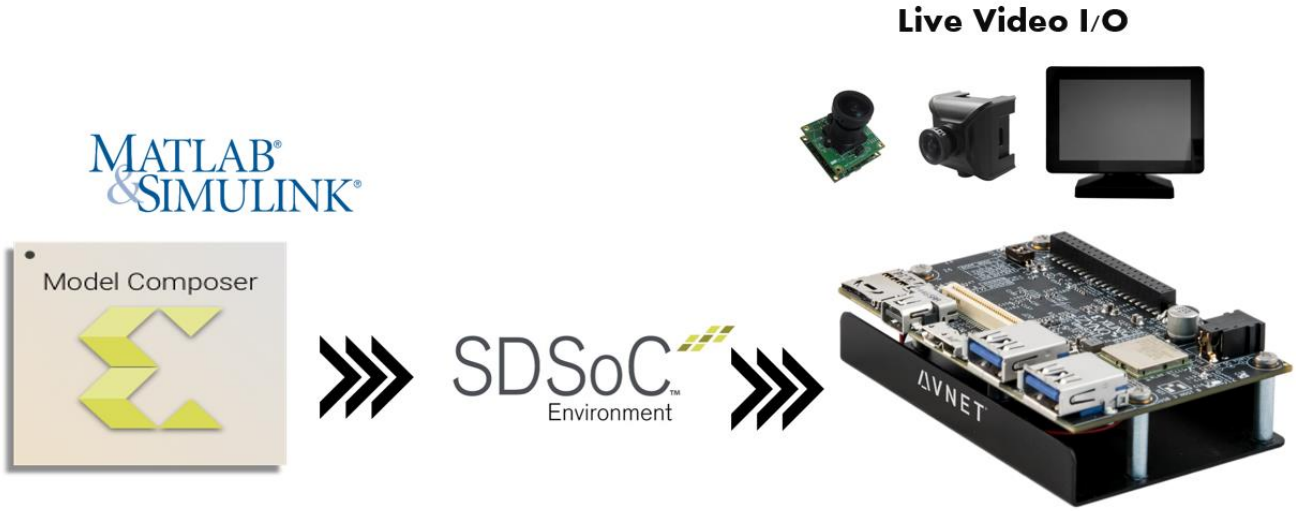
XILINX DEVELOPER FORUM

# Design and Deploy Accelerators for Embedded Vision Systems using Model Composer and SDSoC

# Goal

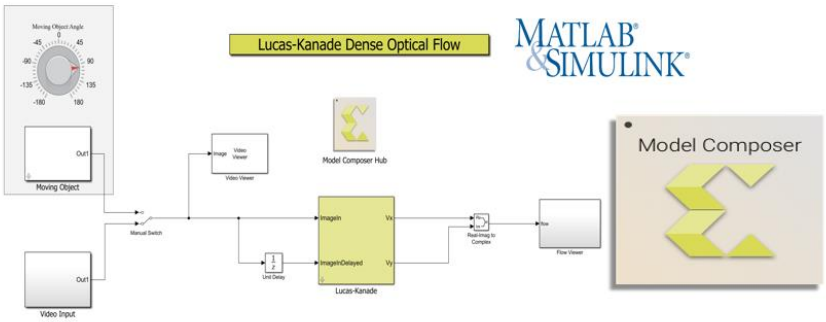
In this self-paced lab, you will walk through a step-by-step flow of using a Model-Based Design entry to model, simulate and accelerate a Lucas-Kanade Dense Optical Flow algorithm in the Programmable Logic on the ***Avnet Ultra96 board*** and view the results of the acceleration on the board with live video I/O.

You will start the lab with exploring and simulating the algorithm, captured at a high-level of abstraction using the ***Xilinx Model Composer*** add-on toolbox within the MathWorks Simulink® environment and end with deploying the algorithm on the ***Avnet Ultra96 board*** with live video I/O, via integration with the ***SDSoC Development Environment***.




# Steps Involved

**Step 1:** Introduction to High-level Modeling and Simulation with Xilinx Model Composer and Simulink (LK Dense Optical Flow Algorithm)

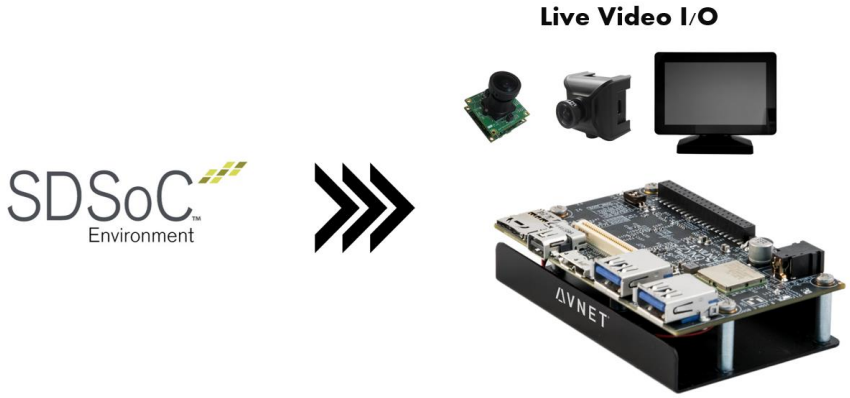


**Step 2:** Automatically generate Optimized Code to accelerate Design in Programmable Logic


**Step 3:** Add Model Composer Generated Accelerator into SDSoC project


**Step 4:** Compile the project for the Avnet Ultra96 reVISION platform

**Step 5:** View the results of acceleration with live Video I/O on the Ultra96



# Software Used

The following tools have been installed and setup on this lab machine

- Xilinx Model Composer 2018.2
- SDSoC Development Environment 2018.2
- MATLAB & Simulink R2017a

# Hardware Setup Used

The following setup is available to view the final results with live Video I/O

- Avnet Ultra96 Board
- e-con USB3 Camera for Video Input
- HDMI Display Monitor for Video Output

# Launching Model Composer

You lab machine should have already have MATLAB launched, setup with the Xilinx Model Composer add-on toolbox and should be ready to use – Please follow the instructions below to if you'd like to launch Model Composer on your own:

1. Open a Terminal
2. cd to **/home/user/xdf_labs/xmc_ultra96_lab/src/**
3. Launch Model Composer by typing **model_composer -matlab /opt/Matlab/R2017a**

# STEP 1: DESIGN OF THE LUCAS-KANADE DENSE OPTICAL FLOW WITH MODEL COMPOSER

Using the Lucas-Kanade Dense Optical flow as an example application, we will introduce you to key features in the Xilinx Model Composer add-on toolbox within Simulink that enable the design of high-level frame-based algorithms that can be accelerated on the programmable logic through automatic code generation.

## Lucas-Kanade Dense Optical Flow: Algorithm Overview

Optical flow refers to the apparent motion of objects, surfaces or edges between two consecutive input video frames caused by relative motion between an observer and the scene.

The Optical Flow algorithm uses two successive images in time, and calculates the direction and magnitude of motion at every pixel position in the image. The calculation is a simple implementation of the Lucas–Kanade method for optical flow estimation. The dense optical flow algorithm returns a flow vector [u v] for every pixel in the image, representing up or down motion in the vertical direction (v) , and left or right motion in the horizontal direction (u) showing movement of the pixels from the first frame to the second.

## (Optional) Math Involved:

To solve the optical flow constraint equation for *u* and *v*, the Lucas-Kanade method divides the original image into smaller sections and assumes a constant velocity for motion in each section. Then, it performs a weighted least-square fit of the optical flow constraint equation to a constant model for $[u\ v]^T$ in each section Ω. The method achieves this fit by minimizing the following equation:

$$\sum_{x\in\Omega} W^2[I_x u + I_y v + I_t]^2$$

*W* is a window function that emphasizes the constraints at the center of each section. The solution to the minimization problem is

$$\begin{bmatrix} \sum W^2 I_x^2 & \sum W^2 I_x I_y \\ \sum W^2 I_y I_x & \sum W^2 I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum W^2 I_x I_t \\ \sum W^2 I_y I_t \end{bmatrix}$$

## High-level Block Diagram

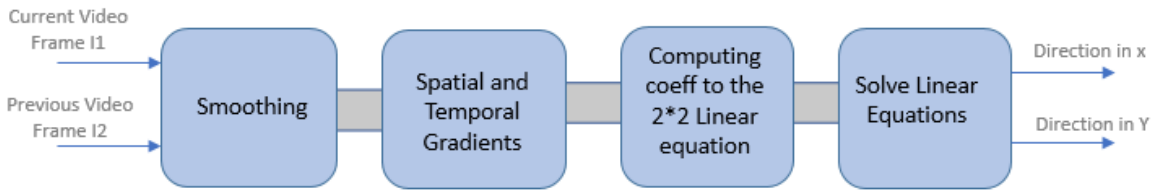



Fig: 1 – Block Diagram for of LK optical flow

**Inputs:** Current Video Frame I1 and Previous Video Frame I2

**Outputs:** Flow vector for each pixel (Direction in X, Direction in Y)

Here are the high-level steps involved in computing the optical flow vectors for each pixel:

1. Smoothing (Spatial Filtering) is performed on the input video frame to reduce noise.
2. Spatial gradients Ix and Iy are calculated separately on X-axis and Y-axis of an image. The displacement between successive frames is also calculated which is called temporal gradient (It).
3. Compute the intermediate products like W2Ix2, W2IxIy e.t.c (Where, W is a Window function)
4. Smooth the gradient components, Ix, Iy, and It, using a separable and isotropic 5-by-5 element kernel.
5. Solve the 2*2 linear equation for each pixel to obtain the flow vector at the Output

*How would you move from this High-level Block Diagram towards implementing and accelerating this algorithm on the programmable logic in your embedded vision system?*

# Frame-based Implementation Model in Model Composer

Xilinx Model Composer is an algorithm-centric design tool that enables algorithm engineers to focus on designing their algorithms, for acceleration and deployment on the programmable logic, at a high-level of abstraction without worrying about low-level implementation specifics.

Using high-level performance-optimized Model Composer block libraries that are integrated into Simulink, you can focus on expressing the math in your algorithms without having to re-implement at a lower level of abstraction and move to an implementation model using low-level primitives.

The screenshot below shows how you can easily translate the high level block diagram for Lucas-Kanade Dense Optical flow algorithm into a frame-based implementation model within Simulink by leveraging key features in Model Composer

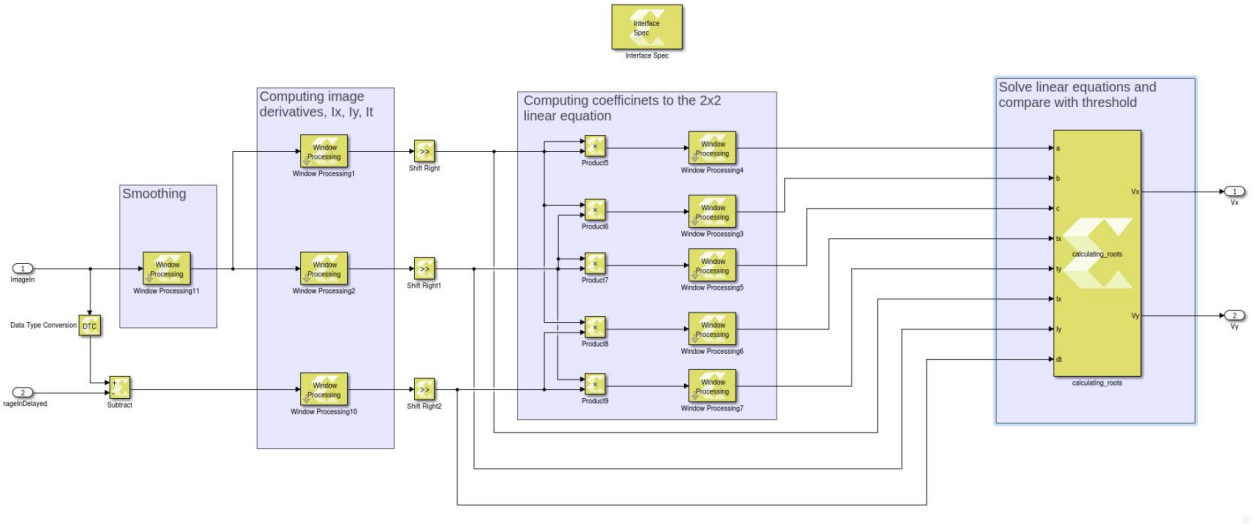



Fig: sub-sectional view of LK-optical flow design

Let's take a closer look at the design captured using Model Composer and get familiar with Model Composer's key features that enable design, simulation and implementation at a higher-level of abstraction

- **Launch Model Composer Example**
- **Explore the Design**
- **Simulate the Design**

## Launch the Example

- In the MATLAB Current Folder, navigate to:
  **home/user/xdf_labs/xmc_ultra96_lab/src/optical_flow**

- Double-click on the **`xmc_optical_flow.slx`** model to open the example design.

  **Note:** If you want to scale the model so all the blocks fit on your screen, Hit ***Spacebar***

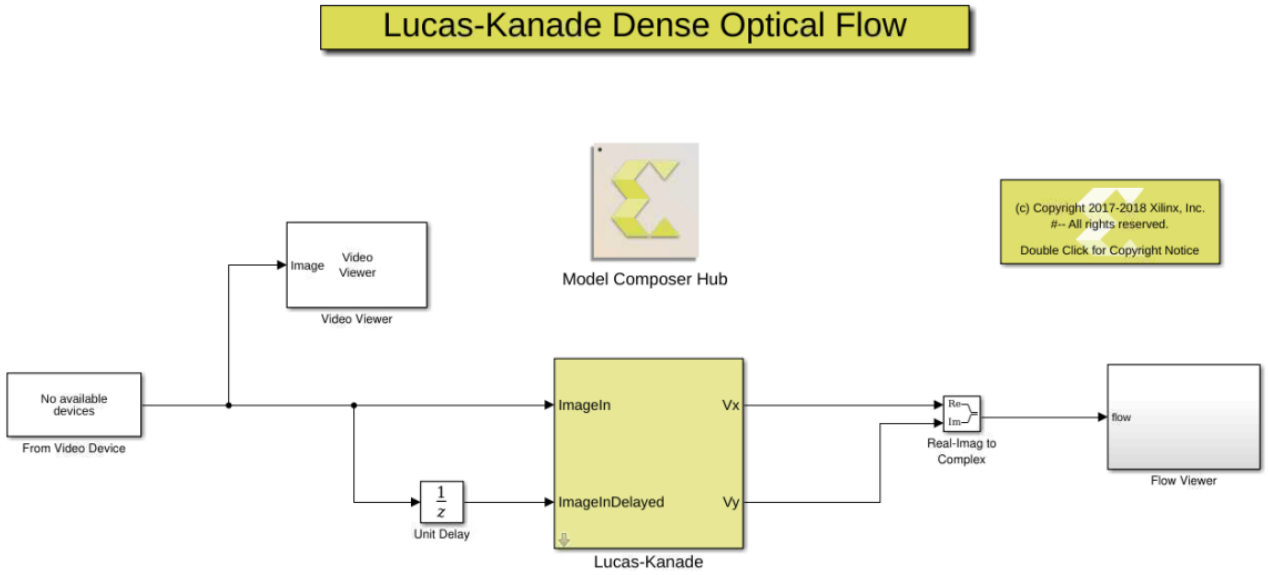



Fig – Model composer design overview

## Explore the Design

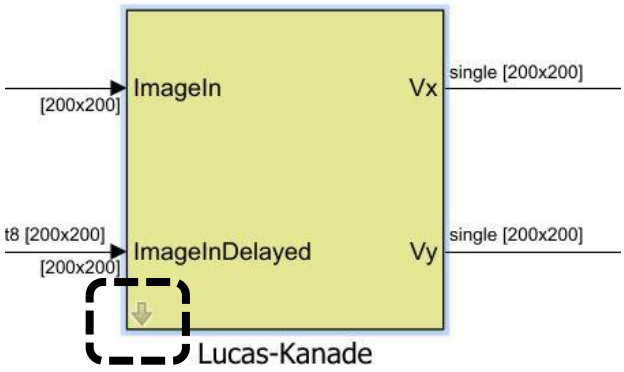
- ***Input/Source Block:*** The first step in designing/capturing an algorithm is understanding the inputs to it. The optical flow algorithm takes 2 consecutive video frames (current video frame, previous video frame) as inputs.

  You can directly connect video input streamed from a file or from a video device (e.g. USB webcam etc.) to the design using the ***"From Multimedia File"*** or the ***"From Video Device"*** blocks from add-on toolboxes like the ***Computer Vision System Toolbox*** or ***Image Acquisition Toolbox***

  Double click on the ***"From Multimedia File"*** block at the input to view how input video from a file is being fed into the algorithm

- ***Algorithm Implementation:*** The video input is connected to the top-level ***"Lucas-Kanade"*** Subsystem, which is essentially the portion of the design intended for implementation/acceleration on the programmable logic, expressed using optimized-blocks from the Model Composer block library.

To observe the design captured under the Subsystem, you can click on the ***"Look inside mask"*** icon on the Subsystem, highlighted in the screenshot below:




**OR** Right-click on the "**Lucas-Kanade**" block and Select "**Mask -> Look under mask**" as shown below.

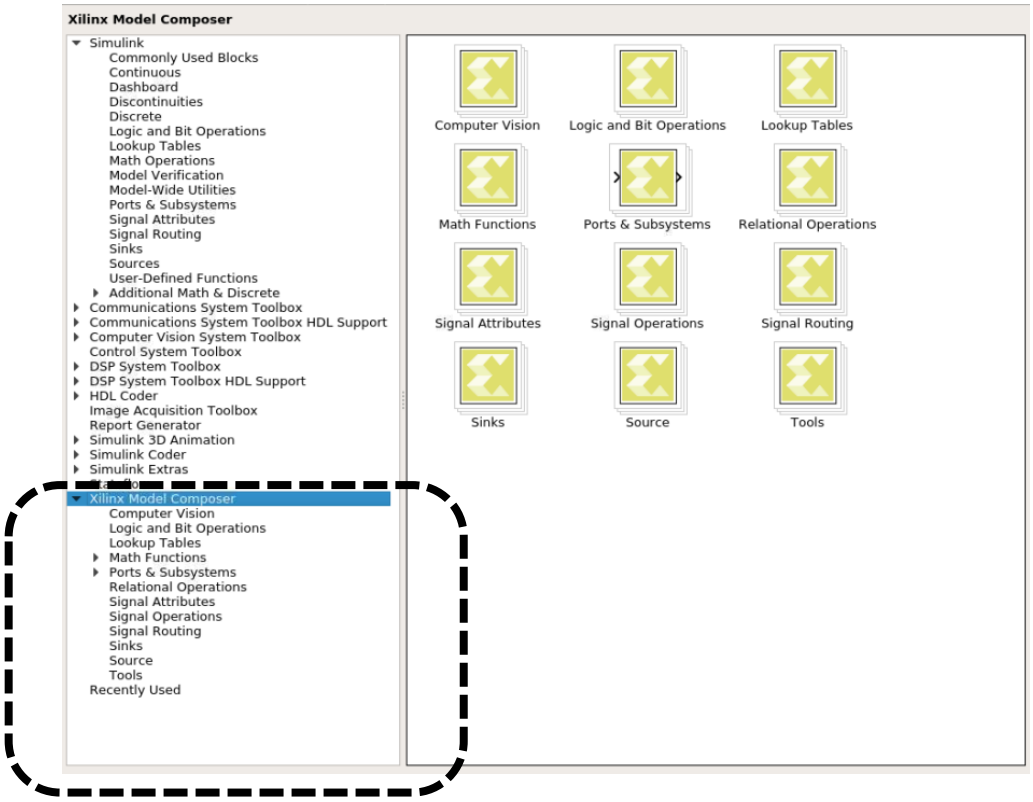

**Observe the following in the Design:**

1. Note the difference in appearance for the Model Composer blocks in the design versus the Simulink blocks – To design an algorithm that can be implemented in the FPGA fabric through automatic code generation, you must express the algorithm using optimized blocks from the Model Composer block library
2. Model Composer fits into the Simulink environment as a library of Xilinx-optimized blocks – It offers a wide range of performance-optimized blocks that are accessible from within the Simulink Library Browser

To explore the list of blocks available in the Model Composer library, use any of these techniques to open the Simulink Library Browser:

a. On the Home tab, click **Simulink**, and choose a model template. In the new model, click the **Library Browser** button 
b. At the command prompt, type: **slLibraryBrowser**

In the browser, navigate to the **Xilinx Model Composer** library (highlighted below)

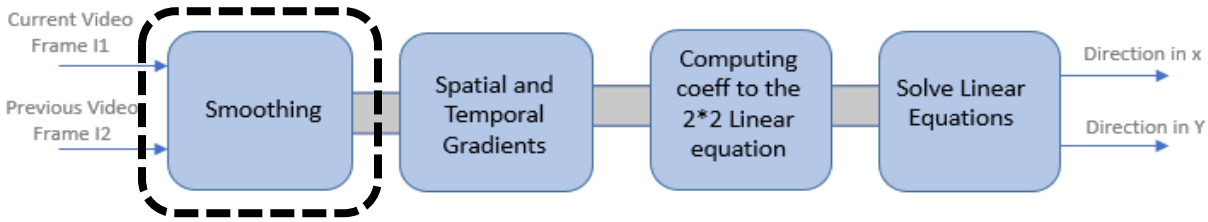



The Model Composer blocks are organized into subcategories based on functionality. For example, the ***"Computer Vision"*** sub-library offers performance-optimized xfOpenCV libraries from the ***Xilinx reVISION stack*** as blocks for image processing and computer vision applications, the Math and Linear Algebra sub-libraries offer HLS Math library functions as high-level blocks etc.

***3.*** Now, let's look at how the components (Smoothing, Calculation of Gradients, Computing Co-efficients and Solving the Linear Equation) in the block-diagram for the optical flow algorithm is implemented at a high-level using the blocks from the Model Composer library or by creating custom blocks to implement custom user functionality.

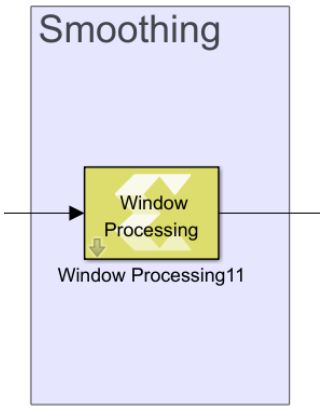
**Note:** If you don't want to dive into more details of algorithm implementation, you can skip over to the "Simulate the Design" section to simulate and view the results of the algorithm

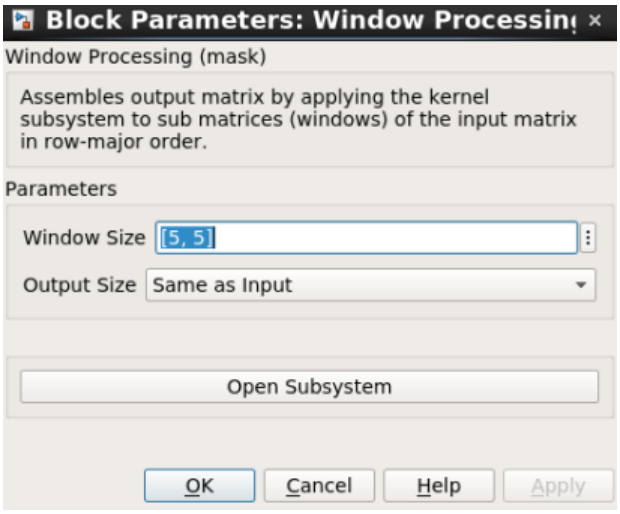
## Smoothing



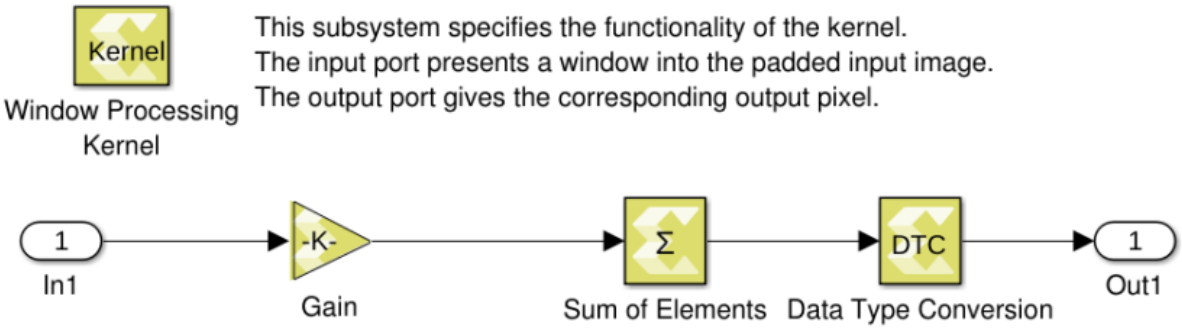

a. The smoothing operation is implemented using the ***"Window Processing"*** block from the Computer Vision sub-library. This block is used to implement custom 2D filters in your algorithm – It assembles an output matrix (Image) by applying a kernel subsystem to sub matrices of the input matrix (Image) in row-major order (sliding window-based operation). You customize Window Processing block by specifying its mask parameters and by adding blocks to the Kernel Subsystem, which defines the operation you want to perform on the sub-matrices

b. Double click on the window processing block in "Smoothing" area and observe the window size specified. In this case it is a 5x5 sliding window



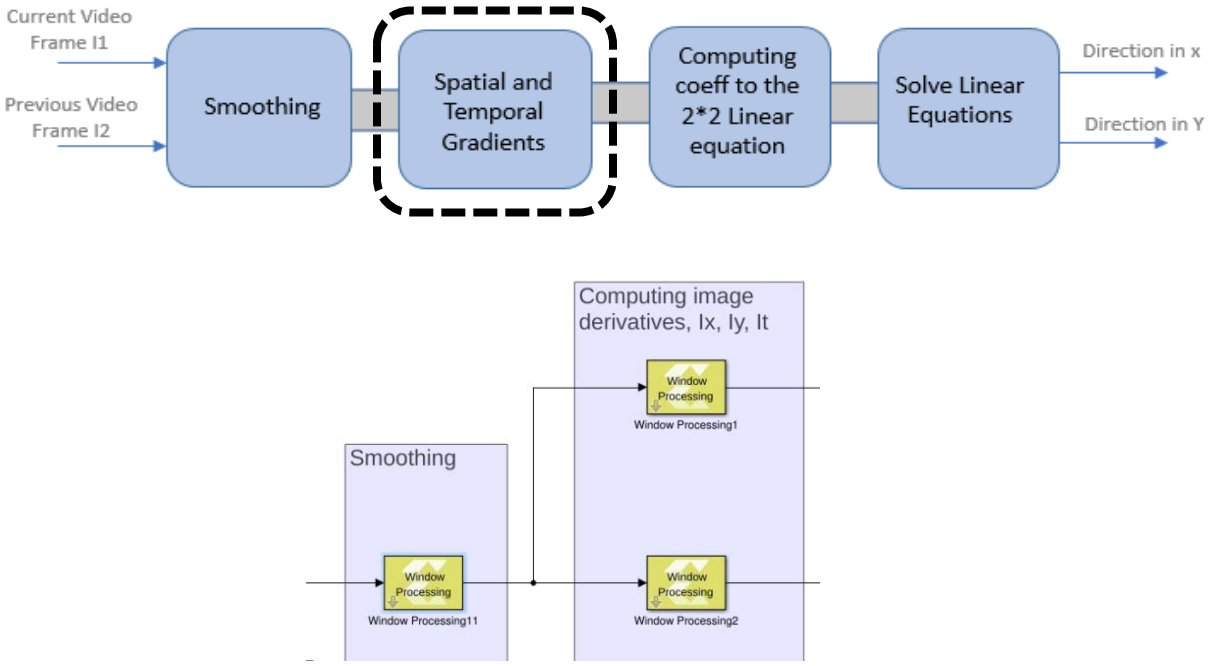
c. Now click on "Open Subsystem" to view the processing defined for each 5x5 sub-window.




d. The value of the ***"Gain"*** parameter for the Window Processing blocks defines filter/kernel coefficients.

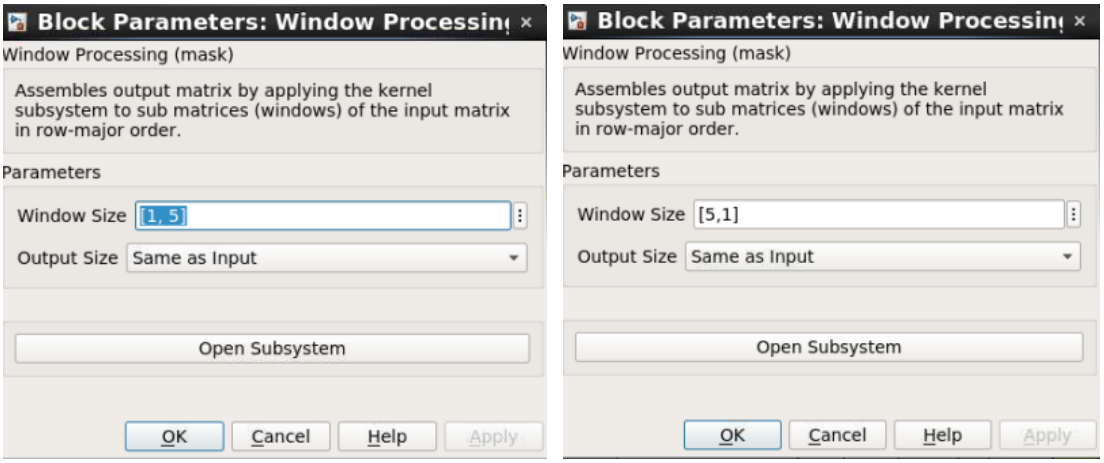
e. Click on the ***"Help"*** button on the Window Processing block to view more details about this block

## Computing Image Derivatives Ix, Iy, It



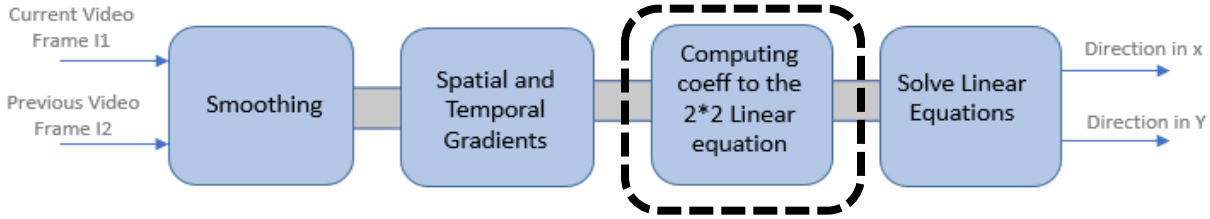

a. The output of the "Smoothing" operation is used to compute the spatial gradients **Ix, Iy** and the temporal gradient **It**

b. The ***Window Processing*** blocks are used again to compute the Image derivatives. Double-click on the Window processing blocks in ***"Computing image derivatives"*** section if you'd like to learn more about how the spatial and temporal gradients are calculated.



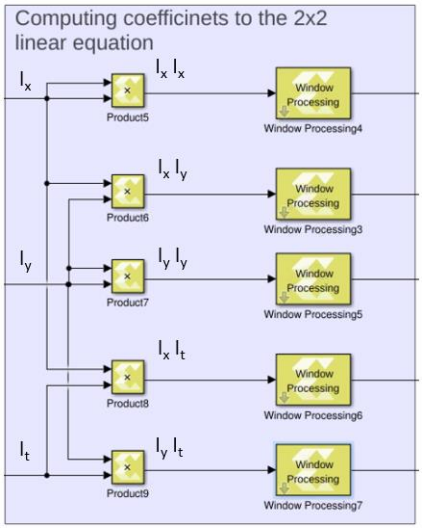

c. Note that in the case of calculating the spatial gradients in the X and Y direction, we use a 1-D vector [1 5] or [5 1] as the Sliding Window Size in the Window Processing block

## Computing Coefficients for solving Linear equation

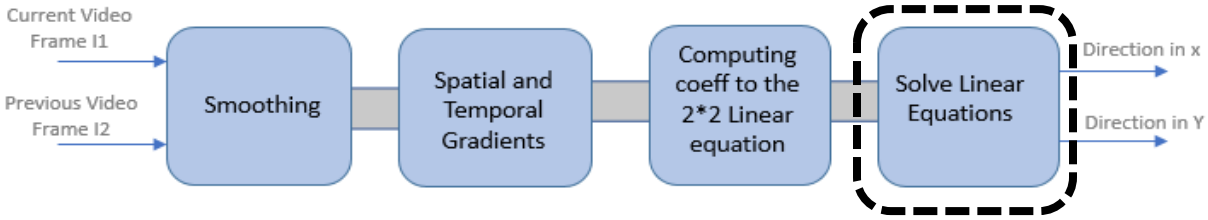



We use the ***Product*** block from the Model Composer library to calculate the co-efficients for the Linear Equation below

$$\begin{bmatrix} \sum W^2 I_x^2 & \sum W^2 I_x I_y \\ \sum W^2 I_y I_x & \sum W^2 I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum W^2 I_x I_t \\ \sum W^2 I_y I_t \end{bmatrix}$$



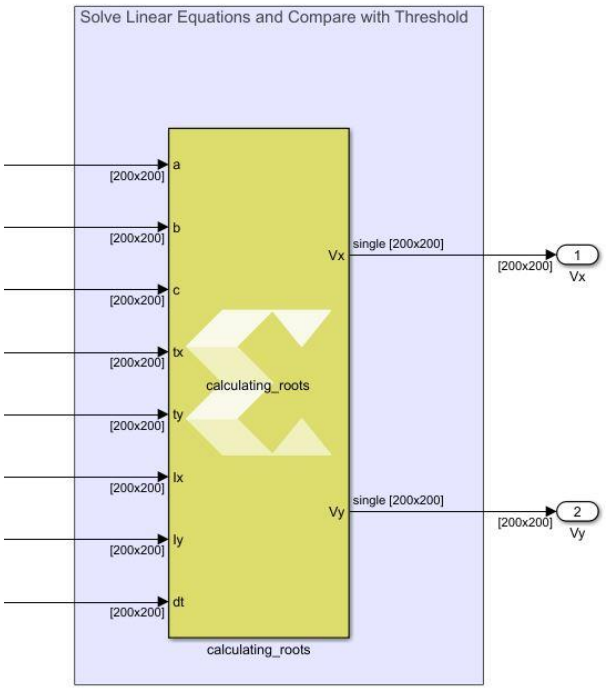

## Solving Linear Equation by creating a Custom Block




In this section of design, you will be introduced to a key feature in Model Composer – the ability to create custom blocks. To solve the linear equation and find the optical flow vectors, mathematically we need to calculate the Eigen values of a matrix.

Sometimes it's easier to express a mathematical operation in your algorithm by writing code instead of expressing it using individual constituent blocks. ***Model Composer lets you create custom blocks by importing C or C++ functions that can be added to a library*** for use in models alongside other Xilinx Model Composer blocks.

The ***"calculating_roots"*** block in the example model highlighted below is a custom block created by importing the equivalent ***"calculating_roots"*** function defined in the ***"calculating_roots.h"*** header file




**Optional:** The steps below will walk you through the Custom Block creation feature step-by-step from the source code till generation of the imported ***"calculating_roots"*** block. You can skip over to the **"Simulate the Design"** section if you'd like

1. In the MATLAB Command Prompt, navigate to:
   **/home/user/xdf_labs/xmc_ultra96_lab/src/import**
2. Double click on ***"calculating_roots.h"*** file which opens the C source code in MATLAB editor.
3. Observe the top function name defined as ***"calculating_roots"*** – function arguments defined with "const" qualifier are inferred as Input ports and rest as Output ports.
4. Please note that Model Composer does not further optimize the code being imported so if you need to tune the custom block for performance, you would have to insert the necessary Vivado HLS pragmas (performance directives) into your custom code, like one used in this code.

   a. `#pragma HLS pipeline`

5. The rest of the code implements the mathematical equations to calculate the Eigen values using the formula

a. $$\lambda_i = \frac{a+c}{2} \pm \frac{\sqrt{4b^2+(a-c)^2}}{2}; i = 1,2$$

b. Where a,b,c are the values from

c. $$\begin{bmatrix} a & b \\ b & c \end{bmatrix} = \begin{bmatrix} \sum W^2 I_x^2 & \sum W^2 I_x I_y \\ \sum W^2 I_y I_x & \sum W^2 I_y^2 \end{bmatrix}$$

6. Finally the calculated Eigen values are compared with a threshold, $\tau$ which decides the optical flow values in direction x and y ($V_x$ and $V_y$) in this case.

Now, let us see how to generate the custom block from the source code and add it to the existing design.

1. Double-click on the ***"import_function.m"*** from current folder in MATLAB which opens in MATLAB editor.
2. Model Composer provides the ***xmcImportFunction*** command, for use from the MATLAB command line, to let you specify functions defined in source and header files to import as custom Model Composer blocks
3. To learn how to use the ***xmcImportFunction*** function, type ***help xmcImportFunction*** at the MATLAB command prompt to view the help text and understand the function signature.
4. Open the ***import_function.m*** MATLAB script to see how to use ***xmcImportFunction*** to create the ***calculating_roots*** block

```
xmcImportFunction('OpticalFlowRoots',{'calculating_roots'},'calculating_roots.h',{},{'$XILINX_VIVADO_HLS/include'})
```

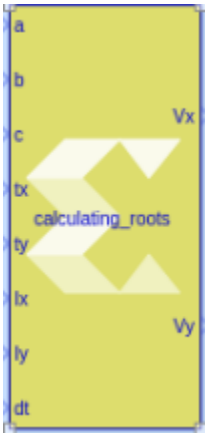
The information specified is as follows:

- **Library_Name: "**OpticalFlowRoots". This is the name of the Simulink library that is created with the new block.
- **Function_Names: '**calculating_roots'. This is the name of the function that you want to import as a block.
- **Header_File: '**calculating_roots.h'. This is the header file for the function.
- **Source Files:** This argument is used to specify the source file for the imported function. In this example all the functionality is defined in header file and hence you can leave it as {} which indicates none.
- **Search_Paths**: '$XILINX_VIVADO_HLS/include'. This argument is used to specify the search path(s) for header files.

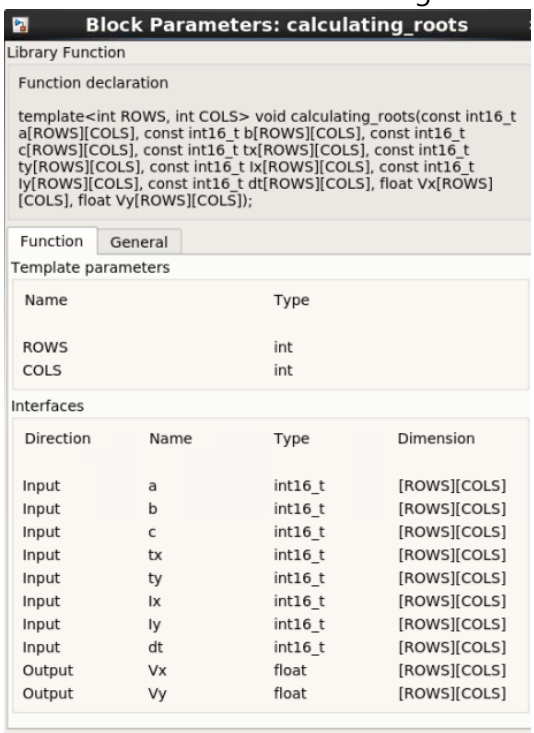
5. Run the ***import_function.m*** script from the MATLAB command line:

```
>>run('import_function.m')
```

6. The Simulink library model opens up with the generated block ***'calculate_roots'*** as shown below.




7. Save this Simulink library model.
8. Double-click the ***calculating_roots*** block, and look at the block interface generated
9. The following figure shows the Block Parameters dialog box for ***calculating_roots***




10. You can add this block into the xmc_optical_flow.slx file in the following location: **/home/user/xdf_labs/xmc_ultra96_lab/src/import**

## Simulate the Design

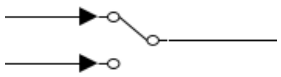
Now that we have walked through how easy it is to design a high-level frame-based implementation of the Lucas-Kanade Dense Optical Flow algorithm within Simulink using the Xilinx Model Composer blocks, the next step is to simulate the design and verify the functional correctness of the algorithm.

With close integration with Simulink and ability to seamlessly connect Model Composer designs to blocks from add-on toolboxes like the ***Computer Vision System Toolbox*** and ***Image Acquisition Toolbox***, a rich simulation and visualization environment is available for both a qualitative and quantitative analysis of algorithms and system-level designs

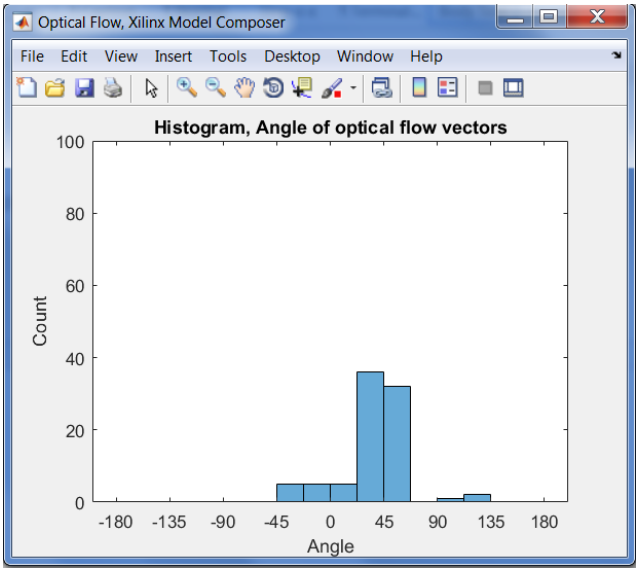
1) On the model, Select the **Simulation > Run** command or click the  button to simulate the design and view the results of the Lucas-Kanade Dense Optical Flow Algorithm.

   The results of the algorithm should look like as shown below where the movement of the object or person in a consecutive frame is visualized in colors (shown in right). Colors indicate the angle of movement.


2) To do a quantitative analysis and verify if the flow vectors generated at the output represent the direction and magnitude of motion correctly, you can use the ***"Moving Object"*** subsystem as the input to the ***"Lucas-Kanade"*** subsystem. Double click on the ***"Manual Switch"*** block to flip the input


Manual Switch

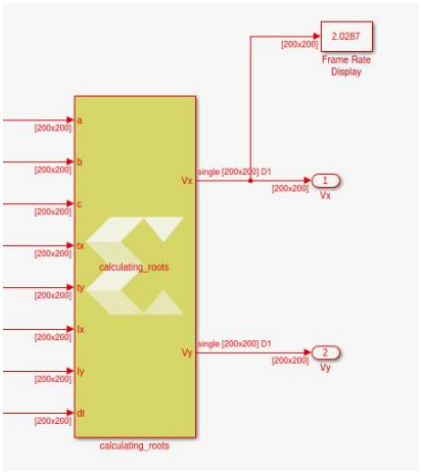
   Now simulate the model once again. For each frame, we plot the histogram of the angles of each vector with a non-zero magnitude. As you turn the dial, you would see the angle of the vectors change.


3) To assess the simulation performance of the algorithm you can check the video frame rate of the simulation. To do this:

   a. Add the **Frame Rate Display** block from the Simulink **Computer Vision System Toolbox** (under the **Sinks** category) and connect it to the output of the algorithm as shown below.

   b. Simulate the model again to see the number of video frames processed per second.




## Optional: Simulating the algorithm with live video from the USB camera

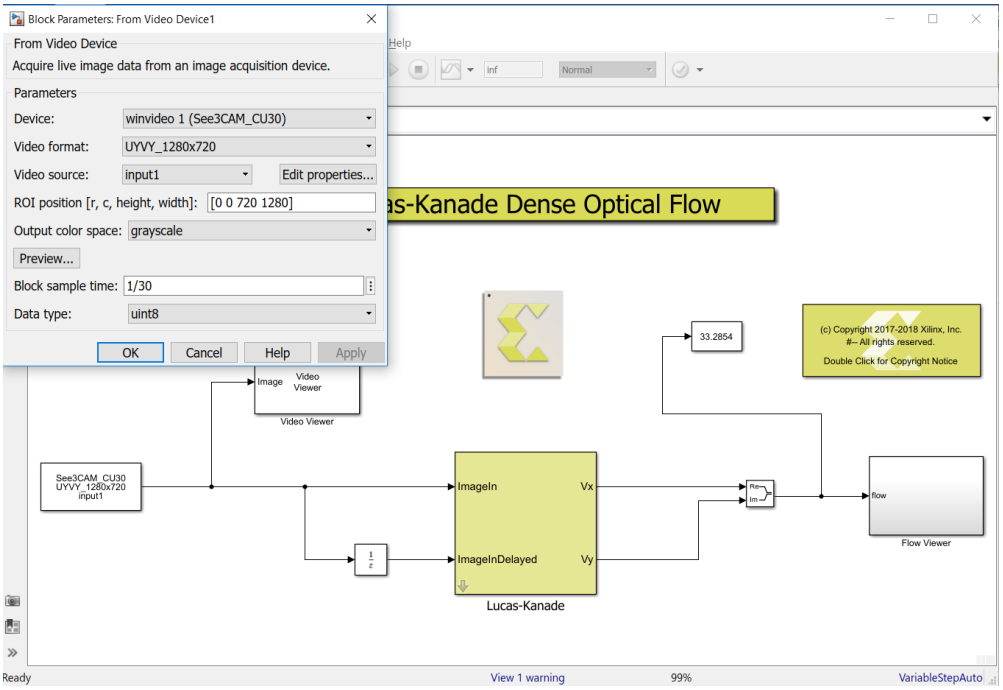
In addition to streaming a video from a multimedia file as input to your Model Composer design in Simulink, you can also stream live video from the **e-con USB camera** into Simulink using the **"From Video Device"** block from the **Image Acquisition Toolbox**. This lets you simulate and functionally verify your algorithms within Simulink using input from the same video device that you will use on the ***Avnet Ultra96*** board.

**Note:** If you'd like to walk through the steps below, you will need to connect the e-con USB camera to the lab machine that you are working on. Reach out to one of the lab assistants in the room to request for one.

1. Connect the e-con USB3 camera to the lab machine
2. Ensure MATLAB can see the USB camera connected to the lab machine – you can verify this by typing the following at the MATLAB Command Prompt and verify that the camera shows up in the list

   **>> webcamlist**

3. At the MATLAB Command Prompt, type **"imaqtool"** to bring up the Image Acquisition Tool dialog. It will display all the webcams and the supported resolutions. You could also use this tool to preview or acquire image/video.
4. Now, open the ***"xmc_optical_flow_webcam.slx"*** example design
5. Note that this is the same Model Composer Optical Flow design but it uses a ***"From Video Device"*** block as the Source block at the input, instead of the "From Multimedia File" block
6. Double click on the ***"From Video Device"*** block and confirm that you have the correct camera selected under the ***"Device"*** parameter. Click **OK**.
7. Now simulate the design and observe the output of the algorithm with the live video input from the USB camera



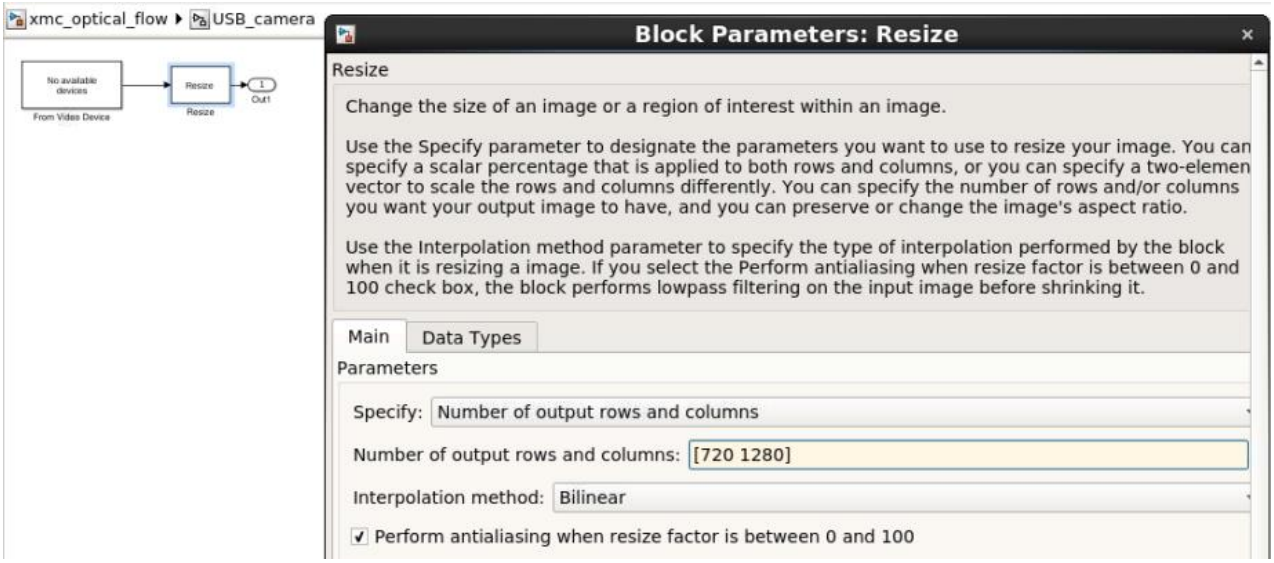
# STEP 2: AUTO-GENERATE THE OPTIMIZED CODE FOR ACCELERATION ON PROGRAMMABLE LOGIC

In Step 1, you learnt how you can leverage Xilinx Model Composer to create a high-level implementation model for the algorithm that you wish to accelerate on the FPGA in the Avnet Ultra96 board.

*Once you have designed, simulated and completed functional verification of your algorithm within the Simulink environment, what is the next step towards deploying this algorithm on the Avnet Ultra96 board?*

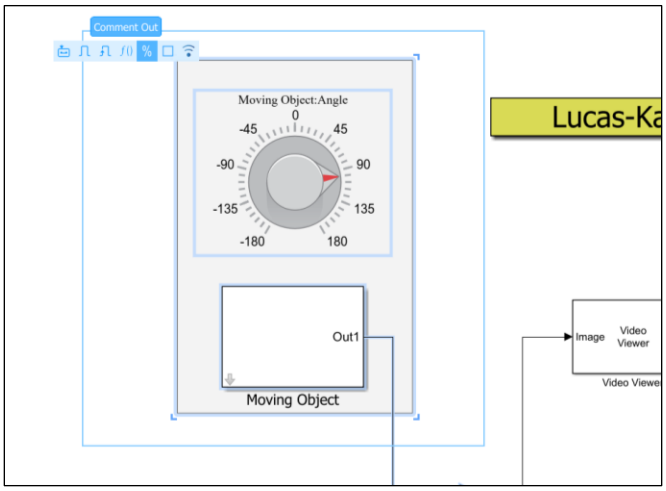
In **Step 2**, we will walk through how Model Composer can automatically generate the optimized HLS-synthesizable code for the accelerator that can be further integrated into the SDSoC Development Environment and deployed on the Avnet Ultra96 board

1) In the **"xmc_optical_flow.slx"** design, go to the top-level of the model
2) Before generating code from the design, we will make a few changes to the design – For example, we want to change the resolution of the input to the design to HD quality.
    a. Double-click the ***"Resize"*** block at the Input and change the number of output rows and columns.

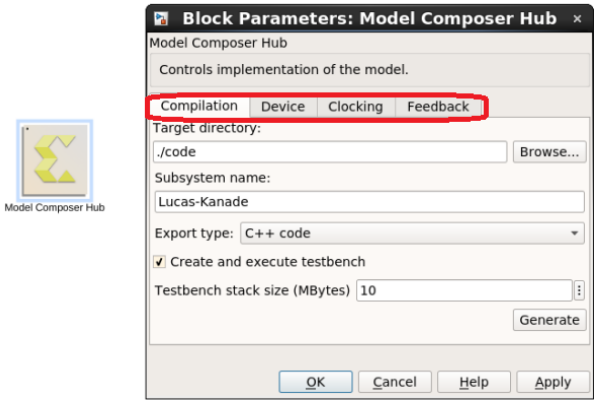



    b. Change the simulation time from "inf" to "0".
    c. **IMPORTANT:** Comment out the "Moving Object" Subsystem in the design – We will only use the video input from the ***"From Multimedia File"*** block

To comment it out, select the area around it and click % to comment it out or Right click on the selection and click "Comment Out"




3) For code generation, the ***Model Composer Hub*** block from the ***"Tools"*** sub-library is what controls the implementation of the algorithm designed using Model Composer blocks.
4) Double-click on the Model Composer Hub block and you can see dialog block has four tabs.




5) **Compilation Tab** (Note: This block has already been configured with the necessary settings)

a. **TARGET DIRECTORY** :Defines the output folder where the compiled results will be written *( Set to **"code"** directory)*
*b.* **SUBSYSTEM NAME** : Specifies the name of the Model Composer subsystem located in the top-level of the model that is required to generate output *( Set to **"Lucas-Kanade"** )*
c. **EXPORT TYPE**: You can convert the design captured within Simulink to packaged IP to add to the Vivado Design Suite IP catalog, System Generator solution for use in RTL level block design, and optimized C++ code for use in Vivado HLS. *( Set to **"C++"** )*
d. **CREATE AND EXECUTE TESTBENCH**: When this checkbox is selected, Model Composer automatically logs the input test vectors and the expected outputs from the design during

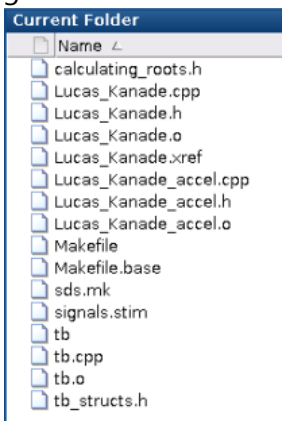
simulation and generates a test bench to verify equivalence of the generated code with the design modeled and simulated in Simulink

*To handle the large signal arrays allocated on the stack and deep nesting of sub systems, the size of the stack has been set to '200'*

6) The Model Composer feature to automatically generate code for integration into the SDSoC Development Environment is done through command line, as this option is not integrated into Model Composer Hub block yet
7) Type ***xmcGenerateForSDSoC('xmc_optical_flow','zcu102')*** in the MATLAB console to generate HLS-optimized code from the ***"Lucas-Kanade"*** subsystem that can be accelerated on the Programmable Logic through the SDSoC Development Environment
8) Messages and instruction should be printed in the console if code is generated successfully. *You don't need to follow the instruction from the MATLAB console. You will be guided through the SDx GUI flow in Step 3 and 4 to add and compile the generated code for the Avnet Ultra96 board*

```
Unset Model Composer preference 'enable_sdsoc'
Unset Model Composer preference 'sdsoc_platform'
To compile the generated code with SDSoC for emulation:
1. Open terminal
2. source ${XILINX_VIVADO_SDX}/.settings64.csh
3. cd /tmp/optical_flow_6/code
4. make all TARGET=emu -f sds.mk
5. make check TARGET=emu -f sds.mk

To compile the generated code with SDSoC for hardware:
1. Open terminal
2. source ${XILINX_VIVADO_SDX}/.settings64.csh
3. cd /tmp/optical_flow_6/code
4. make all TARGET=hw -f sds.mk

For further information: make help -f sds.mk
```

9) A ***"code"*** directory is created (as specified by the Model Composer Hub block) with the generated code for SDSoC from the top-level "Lucas-Kanade" subsystem that can be accelerated on the Programmable Logic




10) In the code generated by Model Composer, the directives to guide the SDSoC System Compiler are automatically inserted into source code
11) Double-click on the ***Lucas_Kanade_accel.h*** and notice the following SDS pragmas (directives for the SDSoC System Compiler)

a. ***#pragma SDS data copy*** - implies that data is explicitly copied between the host processor memory and the hardware function.
b. ***#pragma SDS data mem_attribute*** - SDSoC runtime provides API to allocate physically contiguous memory. This pragma can be used to tell the compiler whether the arguments have been allocated in physically contiguous memory.
c. ***#pragma SDS data access_pattern*** - This pragma specifies the data access pattern in the hardware function. If the access pattern is SEQUENTIAL, a streaming interface (such as ap_fifo) will be generated. Otherwise, with RANDOM access pattern, a RAM interface will be generated.

12) You should now have everything you need to implement the Lucas-Kanade Dense Optical flow algorithm in hardware as an accelerator using the SDSoC Development Environment

# STEP 3: ADD MODEL COMPOSER GENERATED ACCELERATOR INTO SDSOC PROJECT

## QUICK DEFINITIONS

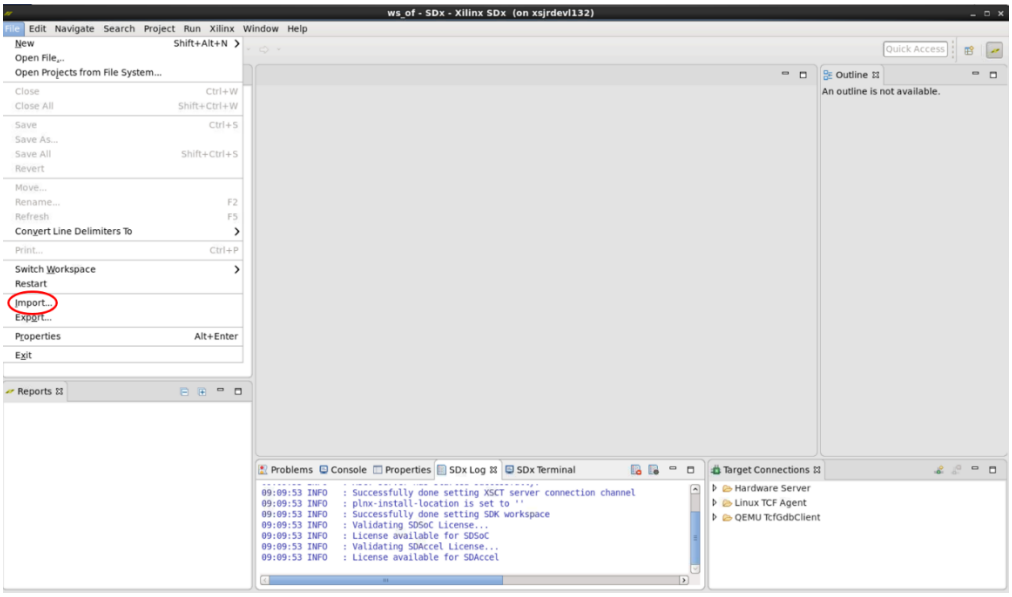
- **SDSOC DEVELOPMENT ENVIRONMENT:** Using SoC devices from Xilinx, such as the Zynq-7000 SoC and the Zynq UltraScale+ MPSoC, you can implement elements of your application into hardware accelerators, running many times faster than optimized code running on a processor.

  The SDSoC™ environment is a tool suite for building efficient system-on-chip (SoC) applications, starting from a platform that provides the base hardware and target software architecture. A boot image and the executable application code are generated by the SDSoC tools. The SDSoC system compiler employs underlying tools from the Vivado Design SuiteHLS Editions including Vivado® HLS, IP integrator, and IP libraries for data movement and interconnect, and the RTL synthesis, implementation, and bitstream generation tools.

- **SDSOC PLATFORM:** An SDSoC platform consists of a hardware portion defining the embedded processor, the hardware function, and any peripherals supported by the platform; and a software portion defining the operating system boot images, drivers, and the application code.

Now that we have the generated code from Model composer, the next step is to add the accelerator into SDSoC project to build the application to run on the Ultra96 board

1) Set the SYSROOT environment variable to point to a directory inside the platform.
   a. Open another Terminal
   b. **export SYSROOT=/opt/Xilinx/sysroots/aarch64-xilinx-linux**
2) Invoke SDSoC
   a. cd **/home/user/xdf_labs/xmc_ultra96_lab/wrk/ultra96-rv-ss-2018-2/workspaces/ws_of**
   b. **sdx -workspace . &**
3) Close the Welcome Screen.
4) Import the existing gstreamer workspaces.
   a. select **'File'→'Import'→'General'→'Existing Projects into Workspace→'Next'**



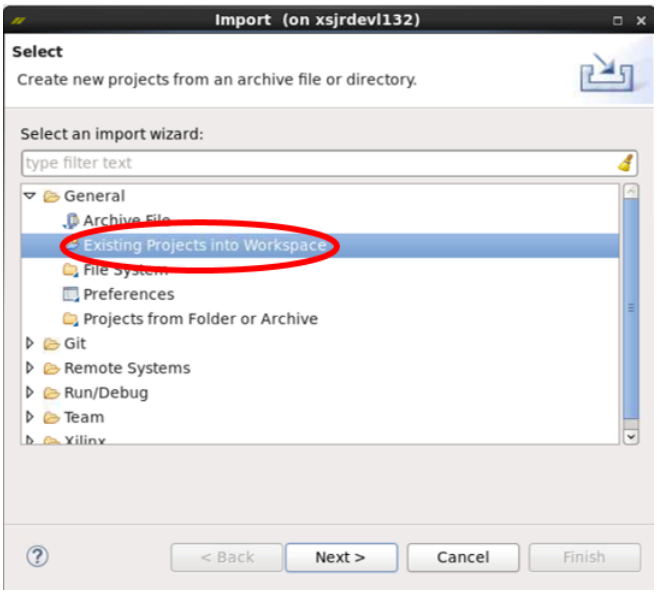




**Figure: Importing Existing Projects**

b. In the **Import** dialog, to the right of '**Select root directory**', click '**Browse**'.

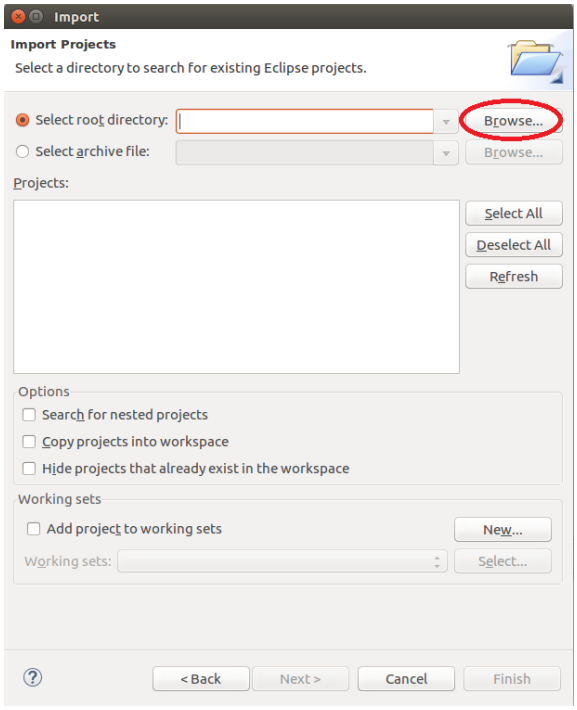



**Figure: Import Projects**

c. By default you are already in the directory you want **/home/user/xdf_labs/xmc_ultra96_lab/wrk/ultra96-rv-ss-2018-2/workspaces/ws_of** , just click 'OK'.

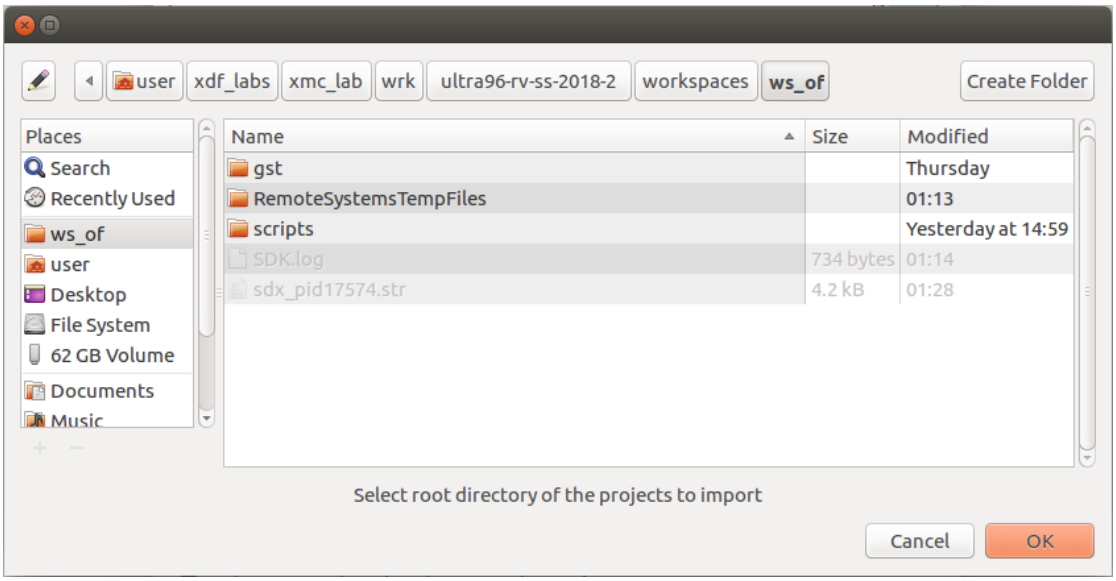



**Figure: Choosing location for Existing projects**

d. You should see a list of projects, with ***gstdemo, gstsdxallocator, gstsdxbase***, and ***gstopticalflow*** selected, click '***Finish***'.

These are Gstreamer projects that you need to work with the interfaces in the Ultra96 reVISION platform. GStreamer is an open-source shared library responsible for initializing the capture, memory-to-memory, and display pipelines as well as managing the video buffer flow through the pipeline stages.

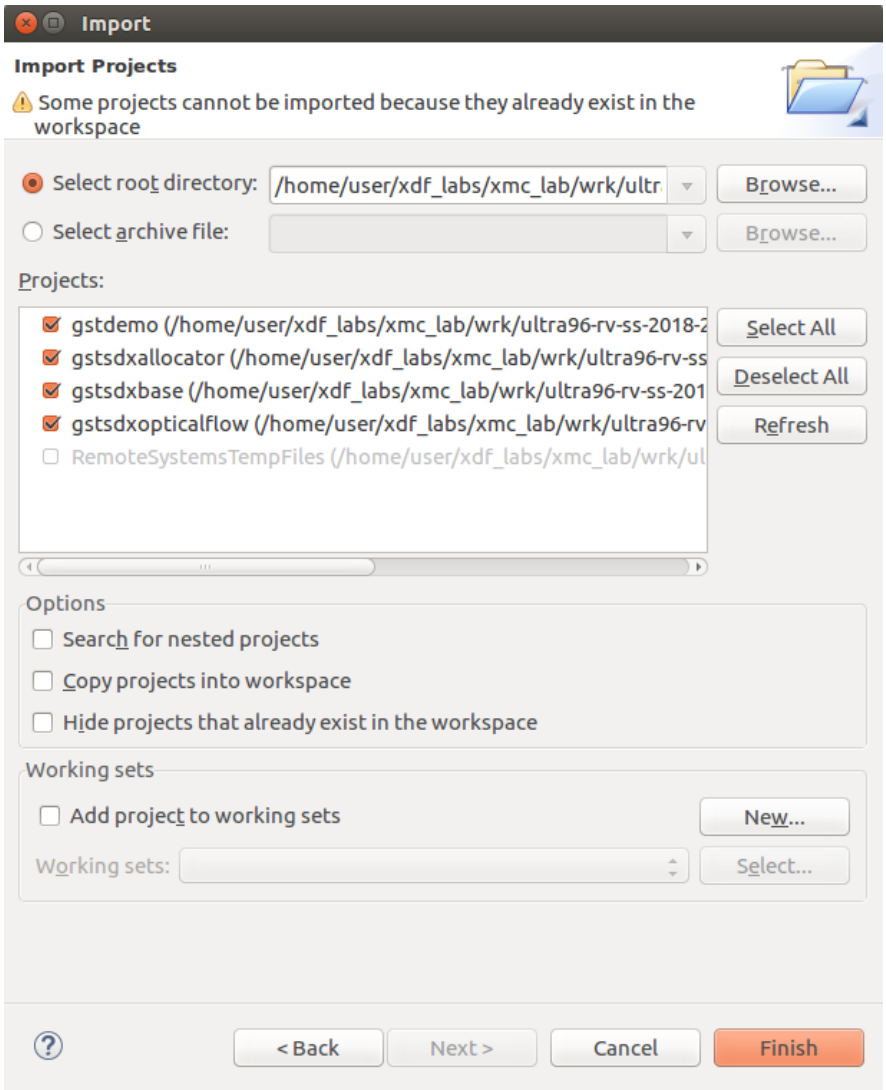



**Figure: Importing the Gstreamer Library source files**

5) Click **Finish**.

6) Create a new SDx Project based on **Dense Optical Flow Live IO** Template.

a. Now select **'File'→'New'→'SDx Project'**... from the menu bar.

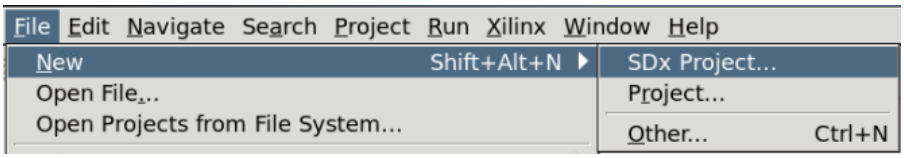



**Figure: Creating New SDx Project**

b. Select **Application Project** and click **'Next'**.

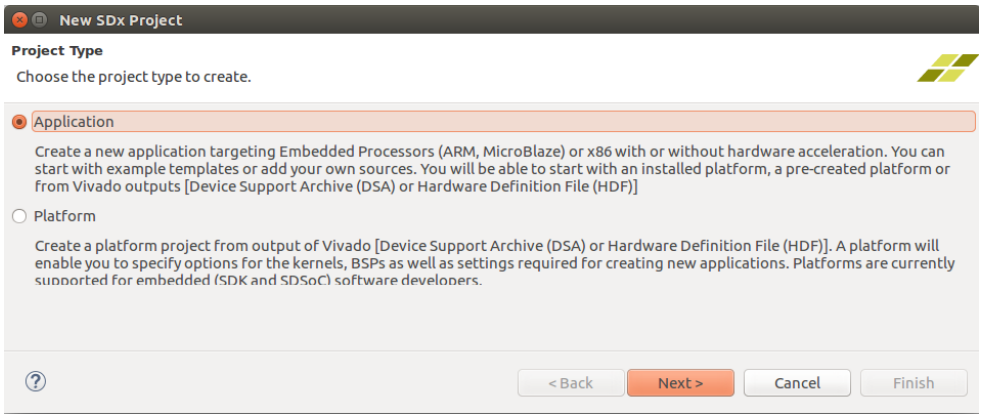
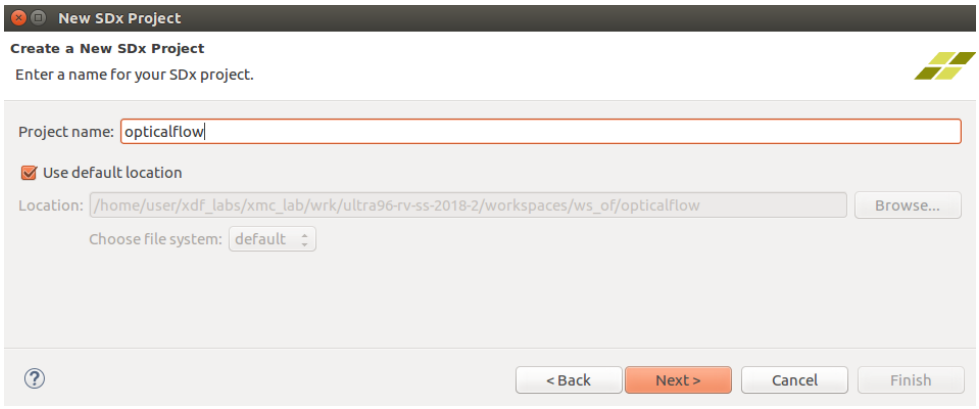
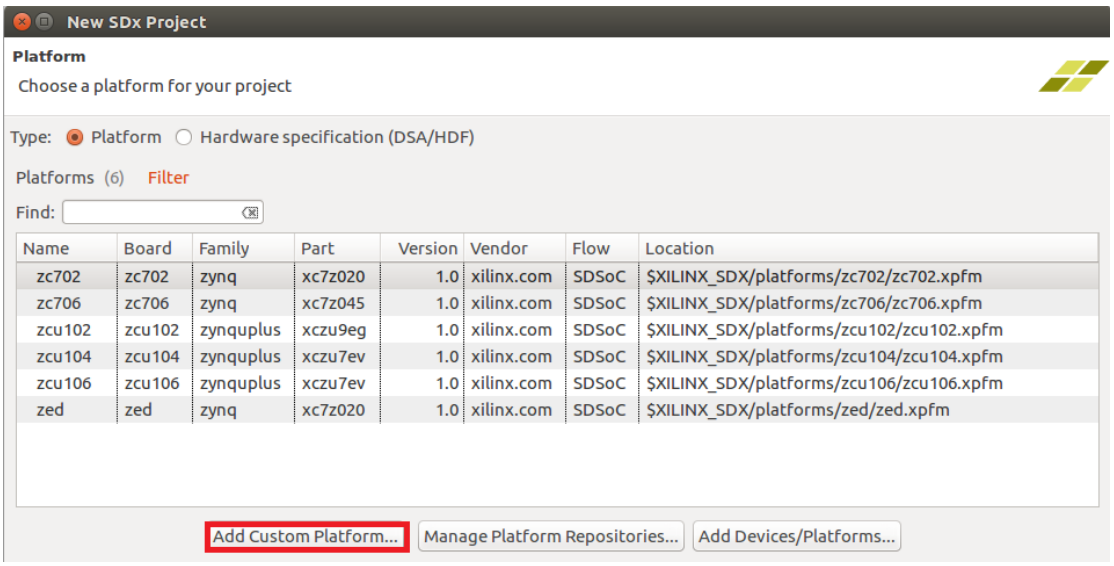


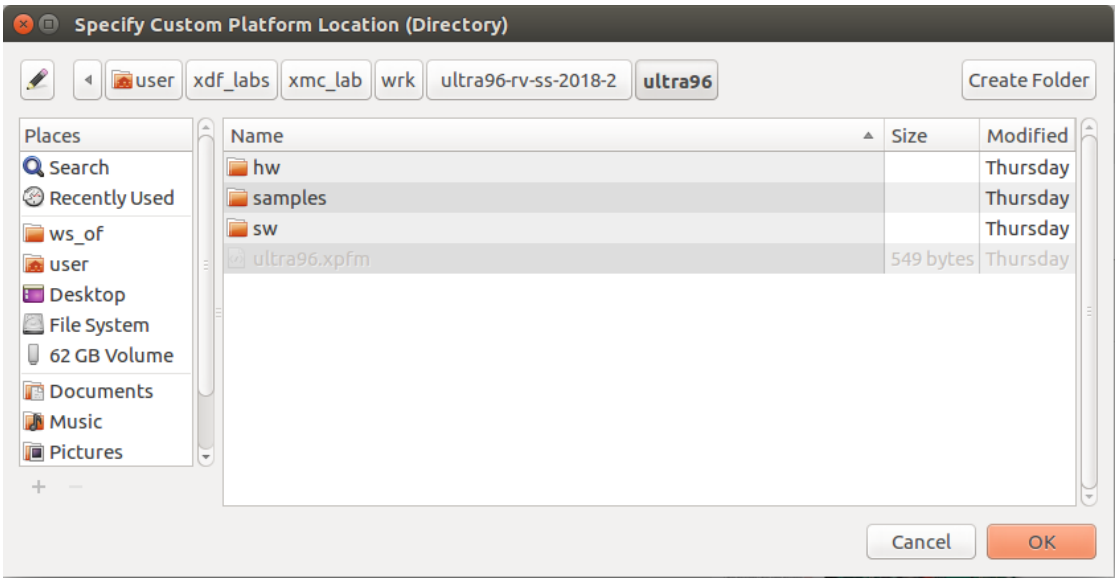

**Figure: Choosing Application as project type**

c. In the ***'Create a New SDx Project'*** dialog, enter Project name ***'opticalflow'***, click ***'Next'***.

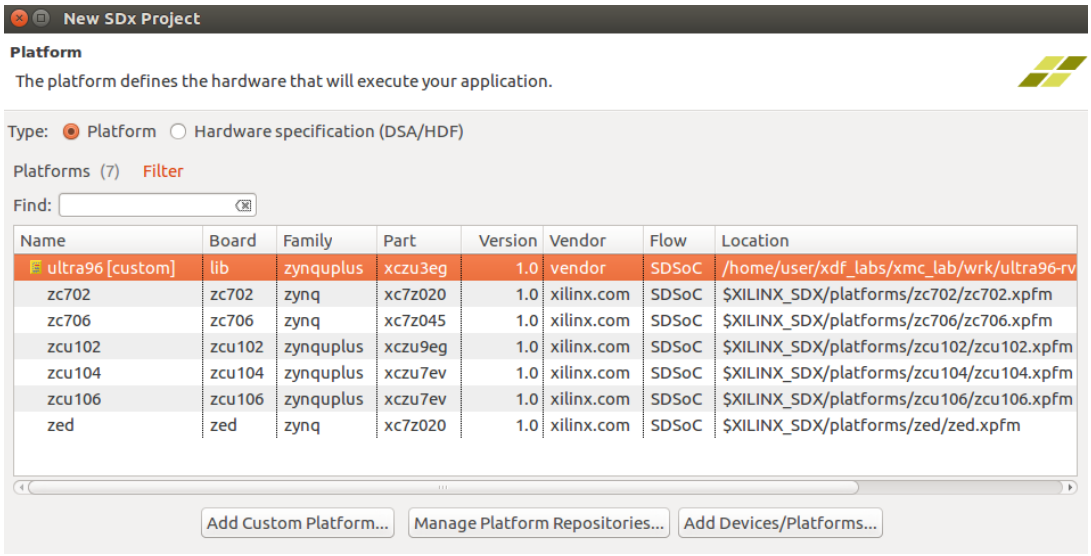


7) In the Platform dialog, click 'Add Custom Platform', then select **/home/user/xdf_labs/xmc_ultra96_lab/wrk/ultra96-rv-ss-2018-2/ultra96** directory where your ultra96 reVISION platform file is located. Click 'OK'.

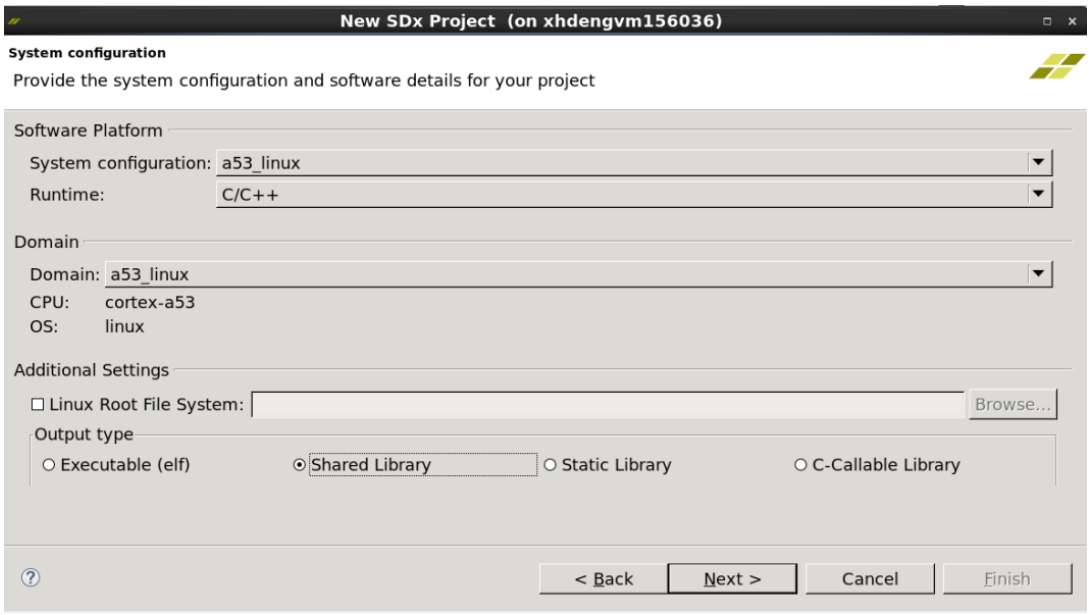



a. Back in the Platform dialog, the new platform appears in the list, but is not selected. Select it (as shown in the figure below), then click '***Next***'.

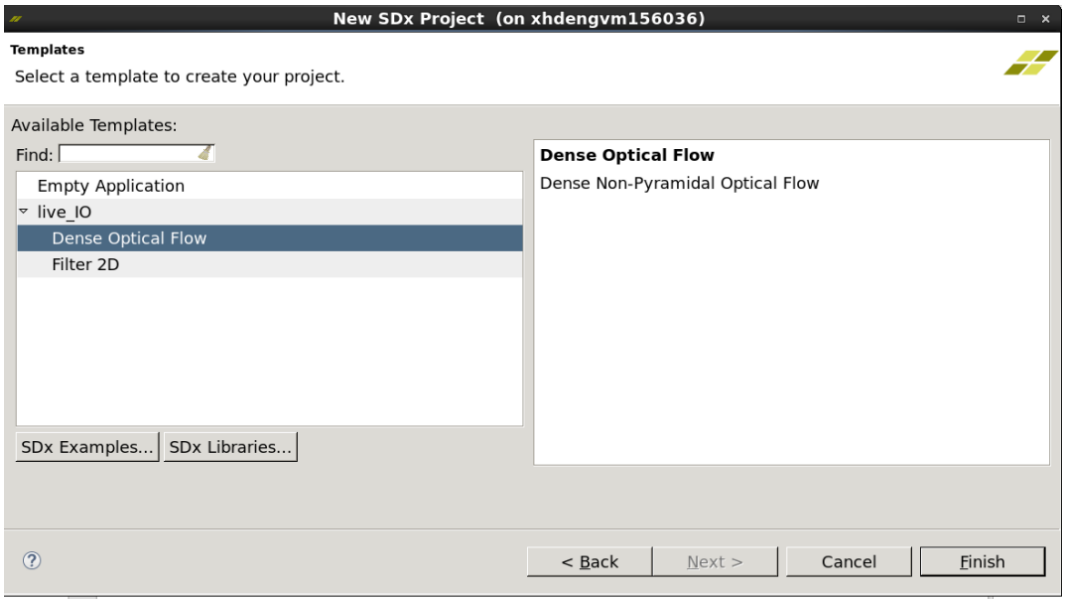



b. In the System configuration dialog, under **Output type**, select **'Shared Library'**, click **'Next'**




c. In the Templates dialog, under live_IO select **Dense Optical Flow** and click **'Finish'**.

This opens up an example Live I/O application for the Ultra96 reVISION platform that uses the Xilinx-optimized xfOpenCV library function as the hardware accelerator – The steps that follow outline how you can replace this with a custom accelerator auto-generated by Model Composer




8) Back again at the main window, the new project '***opticalflow***' appears in the Project Explorer pane.

   You should also be see the opticalflow ***Application Configuration settings*** in the main window.

9) Switch the '**Active Build Configuration**' for the opticalflow project to **Release**. Note that three routines are marked as Hardware Functions.
10) Select the **Data motion network clock frequency** to 300 MHz

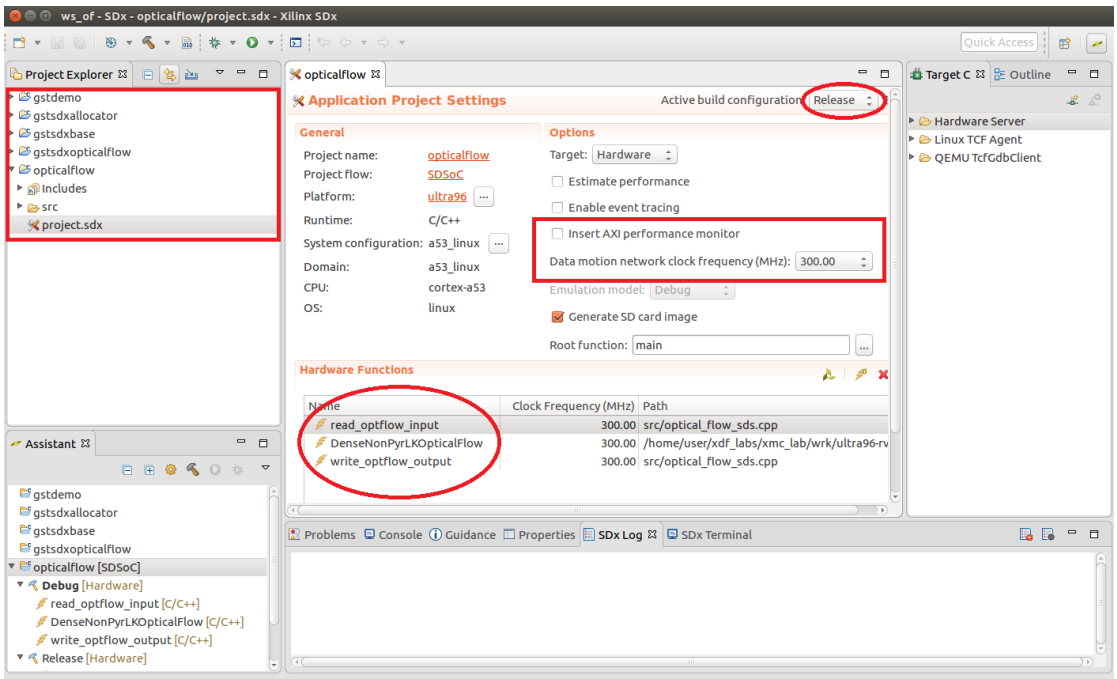
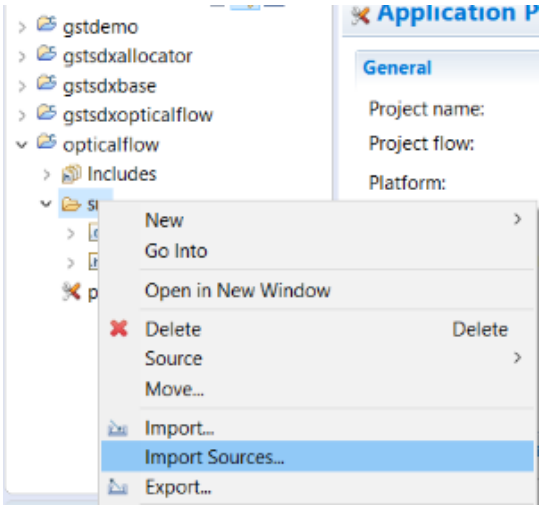


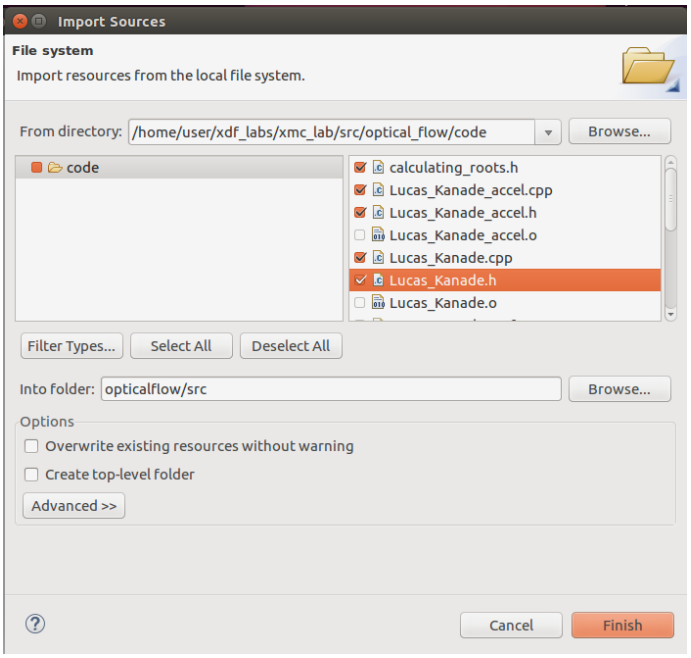

**Figure : Opticalflow Application Configuration settings**

11) Import the Model Composer optical flow accelerator code

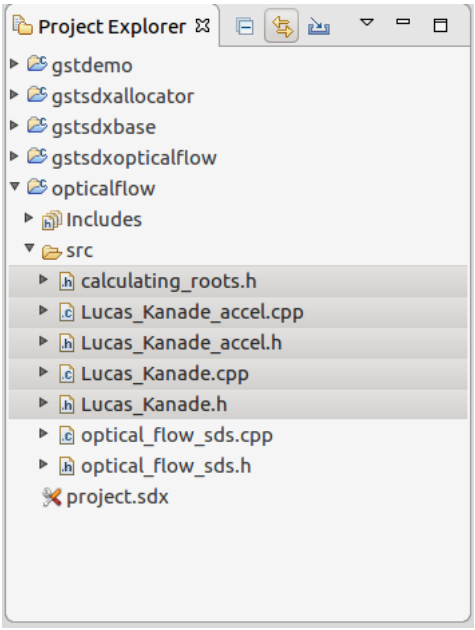
a. Select the **"src"** folder under the ***"opticalflow"*** project in the Project Explorer. Right click and select ***"Import Sources..."***




b. From the ***"Import Sources"*** dialog, select the Model Composer optical flow code directory (path as shown in the screenshot below) and choose the necessary files (checked in the screenshot below). Click ***Finish***




c. The Model Composer generated code for the ***"Lucas-Kanade Optical Flow"*** algorithm should be added to the project.




d. Double click and open the ***optical_flow_sds.cpp*** file and add the following lines of code to include the Model Composer generated header file and define parameters for the width and height of the Input

```
#include "Lucas_Kanade_accel.h"
#define HD_HEIGHT   720
#define HD_WIDTH    1280
```

```
35
36 #include "optical_flow_sds.h"
37
38 // Model Composer define and source header
39 #include "Lucas_Kanade_accel.h"
40 #define HD_HEIGHT   720
41 #define HD_WIDTH    1280
```

e. Change the ***optical_flow_sds.cpp*** to call Model Composer generated ***"Lucas-Kanade Dense Optical Flow"*** accelerator function instead of the ***xf::DenseNonPyrLKOpticalFlow*** function

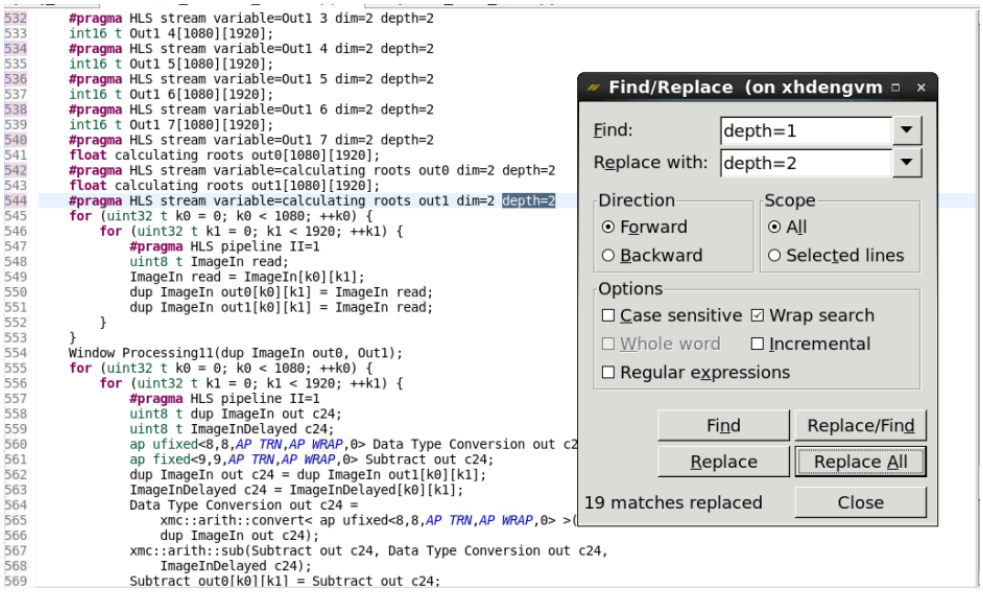
```
Lucas_Kanade_accel(reinterpret_cast<uint8_t(*)[HD_WIDTH]>(ofd->luma_curr->data),

reinterpret_cast<uint8_t(*)[HD_WIDTH]>(ofd->luma_prev->data),

reinterpret_cast<float(*)[HD_WIDTH]>(ofd->flow_x->data),

reinterpret_cast<float(*)[HD_WIDTH]>(ofd->flow_y->data));
```

```
279     read optflow input(f prev, f curr, *ofd->luma prev, *ofd->luma curr, ofd->in fourcc, pcnt);
280     //xf::DenseNonPyrLKOpticalFlow<OF WINDOW SIZE, XF 8UC1, OF HEIGHT, OF WIDTH, OF PIX PER CLOCK>(*ofd->luma prev, *ofd->l
281     Lucas Kanade accel(reinterpret_cast<uint8 t(*)[HD WIDTH]>(ofd->luma curr->data),
282             reinterpret_cast<uint8 t(*)[HD WIDTH]>(ofd->luma prev->data),
283             reinterpret_cast<float(*)[HD WIDTH]>(ofd->flow x->data),
284             reinterpret_cast<float(*)[HD WIDTH]>(ofd->flow y->data));
285     write optflow output(*ofd->flow x, *ofd->flow y, f out, ofd->out fourcc, pcnt);
```

f. Save ***optical_flow_sds.cpp***

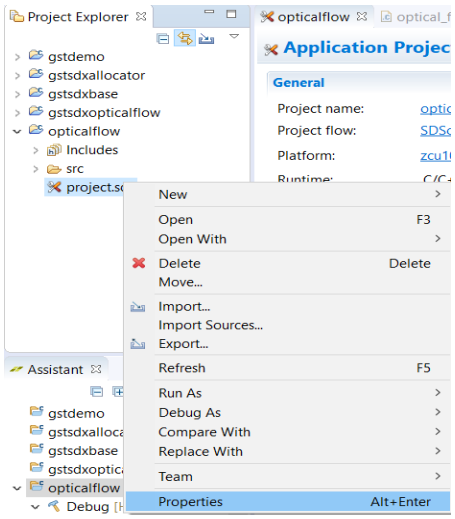
g. The following step is due to a bug in 2018.2. In the Model Composer generated code, the FIFO depth gets set to 1 in our generated code while the depth should be 2. This has been fixed in 2018.3 and should not need to be performed manually for our generated code. The depth=2 will give the accelerator a better performance.

   In the ***Lucas_Kanade_accel.cpp*** file, find and replace ***"depth=1"*** with ***"depth=2"***

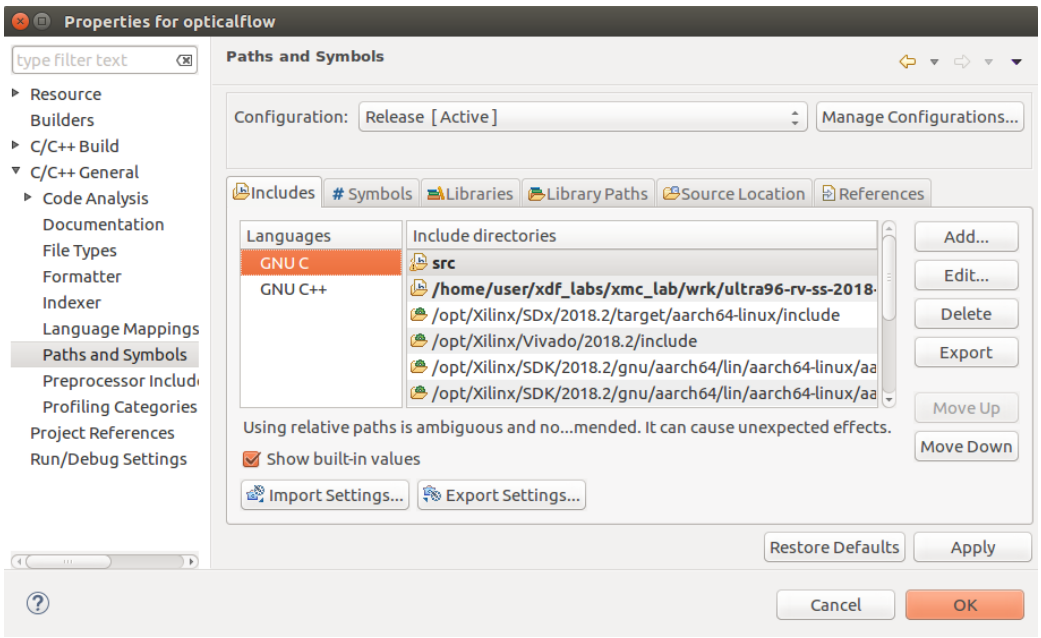



12) Add the Model Composer include/tb directory into the C++ include search path. Make sure it is pointing to 2018.2 since Model Composer code is generated from 2018.2. and add it in both "GNU C" and "GNU C++"

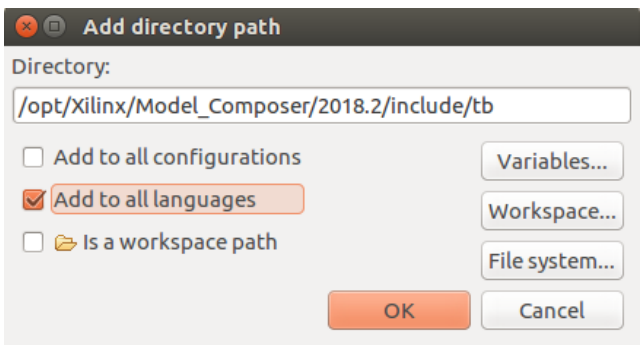
   a. Select ***"project.sdx"***, right click and select ***"Properties..."***




b. Expand C/C++ General Drop down on the left, Select ***"Paths and Symbols"*** and in the ***"Includes"*** tab, Select ***"GNU C"*** and click the ***"Add"*** button



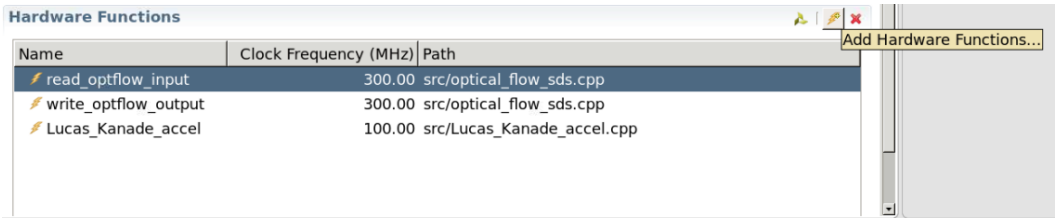

c. Add **/opt/Xilinx/Model_Composer/2018.2/include/tb** path and check ***"Add to all languages"*** to add to both "GNU C" and "GNU C++" and ***click OK***



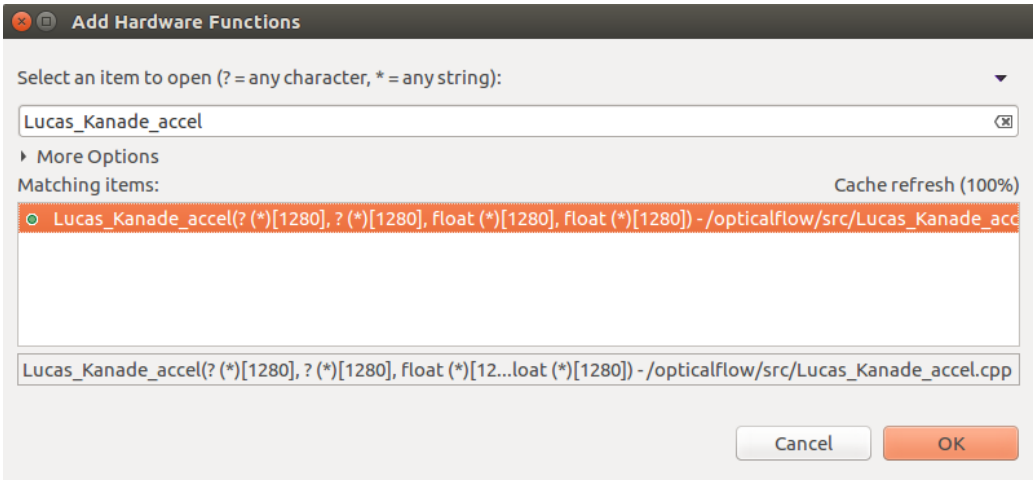
d. You should see the **/opt/Xilinx/Model_Composer/2018.2/include/tb** added in the Include directories. Click OK to exit from properties dialog. ***If a question dialog opens up to ask if you to rebuild the index, click No.(We are yet to specify the hardware function)***

13) Specify the Model Composer generated accelerator function ***"Lucas_Kanade_accel"*** as Hardware function in ***"Project Setting"***

a. Right-click on ***"DenseNonPyrLKOpticalFlow"*** Function and remove it.
b. Click Add Hardware Function button.




c. From the "Add Hardware Functions" window, select the "Lucas_Kanade_accel" and click OK.



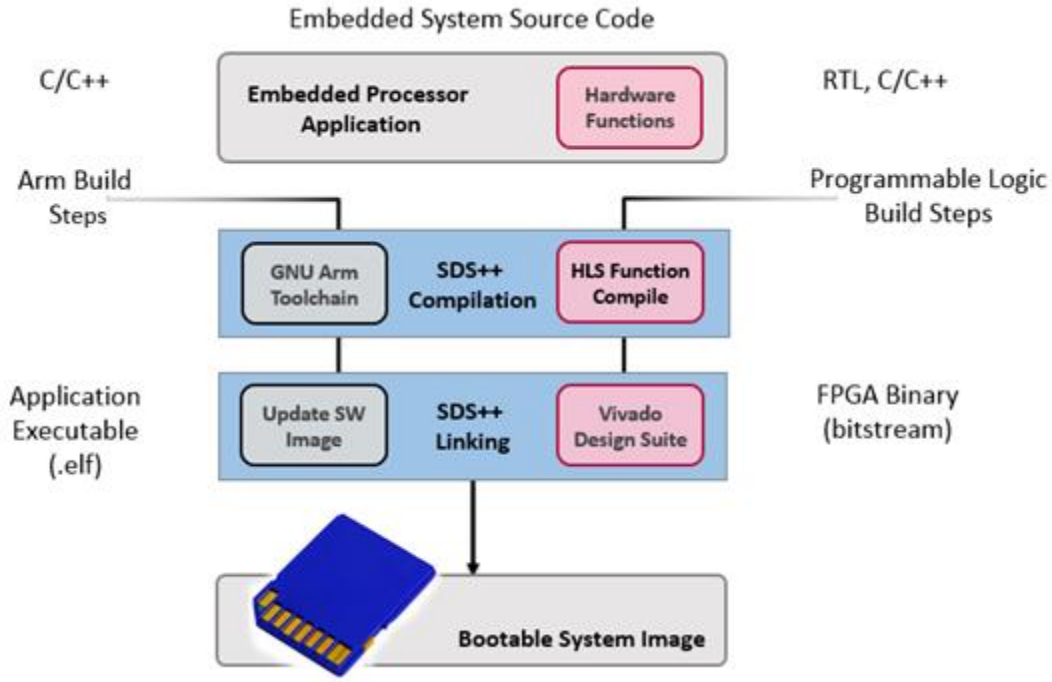

d. Observe the ***"Lucas_Kanade_accel"*** function got added to the list.
e. Change the ***"Lucas_Kanade_accel"*** function Clock Frequency to 300 MHz (Click on the clock frequency and select from the drop-down menu)

# STEP 4: BUILD THE PROJECT FOR THE AVNET ULTRA96 REVISION PLATFORM

***Note:*** *In the interest of time, we have compiled the application using the steps below and have copied the compiled application into SD-cards available – Please ask one of the lab assistants for the* ***pre-compiled SD-cards*** *that you could use to plug into one of the Ultra96 boards available and view the results of the acceleration as outlined in Step 5.*

## SDSOC BUILD PROCESS

The SDSoC system compiler invokes all necessary tools, including Vivado HLS (function compiler), the Vivado® Design Suite to implement the generated hardware system, and the Arm® compiler and linker to create the application binaries that run on the CPU, invoking the accelerator (stubs) for each hardware function by outputting a complete bootable system for an SD card.

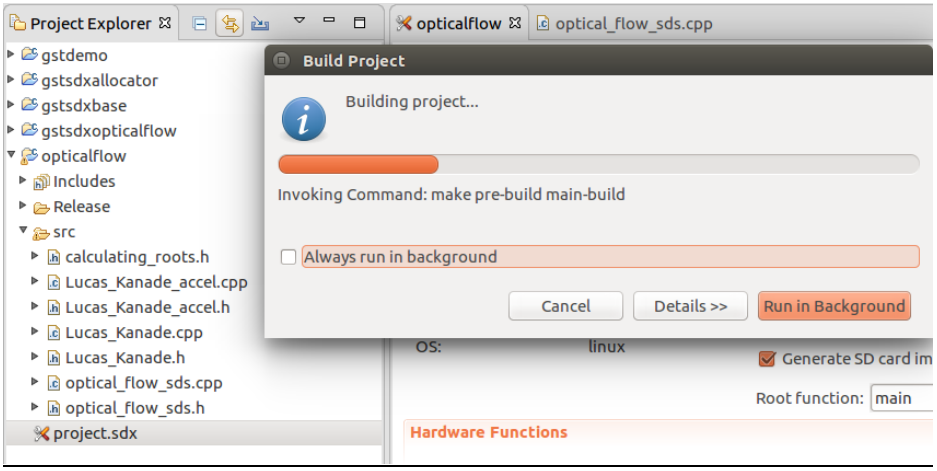



If you'd like to walk through the steps to compile the project on your own, please follow the step-by-step instructions outlined below

1) To build the project, right click on the ***opticalflow*** Project in the Project Explorer and choosing Build Project, or by clicking the '***hammer***' icon at the top

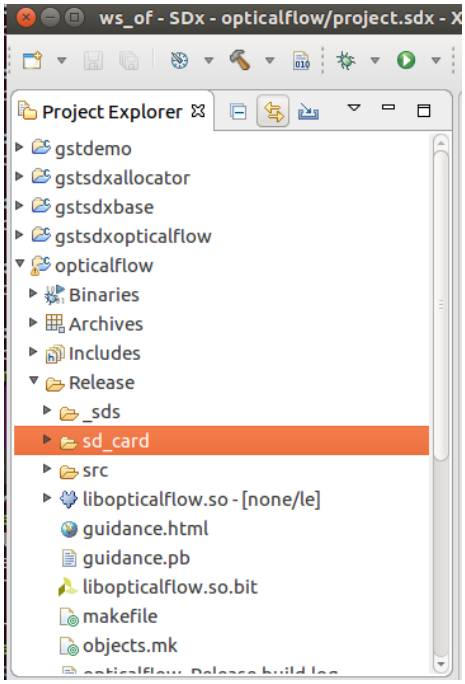
2) In the small Build Project dialog that opens, you may hit the ***"Run in Background"*** button.

- That causes the small dialog box to disappear, though you can still see a progress icon in the lower right part of the GUI, showing that work is in progress.
- Select the Console tab in the lower central pane of the GUI to observe the steps of the build process as it progresses.

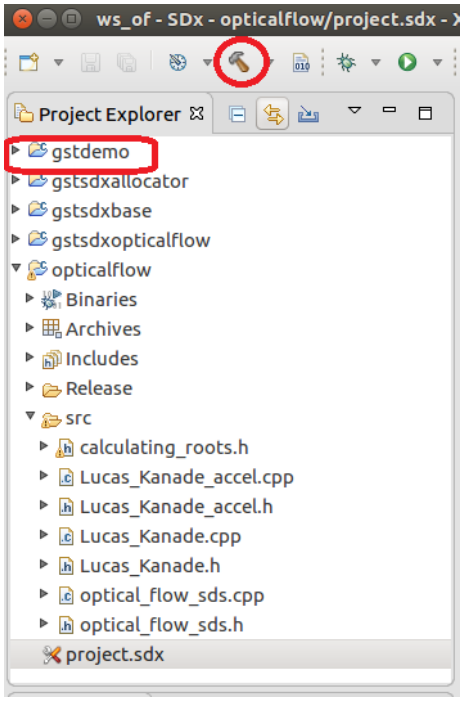



- Generally the build process could take minutes, up to several hours, depending on the power of your host machine, whether you are running on Linux or Windows, and of course the complexity of your design. By far the most time is spent processing the routines that have been tagged for realization in hardware - note the "HW functions" window in the lower part of the SDx Project Settings pane. In our example above, the routines read_optflow_input, Lucas_Kanade_accel, and write_optflow_output are tagged to be built in hardware.
- The synthesis of the C code found in these routines into RTL, and the Placement and Routing of that RTL into the programmable logic in the Zynq MPSoC, are the steps that take most of the time.

3) Once the Build completes, you will find an ***sd_card*** directory has been created containing these files which need to transfer to your SD card.




4) Now that the "bottom" shared library is built, you may build the "top" part, that will be linked with the bottom library. Now select the gstdemo project, and build it. Doing this will build all four of the gst--- projects.



# STEP 5: VIEW THE RESULTS OF ACCELERATION WITH LIVE VIDEO I/O ON THE ULTRA96 BOARD

***Note:*** *To complete this step you would need to connect your Lab laptop to the Avnet Ultra96 board setup, including a monitor that can be connected to the board. We have 5 hardware setups available in this lab - Talk to a Lab assistant*

1. If you would like to create an SD-card, do the following steps (**OR** ask for a pre-loaded SD-card)
    - Open a terminal
    - **cd /home/user/xdf_labs/xmc_ultra96_lab/wrk/ultra96-rv-ss-2018-2/workspaces/ws_of/scripts**
    - **source make_sdcard.sh**
    - This will copy all the necessary files to **/home/user/xdf_labs/xmc_ultra96_lab/wrk/ultra96-rv-ss-2018-2/workspaces/ws_of/opticalflow/Release/sd_card** folder
    - Copy all the files from **/home/user/xdf_labs/xmc_ultra96_lab/wrk/ultra96-rv-ss-2018-2/workspaces/ws_of/opticalflow/Release/sd_card** to the microSD card root directory

**TIP**

- libopticalflow.so, libgstsdxbase.so and libgstsdxallocator.so in /lib directory.
- libgstsdxopticalflow.so in /gstreamer-1.0 directory.
- Rest all images copy directly in sdcard's root folder /.

2. Run the live_IO sample applications.
    - Plug the SD card into the Ultra96 board.
    - Connect the microUSB cable between the laptop and the USB/JTAG Pod
    - Open a new Terminal
    - Type “sudo adduser user dialout” or “sudo usermod -a -G dialout user”
    - Type “sudo putty /dev/ttyUSB0 -serial -sercfg 115200,8,n,1,N” to connect to the board. Enter password: “password” if prompt
    - You should see a PuTTY prompt
    - Power up the Ultra96 board and you should see the Petalinux boots up
    - Login to the terminal using root as the username and password
    - cd to /media/card and run the gstrun.sh script or do the following steps

```
# modetest -M xlnx -s 38@36:1920x1080-60@RG16 -w 35:alpha:0 &
```

- cd over to directory /media/card . This directory contains all the files you copied to your SD card.

```
# cd /media/card/
```

- Copy the shared libraries where they need to go.

```
# cp lib/libopticalflow.so /usr/lib
# cp gstreamer-1.0/libgstsdxopticalflow.so /usr/lib/gstreamer-1.0
# cp lib/libgstsdxbase.so /usr/lib/
# cp lib/libgstsdxallocator.so /usr/lib/
```

- Check if you are able to see the USB Camera output on Display Port Monitor by running the following pipeline.

```
# gst-launch-1.0 v4l2src device=/dev/video0 io-mode=4 ! "video/x-raw, width=1920, height=1080, framerate=30/1, format=UYVY" ! queue max-size-bytes=0 ! kmssink driver-name=xlnx
```

You should be able to see the USB Camera output on the Display Port.

- Press CTRL-C to close the pipeline

- Run the following gst-launch command to run the opticalflow pipeline, from USB Camera, 1920x1080@30fps, UYVY, to DP output.

```
# gst-launch-1.0 v4l2src device=/dev/video0 io-mode=4 ! "video/x-raw, width=1920, height=1080, framerate=30/1, format=UYVY" ! sdxopticalflow filter-mode=1 ! queue max-size-bytes=0 ! kmssink driver-name=xlnx plane-id=34
```

## Key Takeaways

- With Model Composer, you can focus on designing and exploring algorithms that can benefit from acceleration on the Programmable Logic without worrying about the implementation specifics.
- Model Composer enables you to work at a higher-level of abstraction without the need to move to a low-level implementation model using primitives
- Model Composer enables you to create custom blocks by importing existing HLS functions or C/C++ code into existing model composer design.
- Model Composer transforms your algorithmic specifications to production-quality implementation through automatic optimizations that extends the Xilinx High Level Synthesis technology
- You can design and automatically generate optimized code for hardware accelerators that can be integrated with the SDSoC Development Environment for deployment