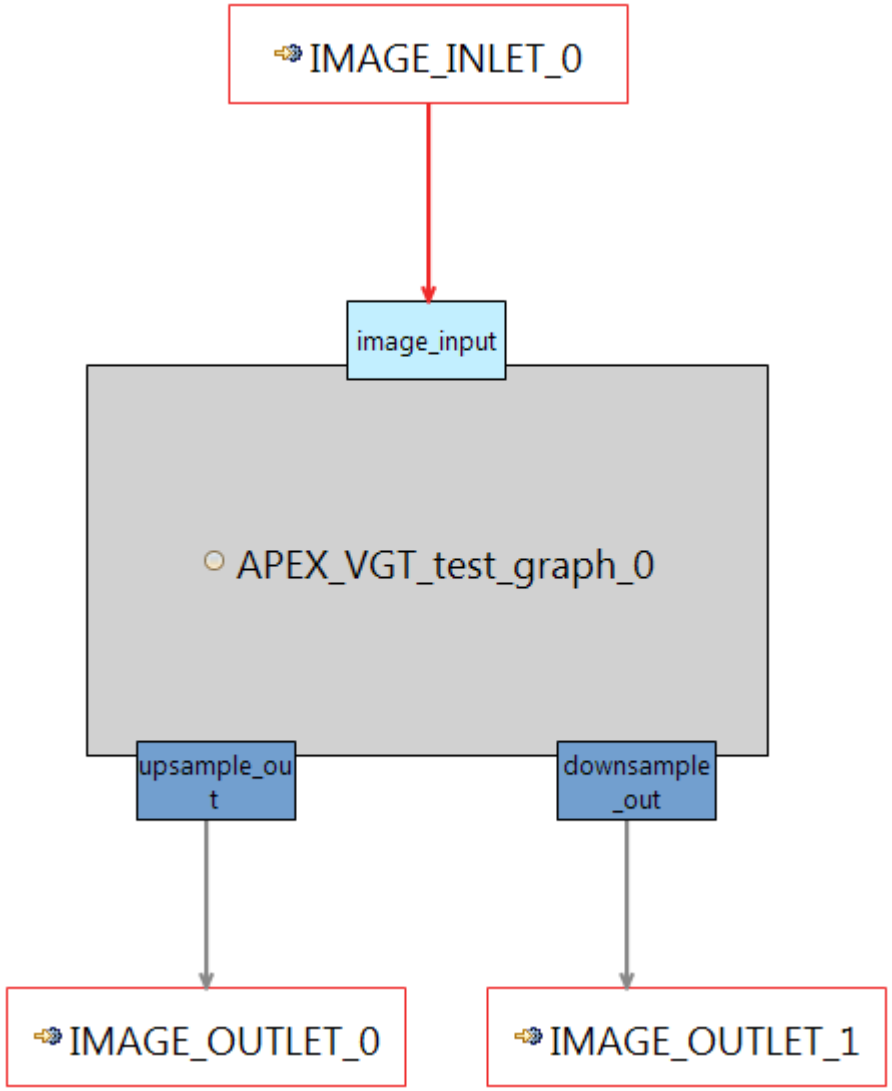
- We will use a .png image for our application
- So, our input and output will be of image types
- Select and create one **Image Inlet**
- Select and create two **Image Outlet**

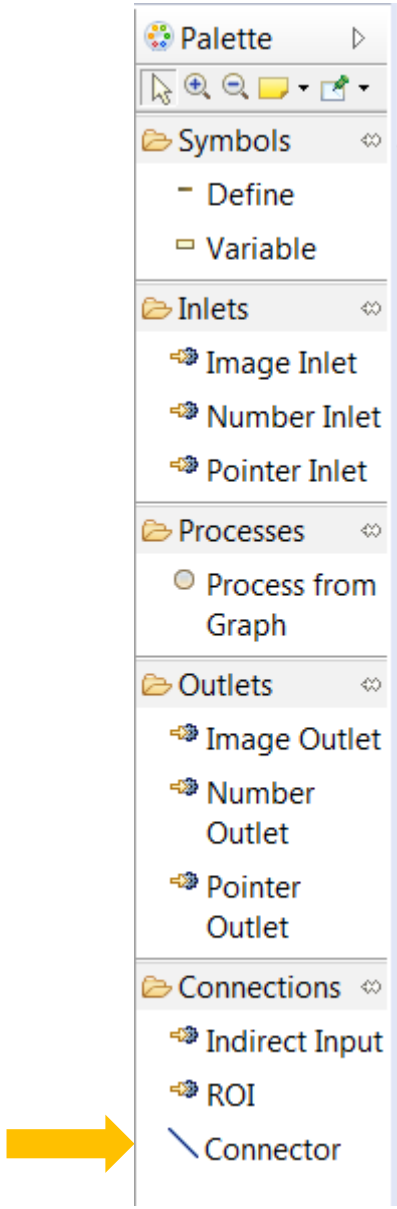




Make a program graph

- **Connect** blocks to create the following program graph

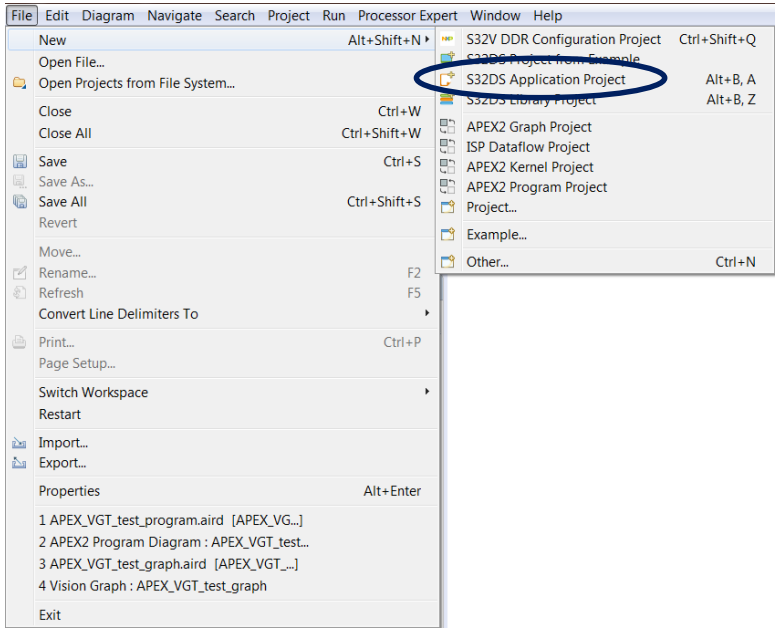




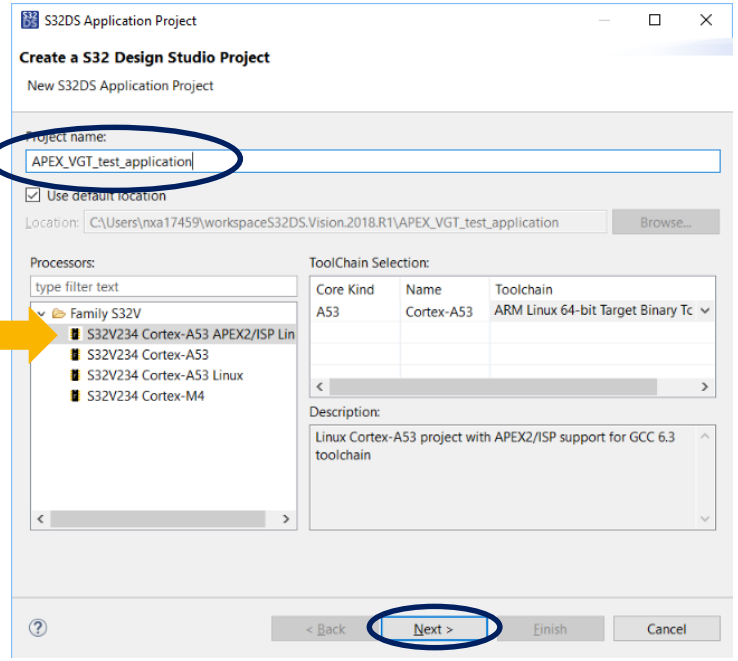
Make a Linux application project *without* APEX program project

- The next step is to configure block properties. This requires interaction with Linux application project
- So, we will make a Linux application project first.

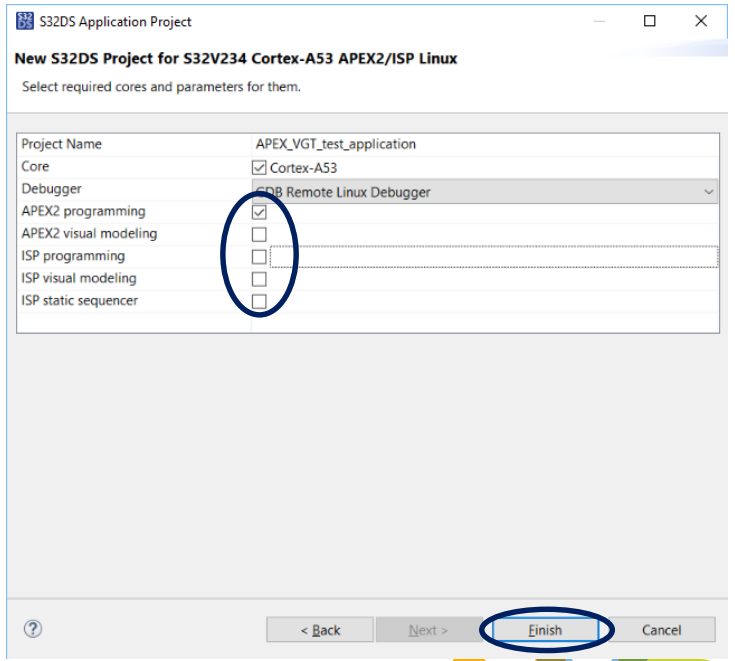
1. Go to **File –> New –> S32DS Application Project**



2. Type the **project name**: **APEX_VGT_test_application**
3. Select **project type** as shown
4. Hit **Next**

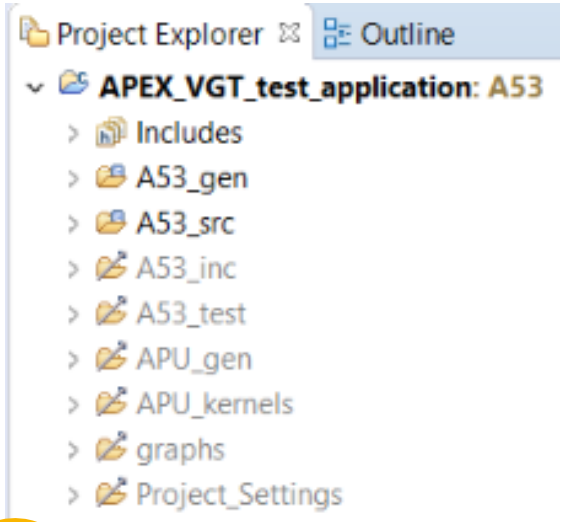


2. Since we are developing separate APEX graph projects and not using ISP, **deselect unnecessary options** as shown
3. Hit **Finish**



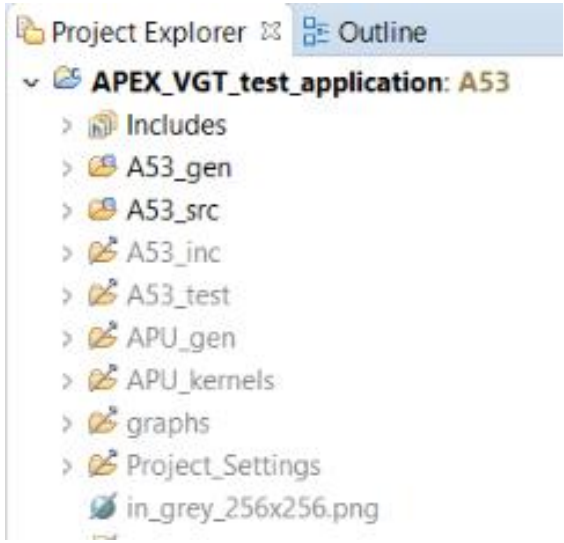
Make a Linux application project *without* APEX program project

- We can now see the project in the **Project Explorer**



Application project is READY!

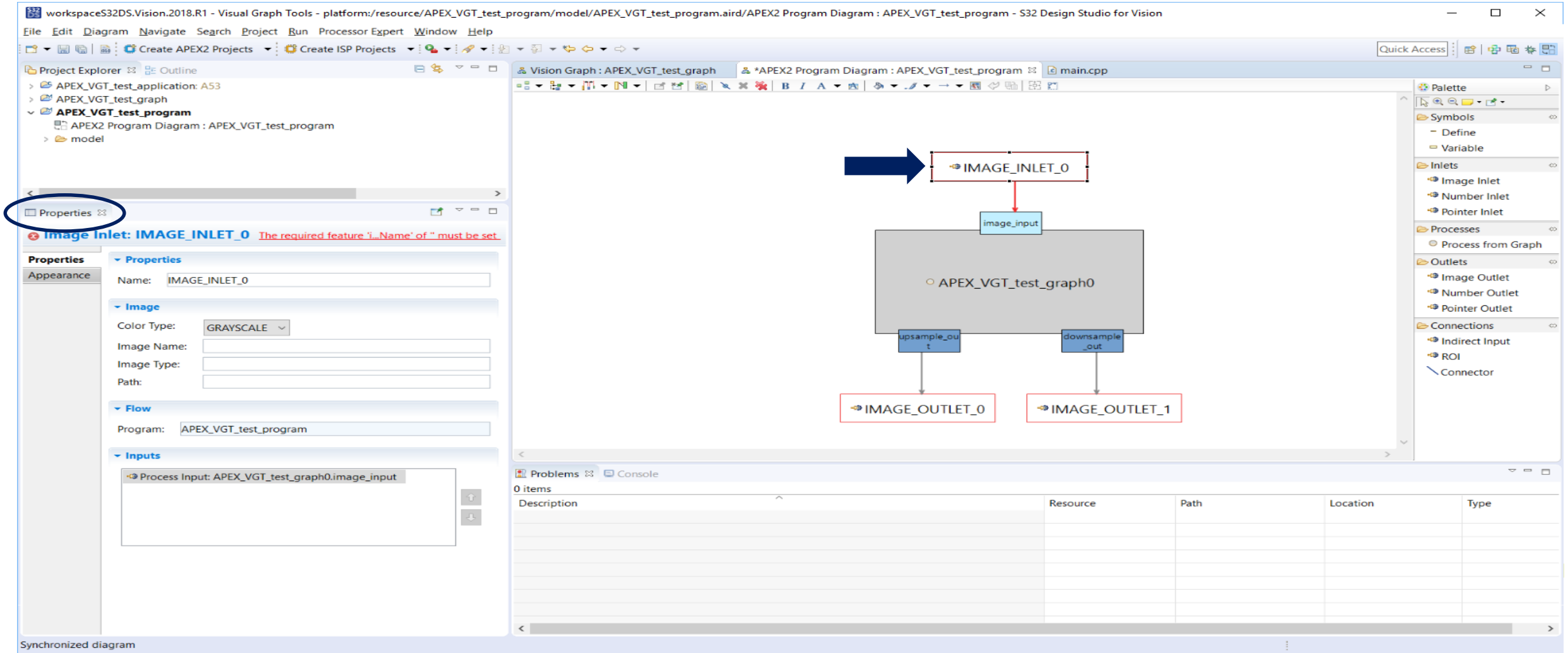
- Copy the picture "**in_grey_256x256.png**" from ***s32ds_installation_directory*\S32DS\s32v234_sdk\demos\data\common** to the project folder



We will stop here with application project and get back to "*configure block properties*"

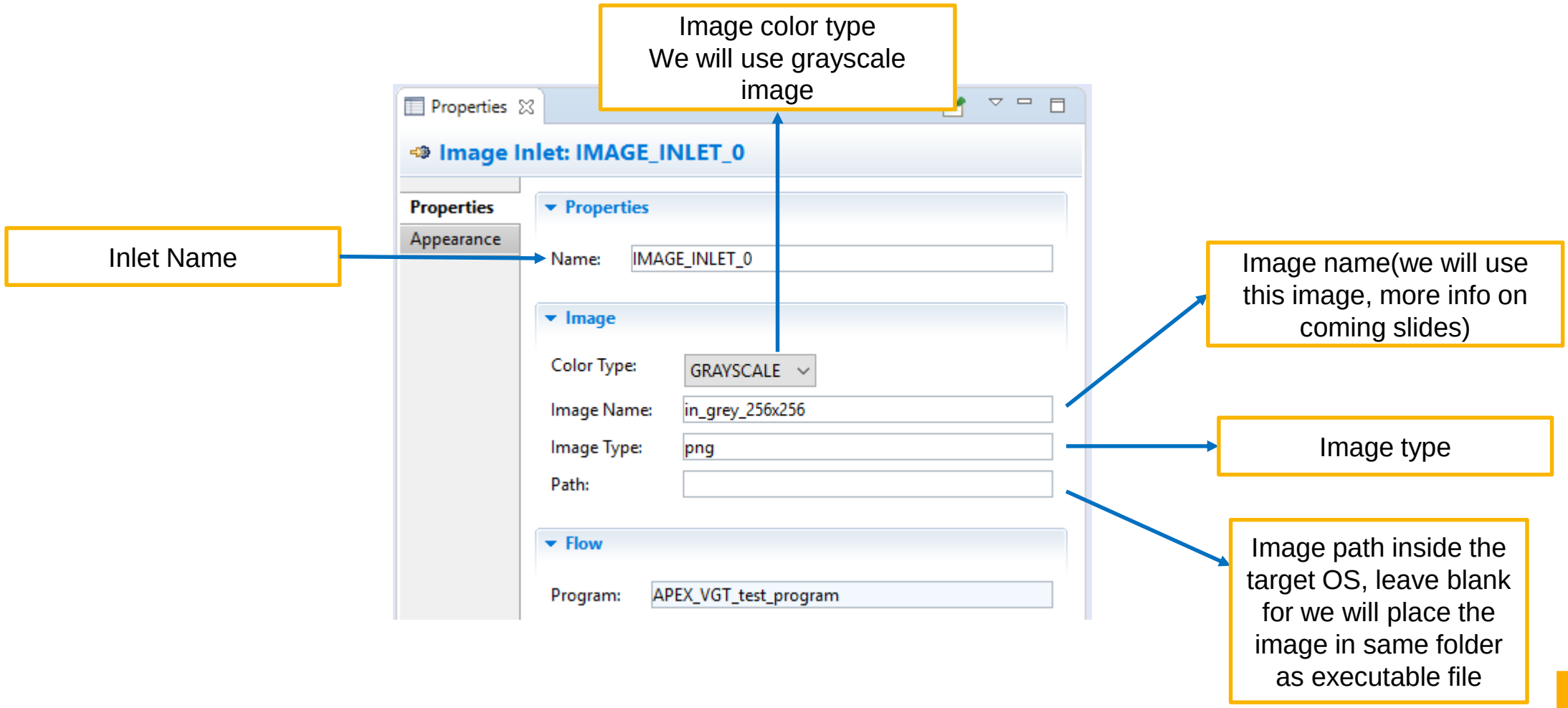
Configure the Blocks Properties

- Now we have Linux application project ready. We will get back to configuring block properties of program project
- Select the **IMAGE_INLET_0** block
- **Look at** (open if not visible) the properties window



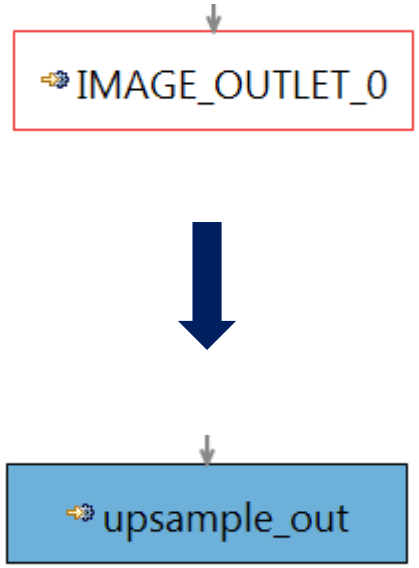
Configure the Blocks Properties

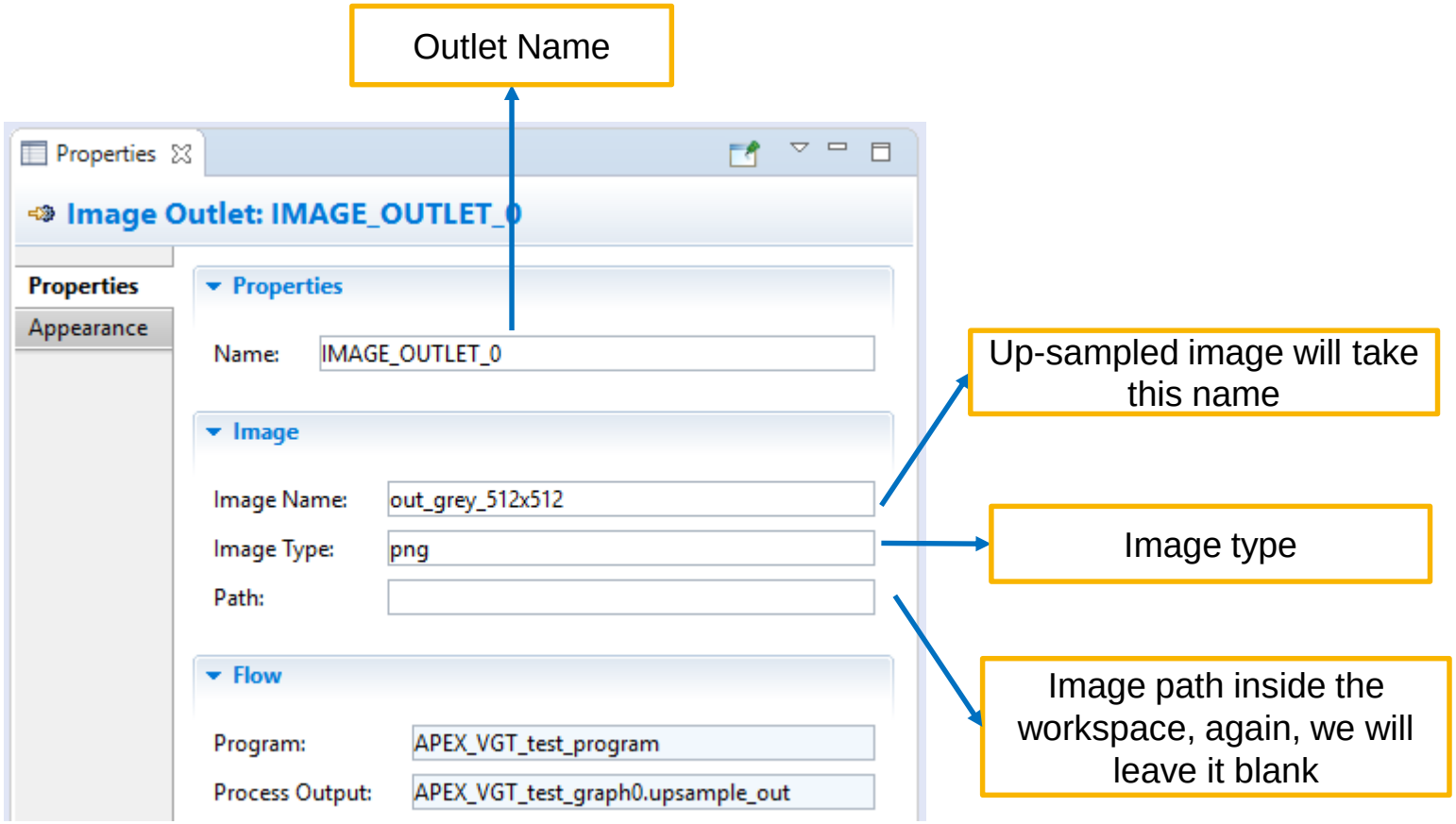
- **IMAGE_INLET_0** block **: Configure** the properties like follow:



Configure the Blocks Properties

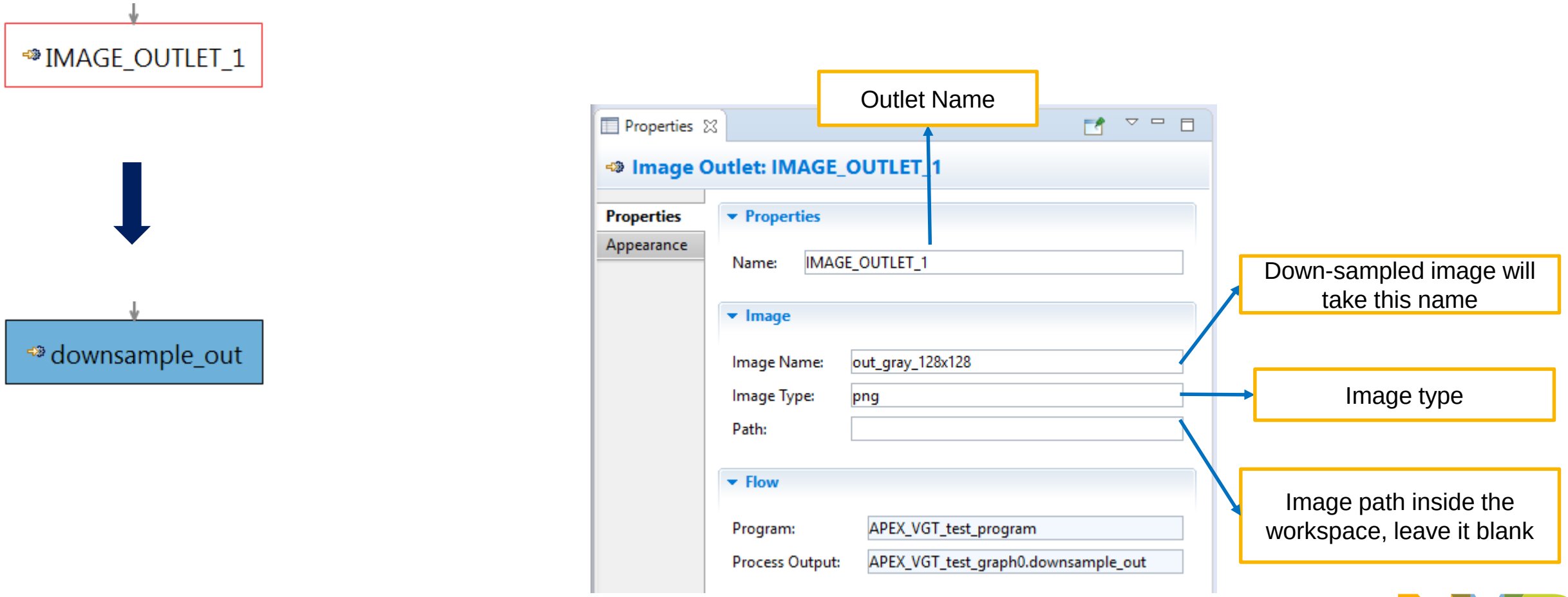
- Select **IMAGE_OUTLET_0** block





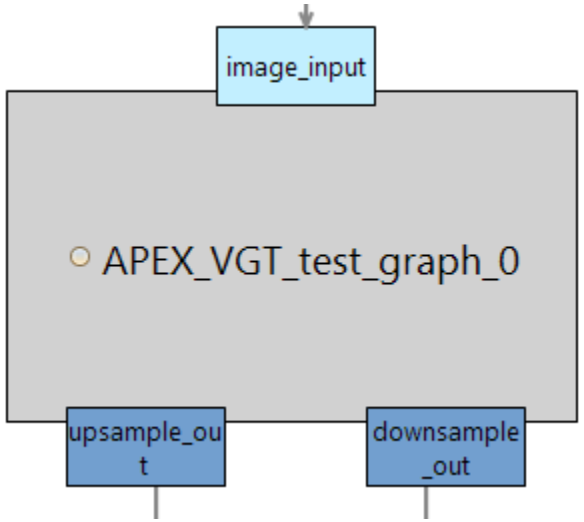
Configure the Blocks Properties

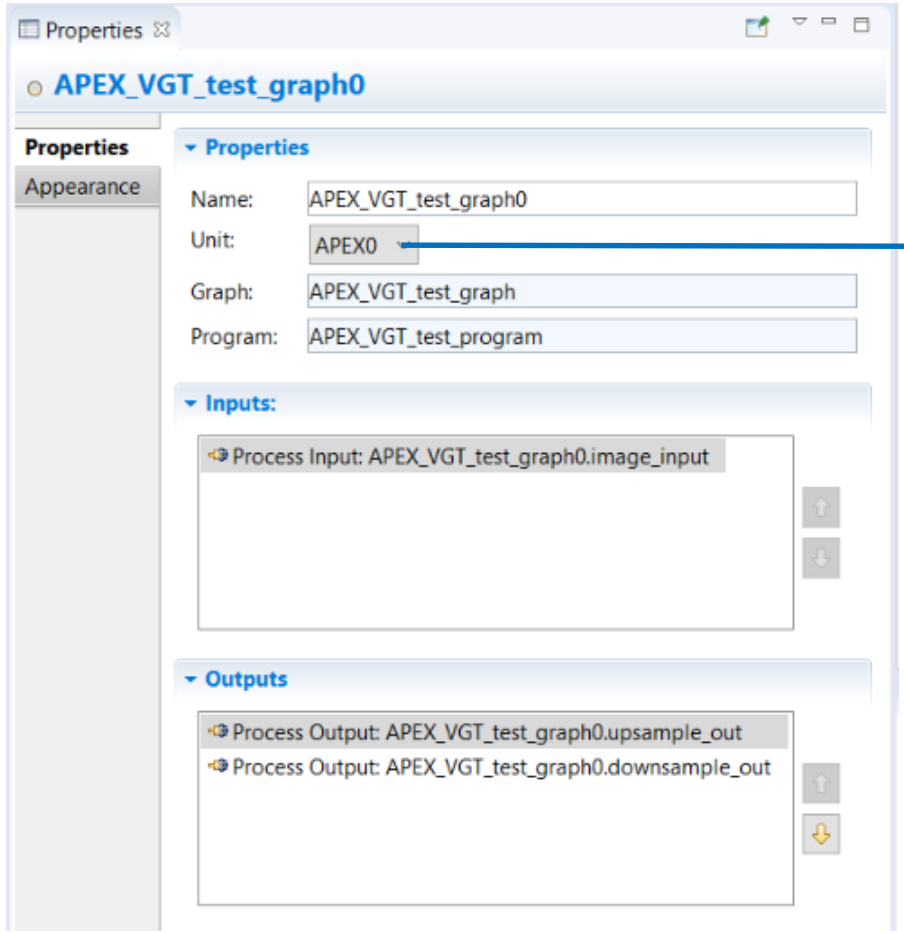
- Similarly, select **IMAGE_OUTLET_1** block



Configure the Blocks Properties

- Now, we will select on which APEX we want to run our graph
- Select **APEX_VGT_test_graph_0** block

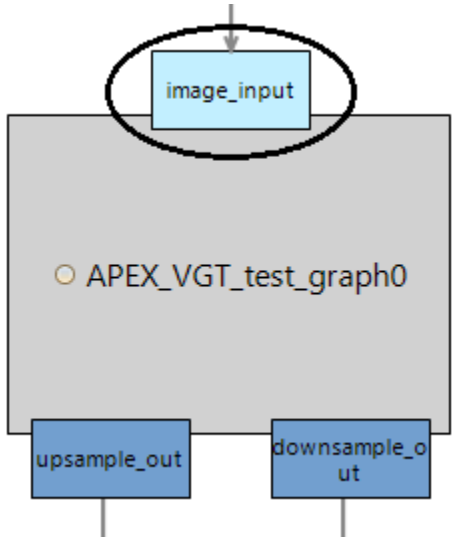


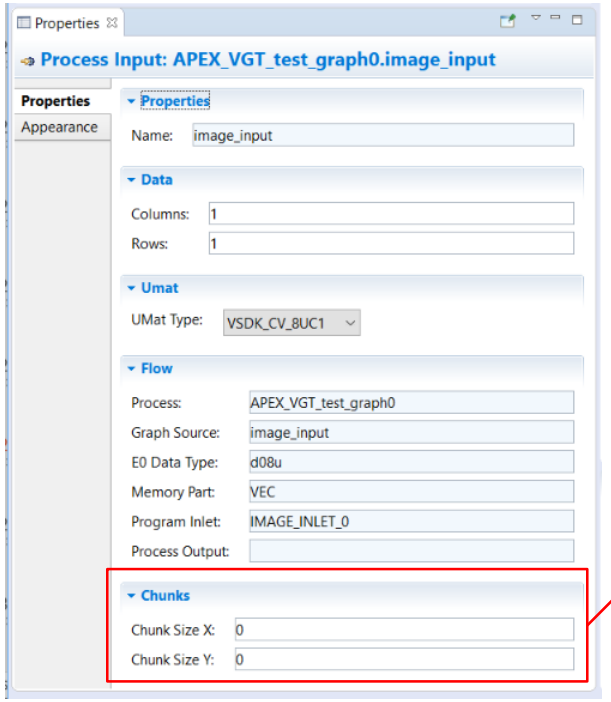


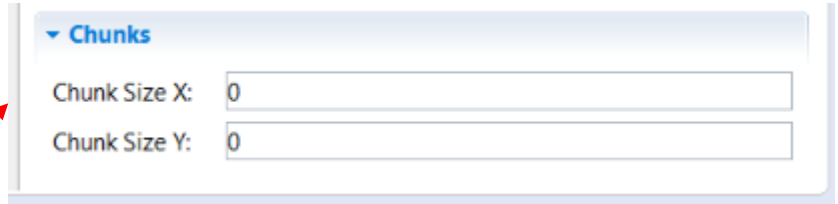
Select the APEX on which you want to run the graph

Configure the Blocks Properties

- You do **not** need to configure the **image_input** properties
  - But, if you want to optimize graph performance, you can hand-tune chunk size of the input. (See APEX documents for more information)
  - Otherwise, leave chunk size to “zero” and ACF will decide chunk size automatically

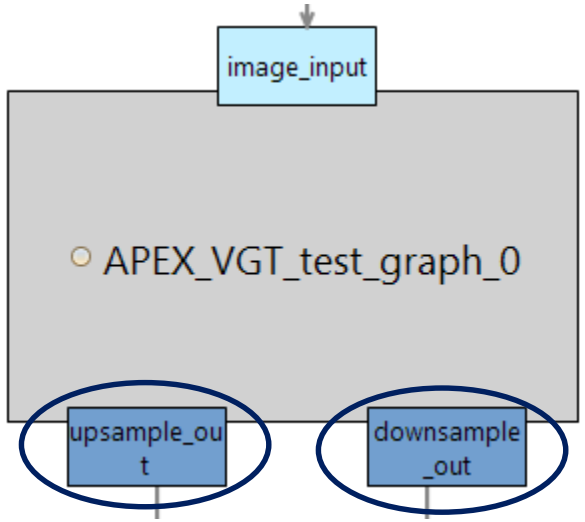


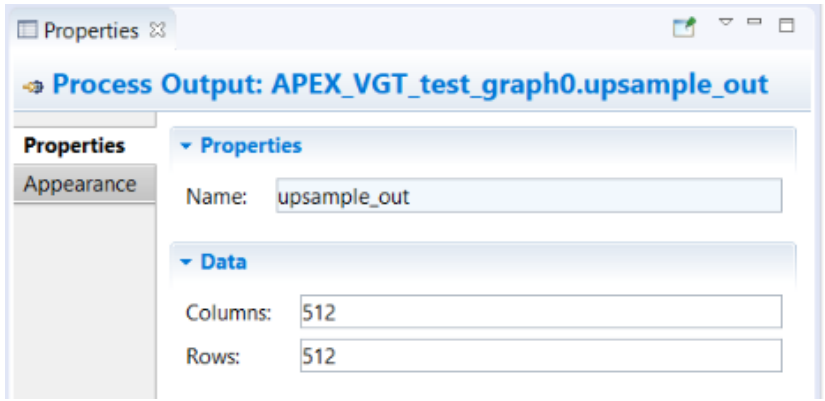


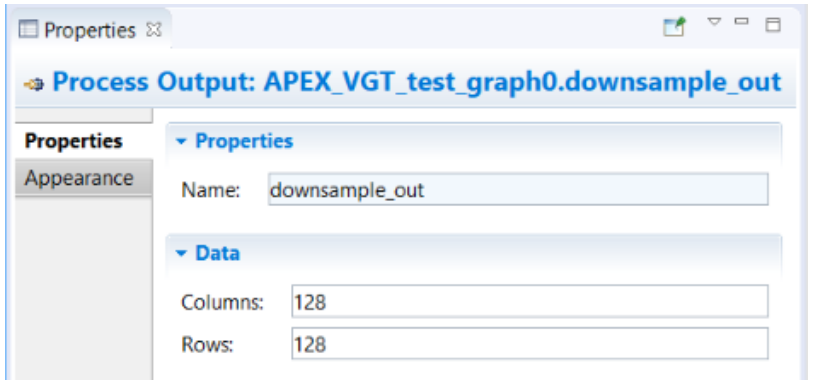


Configure the Blocks Properties

- We will now specify the output image buffers size
- Select **upsample_out** and **downsample_out** of **APEX_VGT_test_graph_0** block
- Modify its properties as shown below



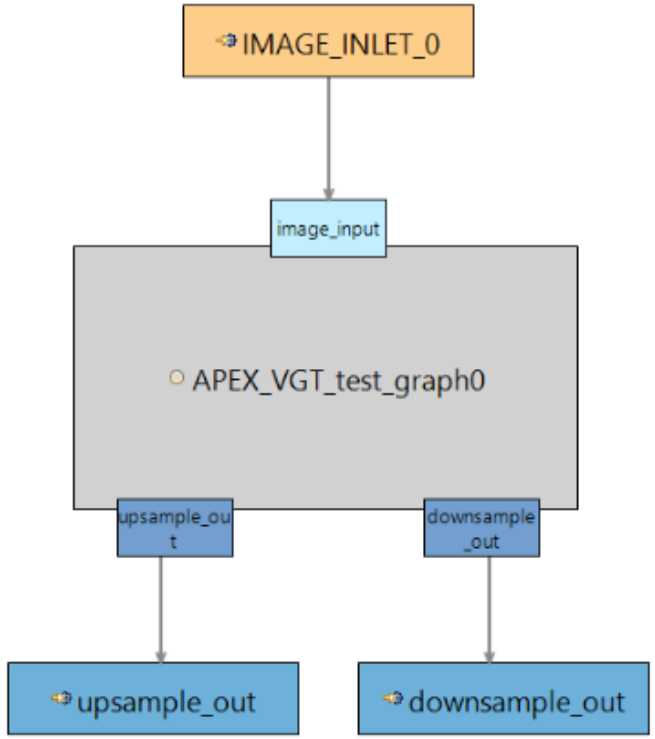


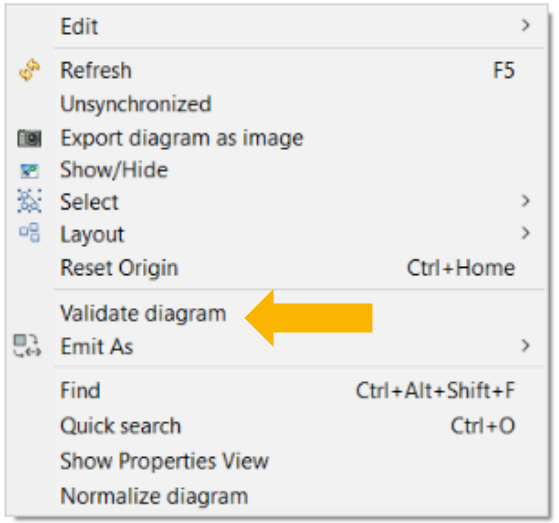


APEX program project is now READY!

Validate graph for correctness

- **Save** the graph
- **Right Click** anywhere in the white part of the graph window
- **Validate** graph
- **Validation errors** are reported and can be taken care of in a same way as explained before

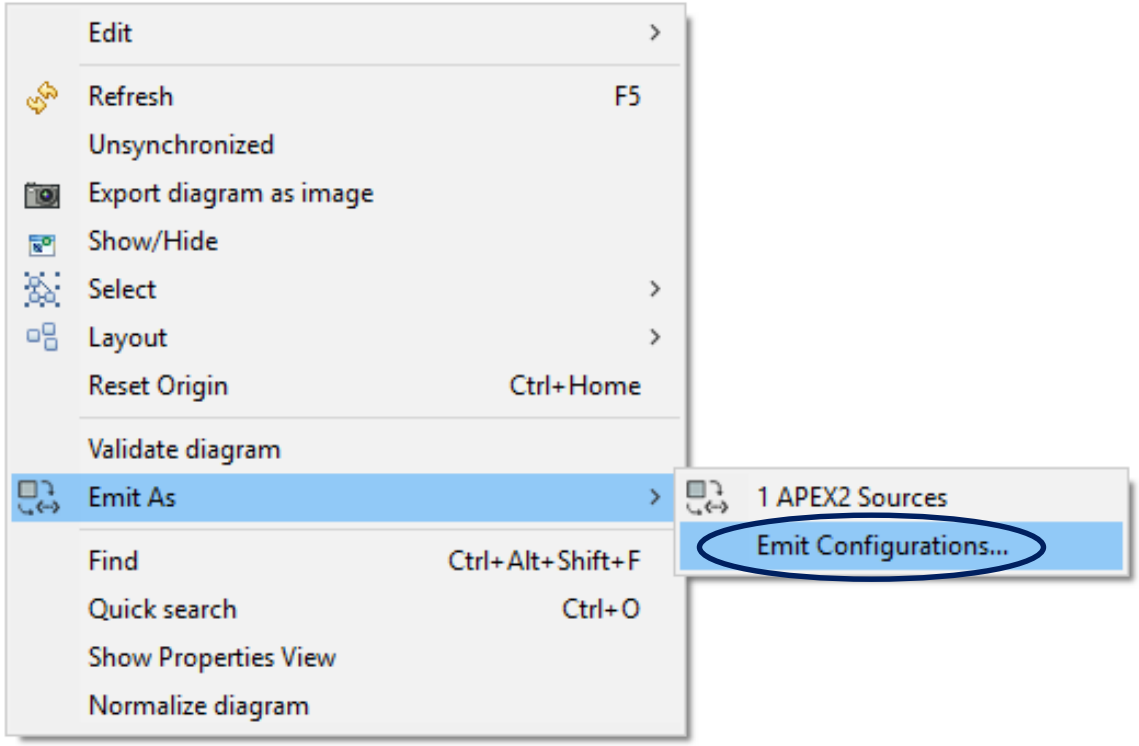




Select the destination of autogenerated source code 1 of 3

- By default all source code will be generated inside the APEX program project itself
- We can reconfigure the destination of source code to any other open projects.
  - We will use this feature and generate the source code in Linux application project.

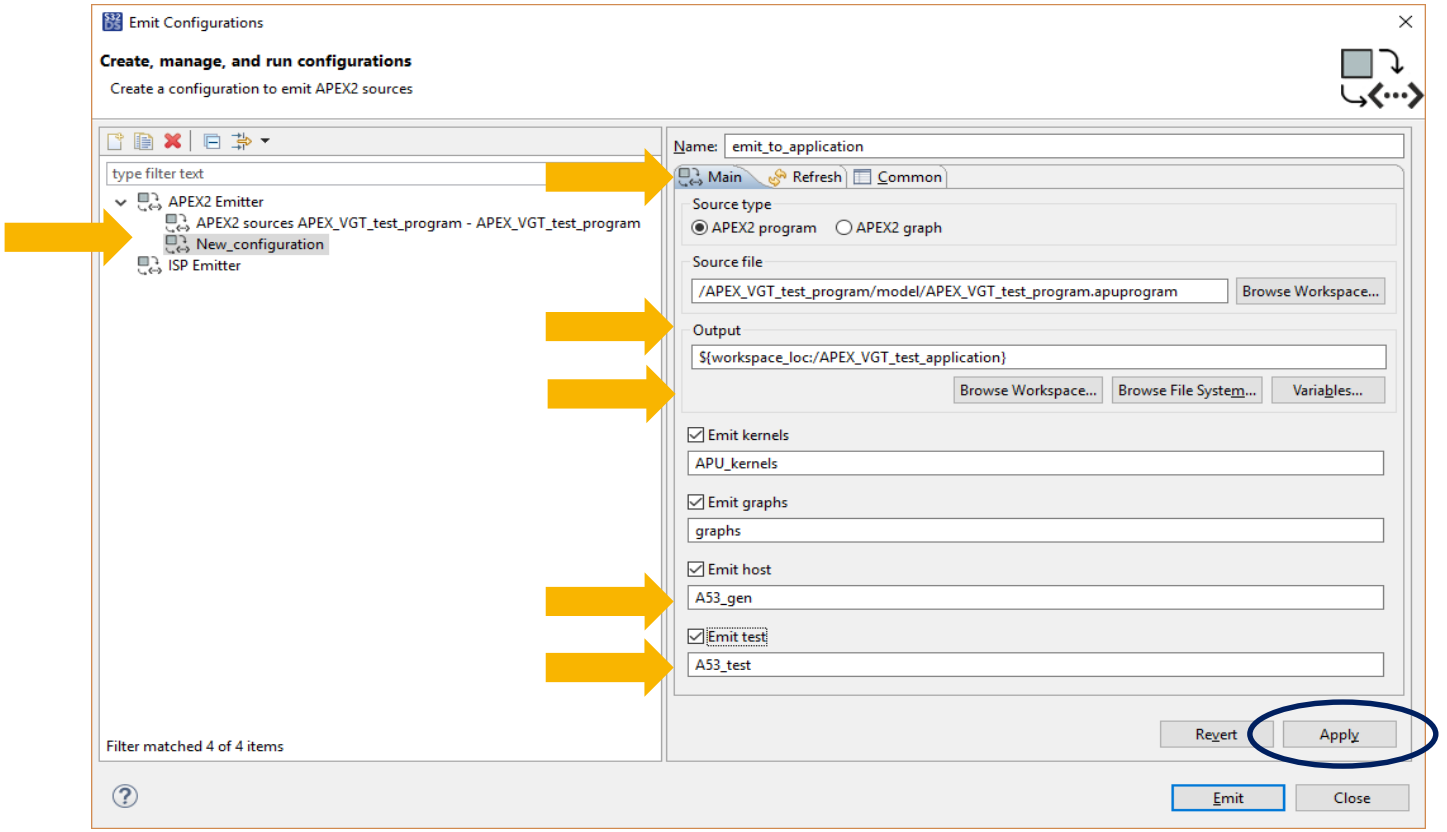
1. Select the **Emit Configuration..** option.



Select the destination of autogenerated source code 2 of 3

- Define a new configuration and specify where we want to generate our source code.

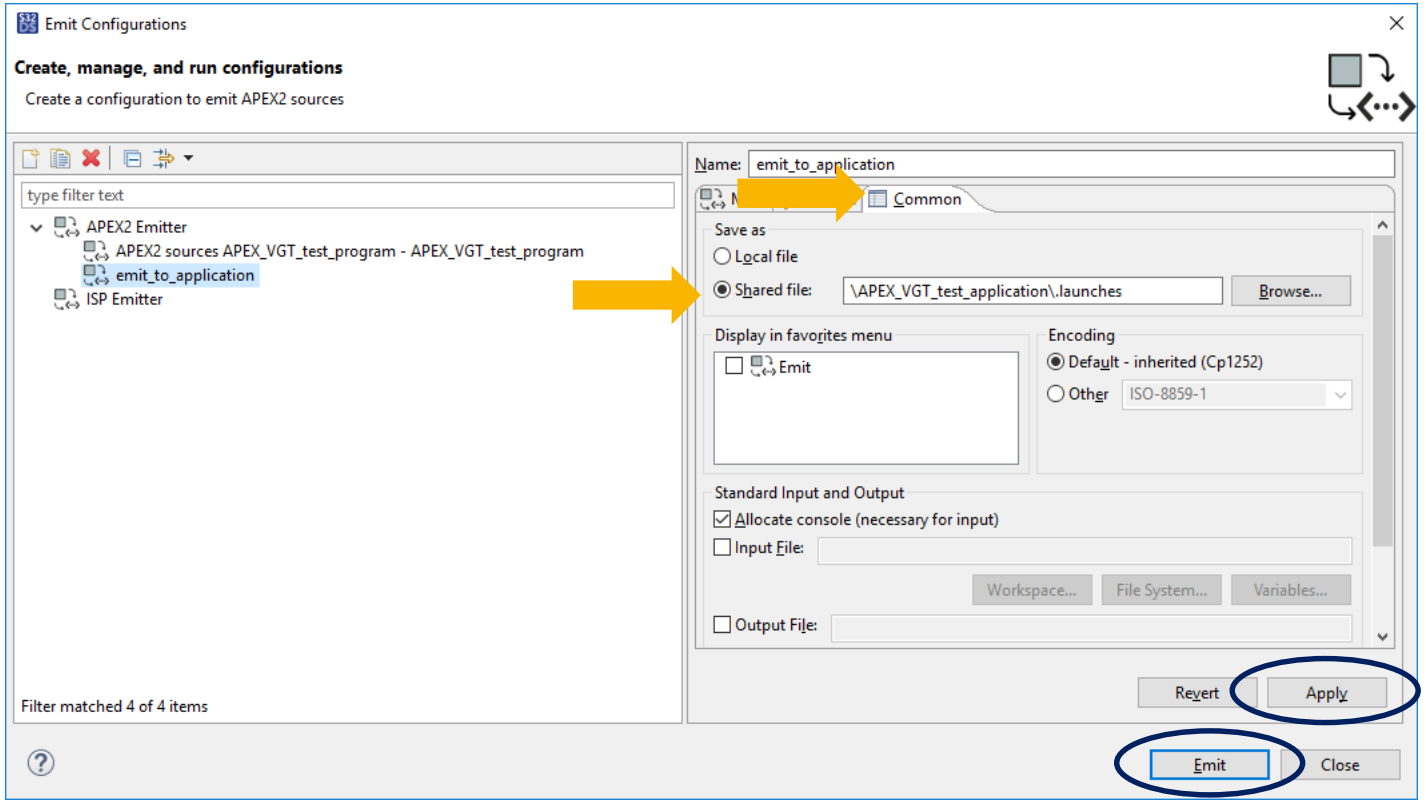
1. **Change** the **configuration** as shown in the picture
2. Click on **Apply** to save the changes



Select the destination of autogenerated source code 3 of 3

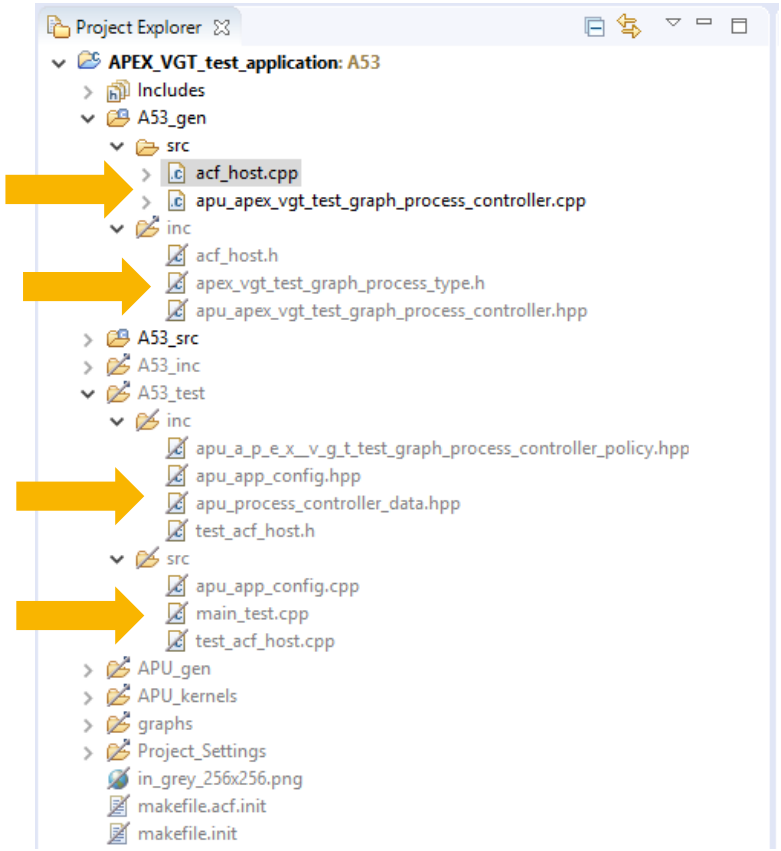
- Edit some more configuration

3. Go to **Common** tab.
4. Select the **\APEX_VGT_test_application\.launches** folder under "*Shared files"* option here.
5. **Apply** the settings and Hit **Emit** button to generate a source code at the designated location



Emit the source code

- Auto generated code can be seen inside the project folder
  - Note: If you can not see source code, please right click on the project and click on **Refresh** from the menu.



LINUX APPLICATION PROJECT FOR APEX

Once APEX code is ready, we will now build our Linux application

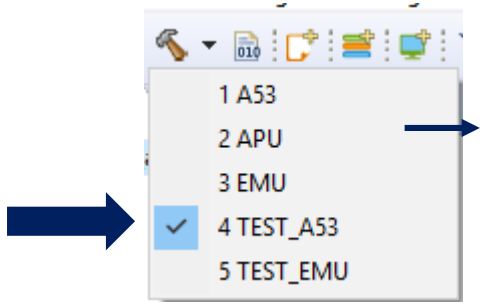
Application Code for APEX

- Basic, auto generated, application code template for APEX can be found in A53_test/src/**test_acf_host.cpp**
- ACF_APP_CALL() inside the A53/src/**main.cpp** is just a place holder.
- User should copy/add/change code inside the A53_gen/**acf_host.cpp** and A53/src/**main.cpp** according to his/her application needs or structure
- User should use build config **TEST_A53** for this tutorial example, and during early APEX graph development. Later, once host code is developed and added to A53_gen/**acf_host.cpp**, the user should switch to build config **A53**
  - **Note:** In this tutorial we will not change default structure as it is not necessary

Application code for APEX: Compile

- In this tutorial, we do not need to add/change any default code

- Go to **C/C++ perspective** and **compile** the application for **TEST_A53**



Info: APU option does “Offline Process Resolution” as described in the ACF User Guide

Application code for APEX: Run

Execute your APEX_VGT_test_application.elf binary on the target!

Connect and Observe

- Do not forget to copy *in_grey_256x256.png* to the same directory where you copied binary
- Run the application
- You can find 2 new image files generated in the same directory
  - One will be doubled the size of the original image and the other will be half the size of the original image

SECURE CONNECTIONS
FOR A SMARTER WORLD