

# **Vitis™ Video Analytics SDK documentation**

***Release 1.0***

**Xilinx, Inc.**

Aug 04, 2021



**List of Figures** iii

<b>1</b>	<b>VVAS Graph Architecture</b>	<b>3</b>
1.1	Vitis Video Analytics SDK Release V1.0 . . . . .	5
1.2	Plug-ins . . . . .	6
1.3	Development Guide . . . . .	28
1.4	Platforms And Applications . . . . .	39
1.5	VVAS GStreamer Plug-ins for Embedded Platforms . . . . .	61
1.6	Tutorials . . . . .	63
1.7	Supported Platforms And Applications . . . . .	63
1.8	VVAS GStreamer Plug-ins for Data Center Platforms . . . . .	64
1.9	Frequently Asked Questions. . . . .	64
1.10	Why VVAS? . . . . .	65
1.11	VVAS Core Components. . . . .	65



Figure 1: VVAS Block Diagram . . . . .	2
Figure 1.1: VVAS solution for VCU block. . . . .	41
Figure 1.2: Sample video pipeline for VCU block . . . . .	42
Figure 1.3: VVAS solution for Preprocessor block. . . . .	42
Figure 1.4: Sample Video Pipeline for VCU + Preprocessor block. . . . .	43
Figure 1.5: VVAS solution for ML block . . . . .	44
Figure 1.6: Sample Video Pipeline adding ML block . . . . .	44
Figure 1.7: Sample video pipeline adding Meta Affixer and HDMI Tx blocks. . . . .	47
Figure 1.8: Sample Video Pipeline adding Bounding Box block. . . . .	48



The Vitis Video Analytics SDK (**VVAS**) is a framework to build transcoding and AI-powered solutions on Xilinx platforms. It takes input data - from USB/CSI camera, video from file or streams over RTSP, and uses Vitis AI to generate insights from pixels for various usecases. VVAS SDK can be the foundation layer for a number of video analytic solutions like understanding traffic and pedestrians in smart city, health and safety monitoring in hospitals, self-checkout and analytics in retail, detecting component defects at a manufacturing facility and others. VVAS can also be used to build Adaptive Bitrate Transcoding solutions that may require re-encoding the incoming video at different bitrates, resolution and encoding format.

The core SDK consists of several hardware accelerator plugins that use various accelerators such as Video Encoder, Decoder, multiscaler (for resize and color space conversion), Deep learning Processing Unit (DPU) for Machine Learning etc.. By performing all the compute heavy operations in dedicated accelerators, VVAS can achieve highest performance for video analytics, transcoding and several other application areas.

For the developer community, VVAS also provides a framework in the form of generic Infrastructure plugins, software acceleration libraries and a simplified interface to develop their own acceleration library to control a custom hardware accelerator. Using this framework, user can easily integrate their custom accelerators/kernels into Gstreamer framework. VVAS builds on top of Xilinx Run Time (XRT) and Vitis AI and abstracts these complex interface, making it easy for developers to build video analytics and transcoding pipelines without having to learn the complexities of XRT, Vitis AI.

Using VVAS SDK, applications can be deployed on an embedded edge device running on Zynq Ultra-Scale+ MPSoc platform or can be deployed on larger edge or datacenter platforms like Alveo U30.

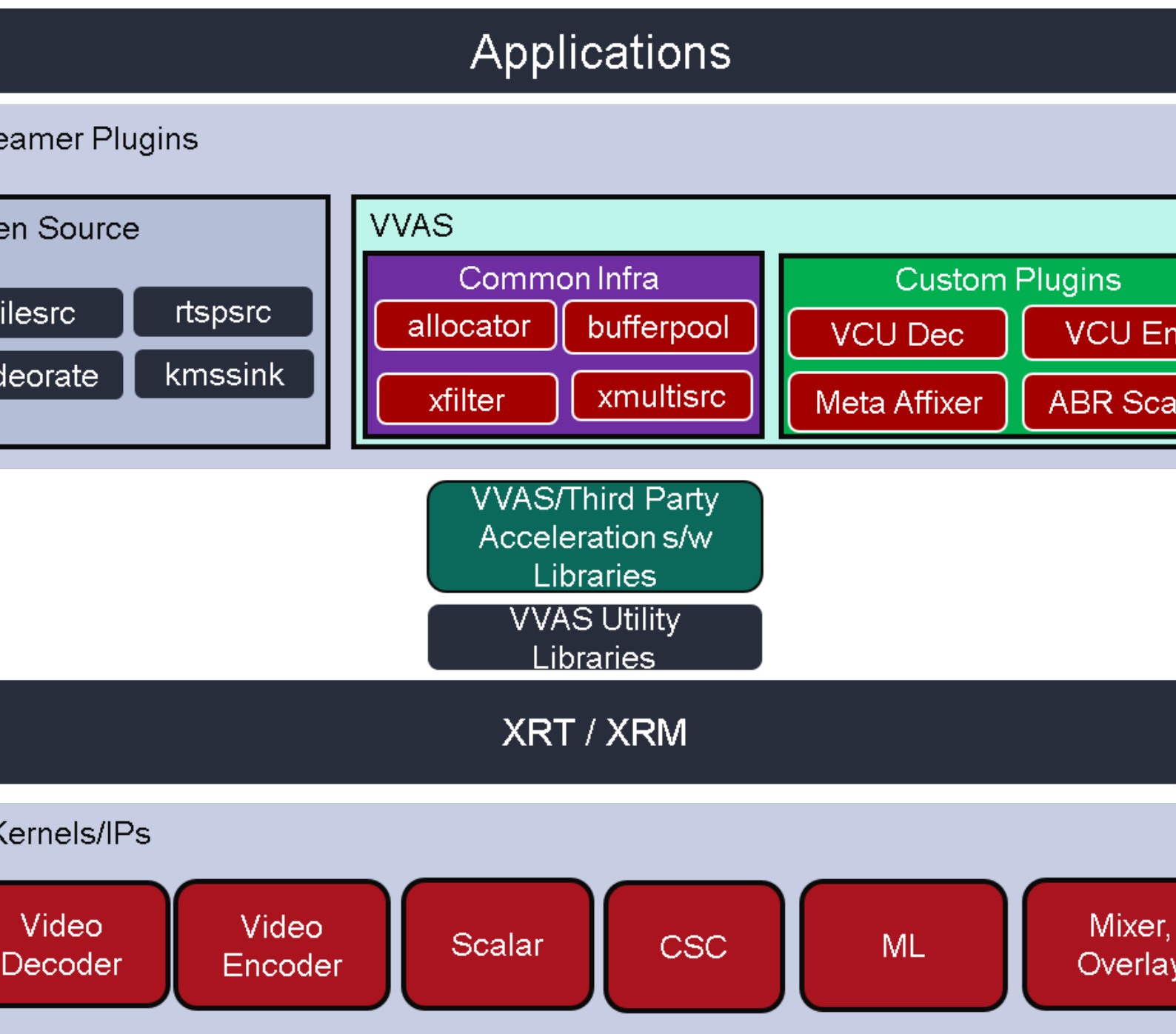


Figure 1. VVAS Block Diagram

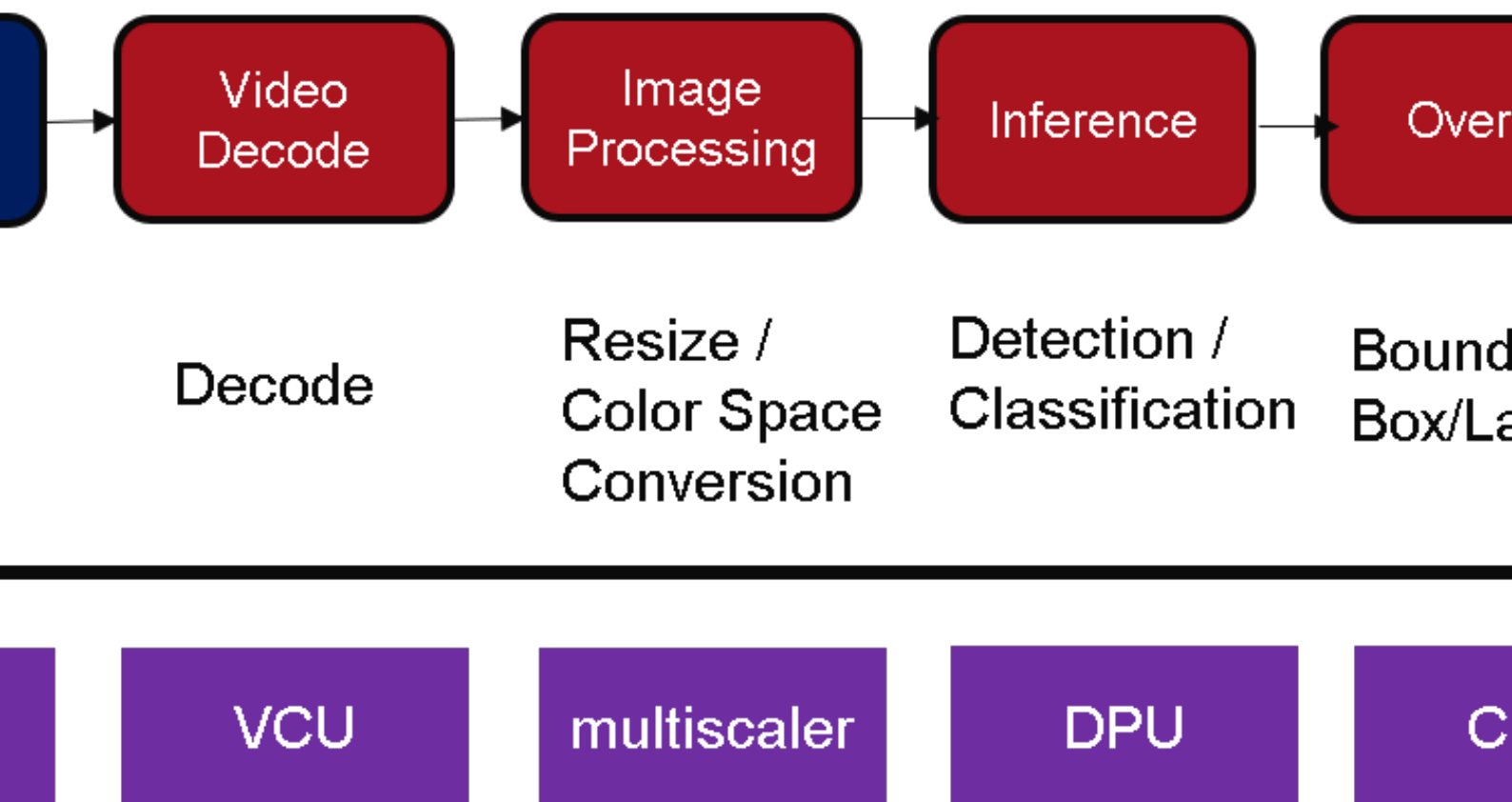


---

## **VVAS Graph Architecture**

---

VVAS is an optimized graph architecture built using the open source GStreamer framework. The graph below shows a typical video analytic application starting from input video to outputting insights. All the individual blocks are various plugins that are used. At the bottom are the different hardware engines that are utilized throughout the application. Optimum memory management with zero-memory copy between plugins and the use of various accelerators ensure the highest performance.



- Streaming data can come over the network through RTSP or from a local file system or from a

camera directly. The captured frames are sent for decoding using the hardware accelerated video decoder. `ivas_xvcudec`, `omxh264dec` and `omxh265dec` are the plugin for decoding.

- After decoding, there is an optional image pre-processing step where the input image can be pre-processed before inference. The pre-processing can be resizing the image or color space conversion. `ivas_xabrscaler` plugin can perform hardware accelerated resize as well as color format conversion on the frame.
- After pre-processing, frame is sent for inference. Inference is performed using `ivas` infrastructure plugin, `ivas_xfilter` and `ivas_xdpuinfer` acceleration library. `ivas_xdpuinfer` is built on top of Vitis AI Development Kit to accelerate the AI Inference on Xilinx hardware platforms.
- To overlay the inference results such as bounding boxes, labels etc., there is a software acceleration library called `ivas_xboundingbox`. This library along with `ivas_xfilter` plug-in draws bounding box and label information on the frame.
- Finally to output the results, VVAS presents various options, like render the output with the bounding boxes on the screen, save the output to the local disk, stream out over RTSP.

## 1.1 Vitis Video Analytics SDK Release V1.0

### 1.1.1 Summary

- Based on Vivado 2021.1, Vitis 2021.1, Petalinux 2021.1
- Based on Vitis AI 1.4 for Machine Learning
- Supports Zynq Ultrascale+ MPSoc based devices like ZCU104 Development Board and KV260 SOM Embedded Platforms

### 1.1.2 Features

- Supports different input sources like Camera, RTP/RTSP streaming, file source etc.
- H264/H265 Video Encoding/Decoding
- Vitis AI based inferencing for detection and classification

- 

#### Supporting 16 ML Models

- `resnet50`
- `resnet18`
- `mobilenet_v2`
- `inception_v1`
- `ssd_adas_pruned_0_95`
- `ssd_traffic_pruned_0_9`
- `ssd_mobilenet_v2`
- `ssd_pedestrian_pruned_0_97`
- `tiny_yolov3_vmss`
- `yolov3_voc_tf`
- `yolov3_adas_pruned_0_9`
- `refinedet_pruned_0_96`
- `yolov2_voc`
- `yolov2_voc_pruned_0_77`

- densebox\_320\_320
  - densebox\_640\_360
- 
- Hardware accelerated Resize and colorspace conversion
  - Region Of Interest based encoding
  - On-screen displaying bounding box around objects and text overly
  - HDMI Tx and Display Port interface for displaying the contents

### 1.1.3 Known Limitations

For known limitations, refer to the example/application design for specific limitations, if any.

### 1.1.4 Known Issues

For known issues, refer to the example/application design for specific issues, if any.

## 1.2 Plug-ins

VVAS is based on the GStreamer framework. The two types of VVAS GStreamer plug-ins are custom plug-ins and infrastructure plug-ins. This section describes the VVAS GStreamer plug-ins, their input, outputs, and control parameters. The plug-ins source code is available in the `ivas-gst-plugins` folder of the VVAS source tree. This section covers the plug-ins that are common for Edge as well as cloud solutions. There are few plug-ins that are specific to Edge/Embedded platforms and are covered in [Plug-ins for Embedded platforms](#). Similarly there are few plug-ins that are specific to Cloud/Data Center platforms and these are covered in [Plug-ins for Data Center Platform](#). The following table lists the VVAS GStreamer plug-ins.

Table 1: GStreamer Plug-ins

Plug-in Name	Functionality
<a href="#">MetaAffixer</a>	Plug-in to scale and attach metadata to the frame of different resolutions.
<a href="#">ivas_xabrscaler</a>	Hardware accelerated scaling and color space conversion.
<a href="#">ivas_xmultisrc</a>	A generic infrastructure plug-in: 1 input, N output, supporting transform processing.
<a href="#">ivas_xfilter</a>	A generic infrastructure plug-in: 1 input, 1 output, supporting pass-through, in-place, and transform processing.

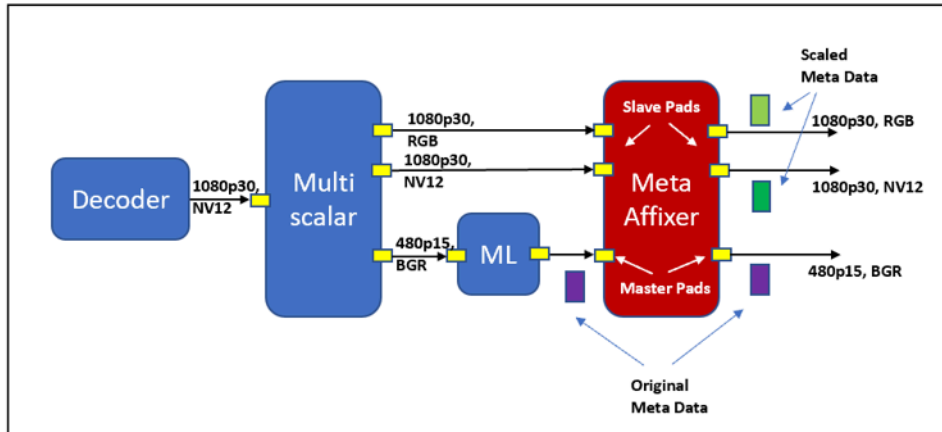
### 1.2.1 Custom Plug-ins

There are specific functions, like video decoder, encoder, and meta affixer where the requirements are difficult to implement in an optimized way using highly simplified and generic infrastructure plug-ins framework. Hence, these functions are implemented using custom GStreamer plug-ins. This section covers details about the custom plug-ins.

#### MetaAffixer

The metaaffixer plug-in, `ivas_xmetaaffixer`, is used to scale the incoming metadata information for the different resolutions. A machine learning (ML) operation can be performed on a different frame resolution and color format than the original frame, but the metadata might be associated with the full resolution, original frame. The `ivas_metaaffixer` has two types of pads, master pads and slave pads. Input pads are pads on request, the number of input pads can be created based on the number of input sources. There is one mandatory master input pad (sink pad) that receives the original/reference metadata. Other input pads are referred to as slave pads. Metadata received on the master sink pad is scaled in relation to the resolution of each of the slave sink pads. The scaled metadata is attached to the buffer going out of the output (source) slave pads. There can be up to 16 slave pads

created as required. For implementation details, refer to [ivas\\_xmetaaffixer source code](#)



### Input and Output

This plug-in is format agnostic and can accept any input format. It operates only on the metadata. The `ivas_xmetaaffixer` plug-in supports the `GstInferenceMeta` data structure. For details about this structure, refer to the VVAS Inference Metadata section.

### Control Parameters and Plug-in Properties

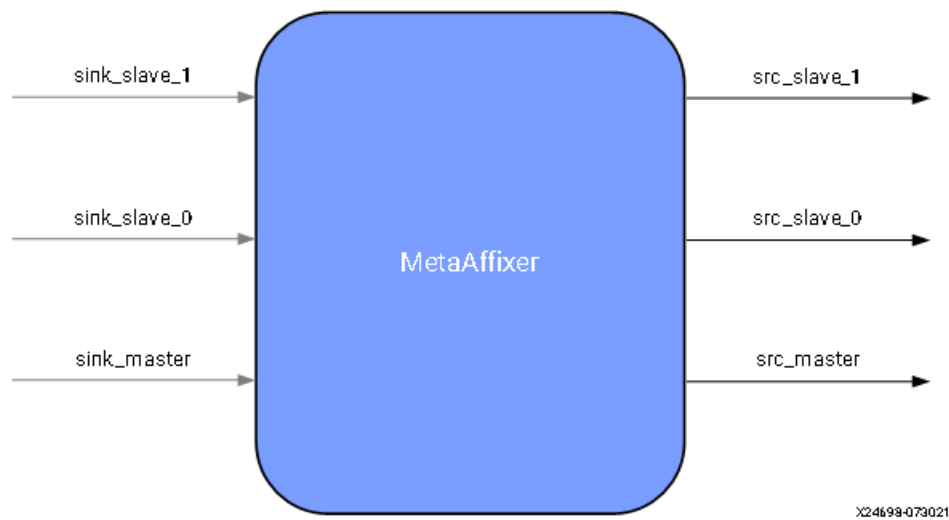
Table 2: `ivas_xmetaaffixer` Plug-in Properties

Property Name	Type	Range	Default	Description
sync	Boolean	True/False	True	This property is to enable the synchronization between master and slave pads buffers. If <code>sync=true</code> is set, then the metadata is scaled and attached to buffers on slave pads that have matching PTS or PTS falls within frame duration of the buffer on the master sink pad. If <code>sync=false</code> is set on the element, then the metadata is scaled and attached to all the buffers on the slave pads. If this option is used, there is possibility that the metadata is not suitable for the frames/buffers that are not corresponding to the frames/buffers on the master pad.

### Pad naming syntax

The pad naming syntax is listed and the following image shows the syntax:

- MetaAffixer master input pad should be named `sink_master`.
- MetaAffixer master output pad should be named `src_master`.
- MetaAffixer slave input pad should be named `sink_slave_0`, `sink_slave_1`.
- MetaAffixer slave output pad should be named `src_slave_0`, `src_slave_1`, `src_slave_2`.



### Example Pipelines

This section covers the example pipelines using the metaaffixer plug-in.

```
gst-launch-1.0 videotestsrc num-buffers=1 \
! video/x-raw, width=1920, height=1080, format=NV12 \
! queue \
! videoconvert \
! queue \
! ima.sink_master ivas_xmetaaffixer name=ima ima.src_master \
! queue \
! fakesink videotestsrc num-buffers=1 \
! video/x-raw, width=1920, height=1080, format=NV12 \
! queue \
! videoconvert \
! video/x-raw, width=1920, height=1080, format=YUY2 \
! ima.sink_slave_0 ima.src_slave_0 \
! queue \
! fakesink -v
```

### ivas\_xabrscaler

In adaptive bit rate (ABR) use cases, one video is encoded at different bit rates so that it can be streamed in different network bandwidth conditions without any artifacts. To achieve this, input frame is decoded, resized to different resolutions and then re-encoded. `ivas_xabrscaler` is a plug-in that takes one input frame and can produce several output frames having different resolutions and color formats. The `ivas_xabrscaler` is a GStreamer plug-in developed to accelerate the resize and color space conversion functionality. For more implementation details, refer to [ivas\\_xabrscaler source code](#).

This plug-in supports:

- Single input, multiple output pads
- Color space conversion
- Resize
- Each output pad has independent resolution and color space conversion capability.

---

**Important** The `ivas_xabrscaler` plug-in controls the multiscaler kernel. If your application uses this plug-in, then make sure that multi-scaler kernel is included in your hardware design.

---

---

**Important** Make sure that the multi-scaler hardware kernel supports maximum resolution required by your application.

---

As a reference, maximum resolution supported by multi-scaler kernel in `Smart Model Select` example design can be found in [multi-scaler kernel config](#)

*Prerequisite*

This plug-in requires the multiscaler kernel to be available in the hardware design. See [Multiscaler Kernel](#)

*Input and Output*

This plug-in accepts buffers with the following color format standards:

- RGBx
- YUY2
- r210
- Y410
- NV16
- NV12
- RGB
- v308
- I422\_10LE
- GRAY8
- NV12\_10LE32
- BGRx
- GRAY10\_LE32
- BGRx
- UYVY
- BGR
- RGBA
- BGRA

---

**Important** Make sure that the color formats needed for your application are supported by the multi-scaler hardware kernel.

---

As a reference, multi-scaler configuration for `smart model select` example design can be found in [multi-scaler configuration](#)

*Control Parameters and Plug-in Properties*

The following table lists the GStreamer plug-in properties supported by the `ivas_xabrscaler` plug-in.

Table 3: `ivas_xabrscaler` Plug-in Properties

Property Name	Type	Range	Default	Description
<code>xclbin-location</code>	string	N/A	<code>./binary _container_1.xclbin</code>	The location of xclbin.
<code>kernel-name</code>	string	N/A	<code>v_multi_scaler: multi_scaler_1</code>	Kernel name and instance separated by a colon.

dev-idx	integer	0 to 31	0	Device index This is valid only in PCIe/Data Center platforms.
ppc	integer	1, 2, 4	2	Pixel per clock supported by a multi-scaler kernel
scale-mode	integer	0, 1, 2	0	Scale algorithm to use: 0:BILINEAR 1:BICUBIC 2:POLYPHASE
coef-load-type	integer	0 => Fixed 1 => Auto	1	Type of filter Coefficients to be used: Fixed or Auto generated
num-taps	integer	6=>6 taps 8=>8 taps 10=>10 taps 12=>12 taps	1	Number of filter taps to be used for scaling

### Example Pipelines

#### One input one output

The following example configures `ivas_xabrscler` in one input and one output mode. The input to the scaler is 1280 x 720, NV12 frames that are resized to 640 x 360 resolution, and the color format is changed from NV12 to BGR.

```
gst-launch-1.0 videotestsrc num-buffers=100 \
! "video/x-raw, width=1280, height=720, format=NV12" \
! ivas_xabrscler xclbin-location="/usr/lib/dpu.xclbin"
kernel-name=v_multi_scaler:v_multi_scaler_1 \
! "video/x-raw, width=640, height=360, format=BGR" ! fakesink -v
```

#### One input multiple output

The following example configures `ivas_xabrscler` for one input and three outputs. The input is 1920 x 1080 resolution in NV12 format. There are three output formats:

- 1280 x 720 in BGR format
- 300 x 300 in RGB format
- 640 x 480 in NV12 format

```
gst-launch-1.0 videotestsrc num-buffers=100 \
! "video/x-raw, width=1920, height=1080, format=NV12, framerate=60/1" \
! ivas_xabrscler xclbin-location="/usr/lib/dpu.xclbin"
kernel-name=v_multi_scaler:v_multi_scaler_1 name=sc sc.src_0 \
! queue \
! "video/x-raw, width=1280, height=720, format=BGR" \
! fakesink sc.src_1 \
! queue \
! "video/x-raw, width=300, height=300, format=RGB" \
! fakesink sc.src_2 \
! queue \
! "video/x-raw, width=640, height=480, format=NV12" \
! fakesink -v
```

## 1.2.2 Infrastructure Plug-ins and Acceleration Software Libraries

Infrastructure plug-ins are generic plug-ins that interact with the acceleration kernel through a set of APIs exposed by an acceleration software library corresponding to that kernel. Infrastructure plug-ins abstract the core/common functionality of the GStreamer framework (for example: caps negotiation and buffer management).



Table 5: Acceleration Software Libraries

Infrastructure Plug-ins	Function
ivas_xfilter	Plug-in has one input, one output Support Transform, passthrough and inplace transform operations
ivas_xmultisrc	Plug-in support one input and multiple output pads. Support transform operation

**Note** Though one input and one output kernel can be integrated using any of the two infrastructure plug-ins, we recommend to use ivas\_xfilter plugin for one input and one output kernels.

Acceleration software libraries control the acceleration kernel, like register programming, or any other core logic required to implement the functions. Acceleration software libraries expose a simplified interface that is called by the GStreamer infrastructure plug-ins to interact with the acceleration kernel. The following table lists the acceleration software libraries developed to implement specific functionality. These libraries are used with one of the infrastructure plug-ins to use the functionality a GStreamer-based application. Example pipelines with GStreamer infrastructure plug-ins and acceleration software libraries are covered later in this section.

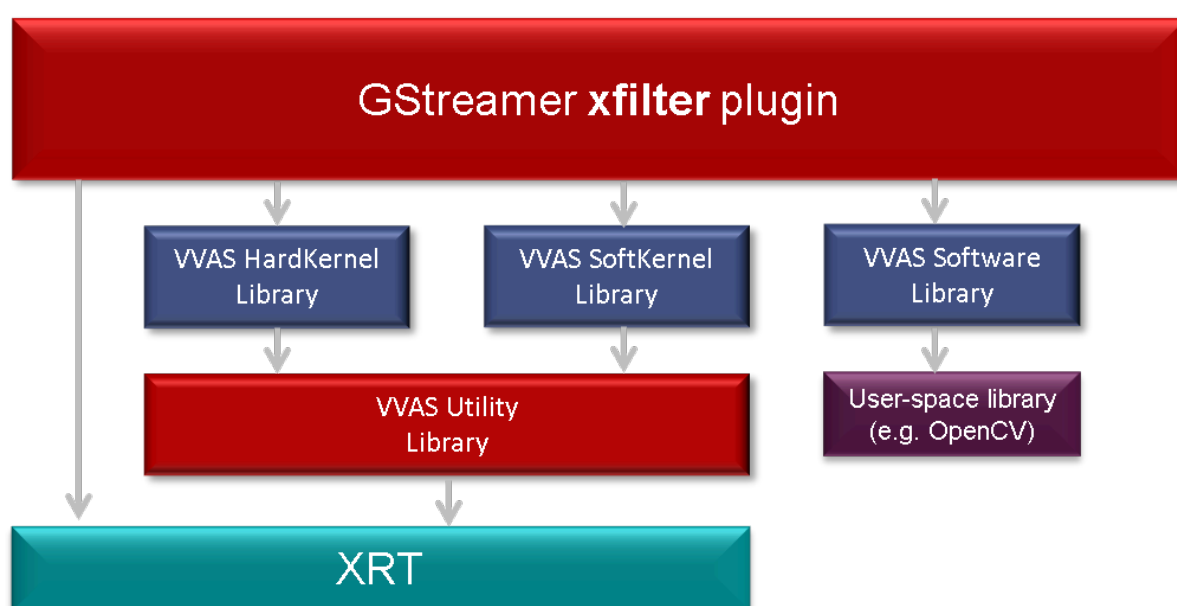
Table 6: Acceleration Software Libraries

Acceleration SoftwareLibrary	Function
ivas_xdpuinfer	Library based on Vitis AI to control DPU kernels for machine learning.
ivas_xboundingbox	Library to draw a bounding box and labels on the frame using OpenCV.

The GStreamer infrastructure plug-ins are available in the ivas-gst-plugins repository/ folder. The following section describes each infrastructure plug-in.

#### ivas\_xfilter

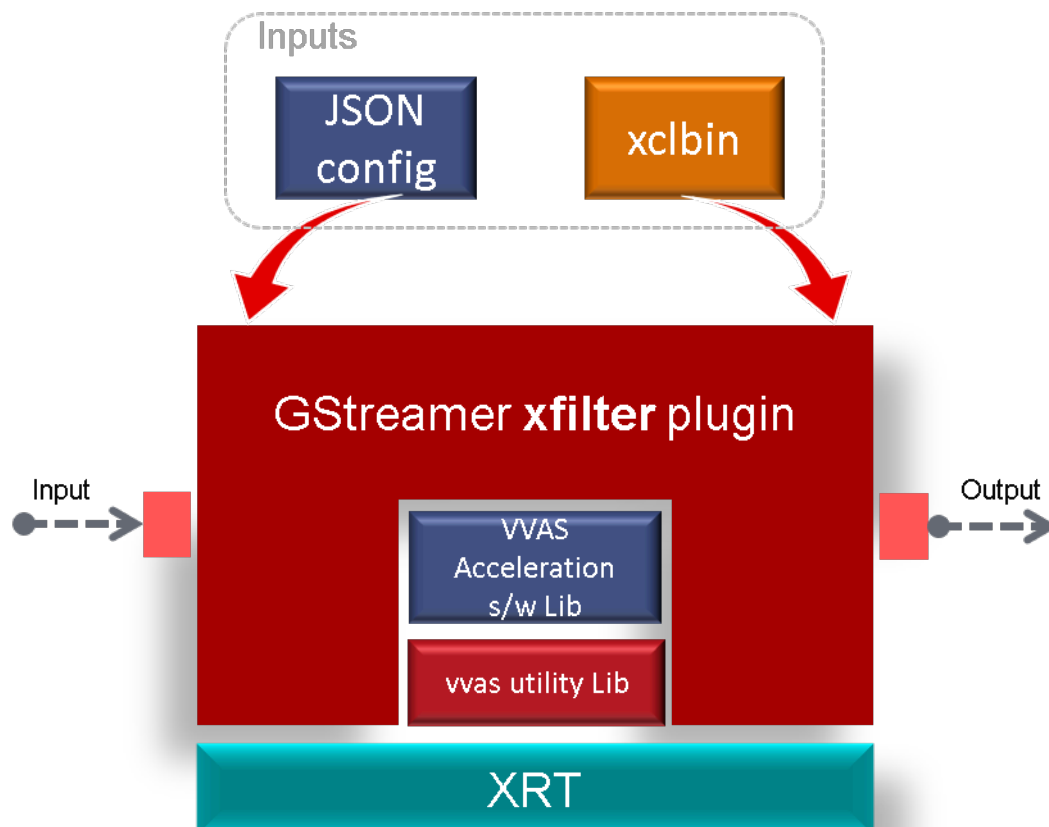
The GStreamer ivas\_xfilter is an infrastructure plug-in that is derived from GstBaseTransform. It supports one input pad and one output pad. The ivas\_xfilter efficiently works with hard-kernel/-soft-kernel/software (user-space) acceleration software library types as shown in the following figure.



This plug-in can operate in the following three modes.

- **Passthrough:** Useful when the acceleration software library does not need to alter the input buffer.
- **In-place:** Useful when the acceleration software library needs to alter the input buffer.
- **Transform:** In this mode, for each input buffer, a new output buffer is produced.

You must set the mode using the JSON file. Refer to JSON File Schema for information related to the kernels-config property.



The `ivas_xfilter` plug-in takes configuration file as one of the input properties, `kernels- config`. This configuration file is in JSON format and contains information required by the kernel. During initialization, the `ivas_xfilter` parses the JSON file and performs the following tasks:

- Finds the VVAS acceleration software library in the path and loads the shared library.
- Understands the acceleration software library type and prepares the acceleration software library handle (IVASKernel) to be passed to the core APIs.

#### *Input and Output*

The `ivas_xfilter` accepts the buffers with the following color formats on input GstPad and output GstPad.

- GRAY8
- NV12
- BGR

- RGB
- YUY2
- r210
- v308
- GRAY10\_LE32
- ABGR
- ARGB

The formats listed are the Xilinx IP supported color formats. To add other color formats, update the `ivas_kernel.h` and `ivas_xfilter` plug-ins.

#### *Control Parameters and Plug-in Properties*

The following table lists the GObject properties exposed by the `ivas_xfilter`. Most of them are only available in PCIe supported platforms.

Table 6: GObject Properties

Property Name	Platforms Supported	Type	Range	Default	Description
dynamic-config	PCIe and Embedded	String	N/A	Null	JSON formatted string contains kernel specific configuration for run time changes
dev-idx	PCIe only	Integer	0 to 31	0	Device used for kernel processing, xclbin download.
kernels-config	PCIe and Embedded	String	N/A	NULL	JSON configuration file path based on VVAS acceleration software library requirements. Refer to the B-JSON-File-Schema
sk-cur-idx	PCIe only	Integer	0 to 31	0	Softkernel current index to be used for executing job on device.
reservation-id	PCIe only	Integer	0 to 1024	0	Reservation ID provided by the Xilinx resource manager (XRM).

#### *JSON Format for ivas\_xfilter Plug-in*

The following table provides the JSON keys accepted by the GStreamer `ivas_xfilter` plug-in.

Table 7: Root JSON Object Members

JSON Key	Item	Item Description
xclbin-location	Description	The location of the xclbin that is used to program the FPGA device.
	Value type	String

	Mandatory or optional	Conditionally mandatory: <ul style="list-style-type: none"> <li>If the VVAS acceleration software library is developed for hard-kernel IP and soft-kernel, then the xclbin-location is mandatory.</li> <li>If the VVAS acceleration software library is developed for a software library (e.g., OpenCV), then the xclbin-location is not required</li> </ul>
	Default value	NULL
ivas-library-repo	Description	This is the VVAS libraries repository path to look for acceleration software libraries by the VVAS GStreamer plug-in.
	Value type	String
	Mandatory or optional	Optional
	Default value	/usr/lib
element-mode	Description	GStreamer element mode to operate. Based on your requirement, choose one of the following modes: <ol style="list-style-type: none"> <li>1. Passthrough: In this mode, element does not want to alter the input buffer.</li> <li>2. Inplace: In this mode, element wants to alter the input buffer itself instead of producing new output buffers.</li> <li>3. Transform: In this mode, element produces a output buffer for each input buffer.</li> </ol>
	Value type	Enum
	Mandatory or optional	Mandatory
kernels	Description	This is the array of kernel objects. Each kernel object provides information about an VVAS acceleration software library configuration. The ivas_xfilter only takes the first kernel object in the kernel array.
	Value type	Array of objects
	Mandatory or optional	Mandatory
	Default value	None
	Minimum value	1
	Maximum value	10
	Object members	Refer to <a href="#">Kernel JSON Object</a>

The information in the following table is specific to the kernel chosen.

Table 8: Kernel JSON Object Members

JSON Key	Item	Item Description
library-name	Description	The name of the VVAS acceleration software library to be loaded by the VVAS GStreamer plug-ins. The absolute path of the kernel library is formed by the pre-pending ivas-library-repo path.

	Value type	String
	Mandatory or optional	Mandatory
	Default value	None
kernel-name	Description	The name of the IP/kernel in the form of <kernel name>:<instance name>
	Value type	String
	Mandatory or optional	Optional
	Default value	None
config	Description	Holds the configuration specific to the VVAS acceleration software library. VVAS GStreamer plug-ins do not parse this JSON object, instead it is sent to the acceleration software library.
	Value type	Object
	Mandatory or optional	Optional
	Default value	None
softkernel	Description	Contains the information specific to the soft-kernel. This JSON object is valid only for the PCIe based platforms.
	Value type	Object
	Mandatory or optional	Mandatory if kernel library is written for soft-kernel.
	Default value	None
	Members	Not required for embedded platforms.

For `ivas_xfilter` implementation details, refer to [ivas\\_xfilter source code](#)

*Example JSON Configuration Files*

*JSON File for Vitis AI API-based VVAS Acceleration Software Library*

The following JSON file is for pure software-based acceleration, it does not involve any kernel for acceleration. However, the Vitis AI API based DPU is a special case, where the DPU hardware kernel is controlled by Vitis AI. The acceleration software library calls the Vitis AI API calls and it is implemented as a pure software acceleration software library. There is no need to provide the path of the xclbin in the JSON file.

```
{
  "ivas-library-repo": "/usr/lib/",
  "element-mode": "inplace",
  "kernels" : [
    {
      "library-name": "libivas_xdpuinfer.so",
      "config": {
        "model-name" : "densebox_320_320",
        "model-class" : "FACEDETECT",
        "model-format": "BGR",
        "model-path" : "/usr/share/vitis_ai_library/models/",
        "run_time_model" : false,
        "need_preprocess" : true,
        "performance_test" : true,
        "max_num" : -1,
        "prob_cutoff" : 0.0,
        "debug_level" : 1
      }
    }
  ]
}
```

### *JSON File for a Hard Kernel*

The following JSON file uses `ivas_xfilter` to control multi-scaler IP (hard-kernel). The acceleration software library accessing the register map is `libivas_xcrop.so`.

```
{
  "xclbin-location":"/usr/lib/dpu.xclbin",
  "ivas-library-repo": "/usr/lib/",
  "element-mode":"passthrough",
  "kernels" :[
    {
      "kernel-name":"v_multi_scaler:v_multi_scaler_1",
      "library-name":"libivas_xcrop.so",
      "config": {
      }
    }
  ]
}
```

### *Example Pipelines*

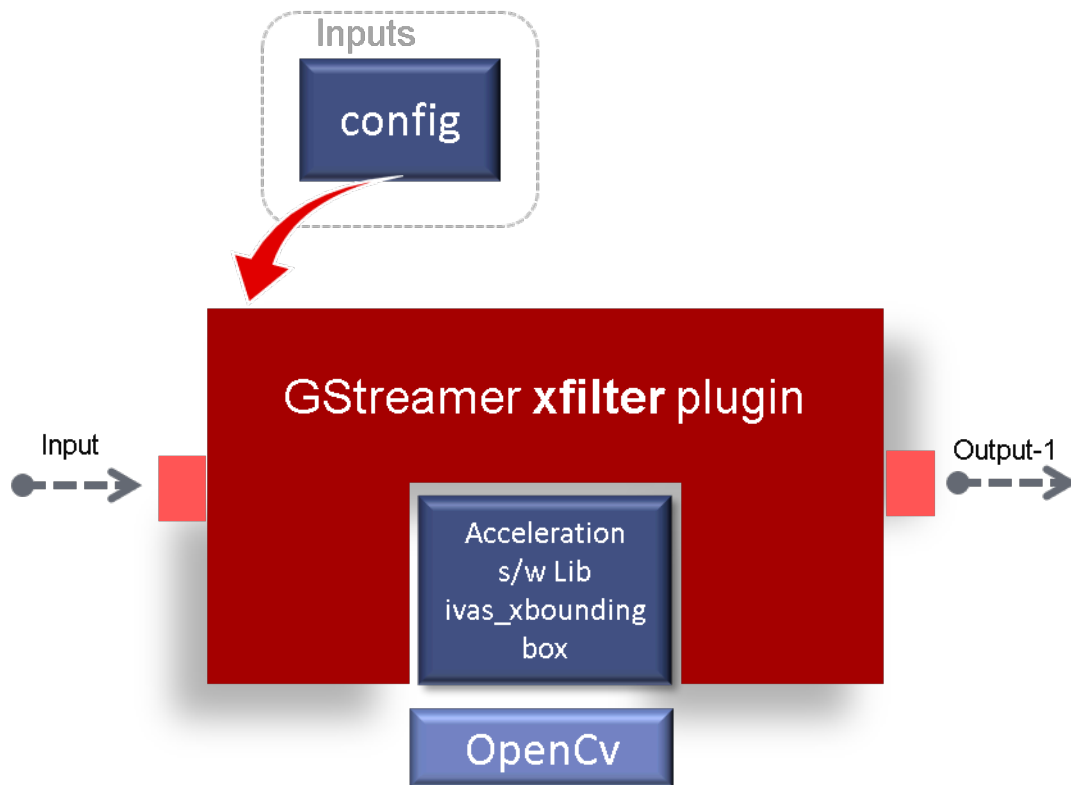
This section covers the GStreamer example pipelines using the `ivas_xfilter` infrastructure plug-in and several acceleration software libraries. This section covers the bounding box functionality and the machine learning functions using the `ivas_xfilter` plug-in.

- The bounding box functionality is implemented in the `ivas_xboundingbox` acceleration software library that is controlled by the `ivas_xfilter` plug-in.
- Machine learning using the DPU is implemented by the `ivas_xdpuinfer` acceleration software library that is called by the `ivas_xfilter` plug-in.

### **Bounding Box Example**

This section describes how to draw a bounding box and label information using the VVAS infrastructure plug-in `ivas_xfilter` and the `ivas_xboundingbox` acceleration software library. The `ivas_xboundingbox` interprets machine learning inference results from the `ivas_xdpuinfer` acceleration software library and uses an OpenCV library to draw the bounding box and label on the identified objects.

For `ivas_xboundingbox` implementation details, refer to [ivas\\_xboundingbox source code](#)



### Prerequisites

There are a few requirements before start running bounding box examples. Make sure these prerequisites are met.

- Installation of OpenCV: The `ivas_xboundingbox` uses OpenCV for the graphics back-end library to draw the boxes and labels. Install the `libopencv-core-dev` package (the preferred version is 3.2.0 or later).

### JSON File for `ivas_xboundingbox`

This section describes the JSON file format and configuration parameters for the bounding box acceleration software library. The GStreamer `ivas_xfilter` plug-in used in the inplace mode. Bounding box and labels are drawn on identified objects on the incoming frame. Bounding box functionality is implemented in the `libivas_xboundingbox.so` acceleration software library.

The following example is of a JSON file to pass to the `ivas_xfilter`.

```
{
  "xclbin-location": "/usr/lib/dpu.xclbin",
  "ivas-library-repo": "/usr/local/lib/ivas",
  "element-mode": "inplace",
  "kernels" : [
    {
      "library-name": "libivas_xboundingbox.so",
      "config": {
        "font_size" : 0.5,
        "font" : 3,
        "thickness" : 2,
        "debug_level" : 2,
        "label_color" : { "blue" : 0, "green" : 0, "red" : 0 },
        "label_filter" : [ "class", "probability" ],
        "classes" : [
```

```

        {
            "name" : "car",
            "blue" : 255,
            "green" : 0,
            "red" : 0
        },
        {
            "name" : "person",
            "blue" : 0,
            "green" : 255,
            "red" : 0
        },
        {
            "name" : "bicycle",
            "blue" : 0,
            "green" : 0,
            "red" : 255
        }
    ]
}

```

Various configuration parameters of the bounding box acceleration software library are described in the following table.

Table 9: ivas\_xboundingbox Parameters

Parameter	Expected Values	Description
debug_level	0: LOG_LEVEL_ERROR 1: LOG_LEVEL_WARNING 2: LOG_LEVEL_INFO 3: LOG_LEVEL_DEBUG	Enables the log levels. There are four log levels listed in the expected values column.
font	0 to 7	Font for the label text. 0: Hershey Simplex 1: Hershey Plain 2: Hershey Duplex 3: Hershey Complex 4: Hershey Triplex 5: Hershey Complex Small 6: Hershey Script Simplex 7: Hershey Script Complex
font_size	0.5 to 1	Font fraction scale factor that is multiplied by the font-specific base size.
thickness	Integer 1 to 3	The thickness of the line that makes up the rectangle. Negative values, like -1, signify that the function draws a filled rectangle. The recommended value is between 1 and 3.
label_color	{ "blue" : 0, "green" : 0, "red" : 0 }	The color of the label is specified.
label_filter	[ "class", "probability" ]	This field indicates that all information printed is the label string. Using "class" alone adds the ML classification name. For example car, person, etc. The addition of "probability" in the array adds the probability of a positive object identification.

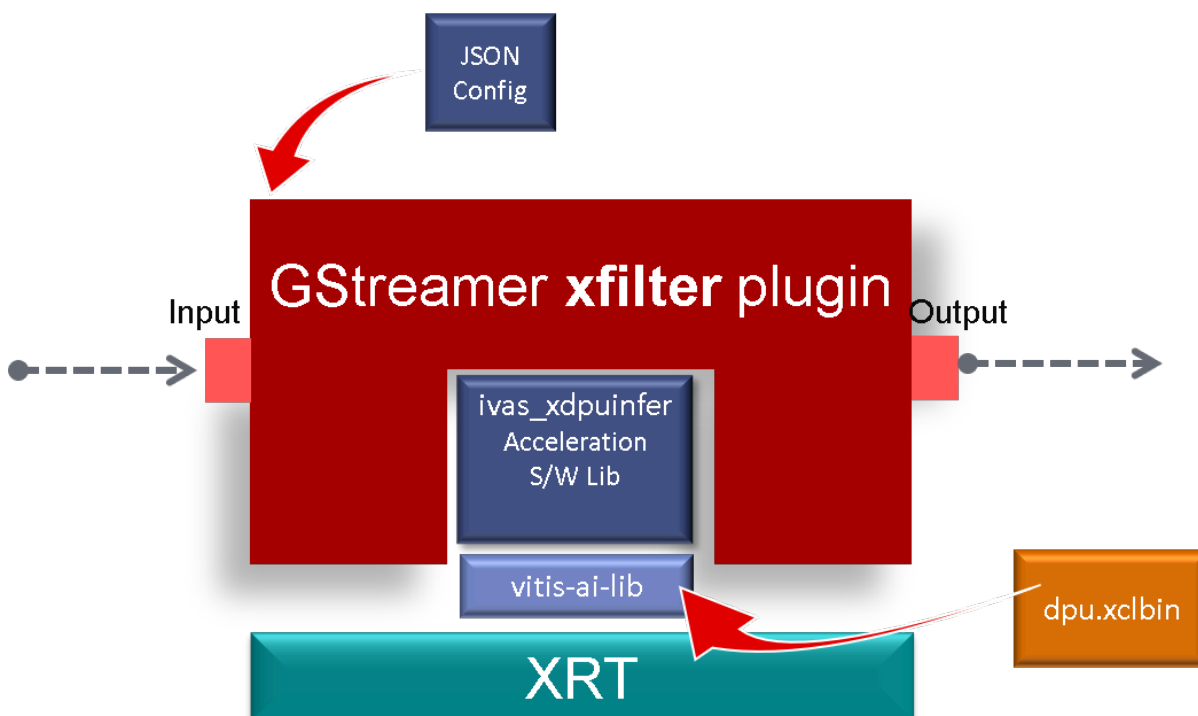


classes	<pre>{ "name" : "car",   "blue" : 255, "green" :   0, "red" : 0 }</pre>	<p>This is a filtering option when using the <code>ivas_xboundingbox</code>. The bounding box is only drawn for the classes that are listed in this configuration. Other classes are ignored. For instance, if "car", "person", "bicycle" are entered under "classes", then the bounding box is only drawn for these three classes, and other classes like horse, motorbike, etc. are ignored.</p> <p>The expected value columns shows an example of how each class should be described. All objects in this example, by class, are drawn using the color combination listed.</p> <p>The class names in this list match the class names assigned by the <code>ivas_xdpuinfer</code>. Otherwise, the bounding box is not drawn.</p> <p>For face detect, keep the "classes" array empty.</p>
---------	---	--

An example of using a bounding box along with the machine learnin plug-in is shown in the [Multi Channel ML Tutorial](#).

### Machine Learning Example

This section discusses how machine learning solutions can be implemented using the VVAS infrastructure `ivas_xfilter` plug-in and the `ivas_xdpuinfer` acceleration software library.



The `ivas_xdpuinfer` is the acceleration software library that controls the DPU through the Vitis AI interface. The `ivas_xdpuinfer` does not modify the contents of the input buffer. The input buffer is

passed to the Vitis AI model library that generates the inference data. This inference data is then mapped into the VVAS metameta structure and attached to the input buffer. The same input buffer is then pushed to the downstream plug-in.

For `ivas_xdpuinfer` implementation details, refer to [ivas\\_xdpuinfer source code](#)

#### *Prerequisites*

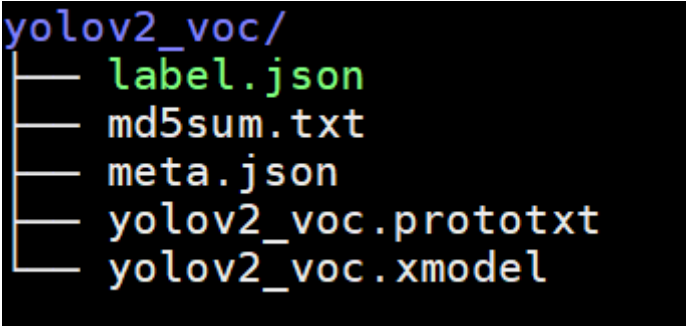
There are a few requirements to be met before you can start running the machine learning examples.

#### *Model Information*

The model directory name must match with the ELF and prototxt files inside the model directory. The model directory should contain:

- `model-name.elf/model-name.xmodel` and `model-name.prototxt` file.
- `label.json` file containing the label information, if the models generate labels.

The following is an example of the model directory (`yolov2_voc`), which contains the `yolov2_voc.x-model` and `yolov2_voc.prototxt` files along with the `label.json` file.



```
yolov2_voc/  
├── label.json  
├── md5sum.txt  
├── meta.json  
├── yolov2_voc.prototxt  
└── yolov2_voc.xmodel
```

#### *xclbin Location*

By default, the Vitis AI interface expects `xclbin` to be located at `/usr/lib/` and the `xclbin` is called `dpu.xclbin`. Another option is to use the environment variable `XLNX_VART_FIRMWARE` to change the path and the corresponding path can be updated in the JSON file. That is, export `XLNX_VART_FIRMWARE=/where/your/dpu.xclbin`.

#### *Input Image*

The `ivas_xdpuinfer` works with raw BGR and RGB images as required by the model. Make sure you have specified correct color format in `model-format` field in json file. The exact resolution of the image to `ivas_xdpuinfer` must be provided, it is expected by the model. There is a performance loss if a different resolution of the BGR image is provided to the `ivas_xdpuinfer`, because resizing is done on the CPU inside the Vitis AI library.

#### *JSON File for ivas\_xdpuinfer*

The following table shows the JSON file format and configuration parameters for `ivas_xdpuinfer`.

Table 10: JSON File for `ivas_xdpuinfer`

Parameter	Type	Expected Values	Default	Description
xclbin-location	string	/usr/lib/dpu.xclbin	NULL	By default, Vitis AI expects xclbin to be located at /usr/lib/ and xclbin is called dpu.xclbin. The environment variable XLNX_VART_FIRMWARE could also be used to change the path and the corresponding path can be updated in the JSON file. e.g., export XLNX_VART_FIRMWARE=/where/your/ dpu.xclbin
ivas-library-repo	string	/usr/local/lib/ivas/	/usr/lib/	This is the path where the ivas_xfilter will search the acceleration software library. The kernel name is specified in the library-name parameter of the JSON file.
element-mode	string	inplace	None	Because the input buffer is not modified by the ML operation, but the metadata generated out of an inference operation needs to be added/appended to the input buffer, the GstBuffer is writable. The ivas_xfilter is configured in inplace mode
kernels	N/A	N/A	N/A	The JSON tag for starting the kernel specific configurations.
kernel-name	string	N/A	NULL	The name and instance of a kernel separated by ":"
library-name	string	N/A	NULL	Acceleration software library name for the kernel. It is appended to the ivas-library-repo for an absolute path.
config	N/A	N/A	N/A	The JSON tag for kernel-specific configurations depending on the acceleration software library.
model-name	string	resnet50	N/A	Name string of the machine learning model to be executed. The name string should be same as the name of the directory available in model -path parameter file. If the name of the model ELF file is resnet50.elf, then the model-name is resnet50 in the JSON file. The ELF file present in the specified path model-path of the JSON file.
model-class	string	YOLOV3 FACEDETECT CLASSIFICATION SSD REFINEDET TFSSD YOLOV2	N/A	Class of model corresponding to model. Some of the examples are shown here: <ul style="list-style-type: none"> <li><b>YOLOV3:</b> yolov3_adas_pruned_0_9, yolov3_voc, yolov3_voc_tf</li> <li><b>FACEDETECT:</b> densebox_320_320, densebox_640_360</li> <li><b>CLASSIFICATION:</b> resnet18, resnet50, resnet_v1_50_tf</li> </ul>

model-format	string	RGB/BGR	N/A	Image color format required by model.
model-path	string	/usr/share/ vitis_ ai_library/ models/	N/A	Path of the folder where the model to be executed is stored.
run_time_model	Boolean	True/False	False	If there is a requirement to change the ML model at the frame level, then set this flag to true. If this parameter is set to true then ivas_xdpuinfer will read the model name and class from the incoming input metadata and execute the same model found in the path specified in the model-path. The model-name and model-class parameter of the JSON file are not required when enabling this parameter.
need_preprocess	Boolean		True	If need_preprocess = true: Normalize with mean/scale through the Vitis AI Library If need_preprocess = false: Normalize with mean/scale is performed before feeding the frame to ivas_xdpuinfer. The Vitis AI library does not perform these operations.
performance_test	Boolean	True/False	False	Enable performance test and corresponding flops per second (f/s) display logs. Calculates and displays the f/s of the standalone DPU after every second.
debug_level	integer	0 to 3	1	Used to enable log levels. There are basically four log levels for a message sent by the kernel library code, starting from level 0 and decreasing in severity till level 3 the lowest log-level identifier. When a log level is set, it acts as a filter, where only messages with a log-level lower than it, (therefore messages with an higher severity) are displayed. 0: This is the highest level in order of severity: It is used for messages about critical errors, both hardware or software related. 1: This level is used in situations where your attention is immediately required. 2: This is the log level used for informational messages about the action performed by the kernel and output of model 3: This level is used for debugging. level is

#### Model Parameters

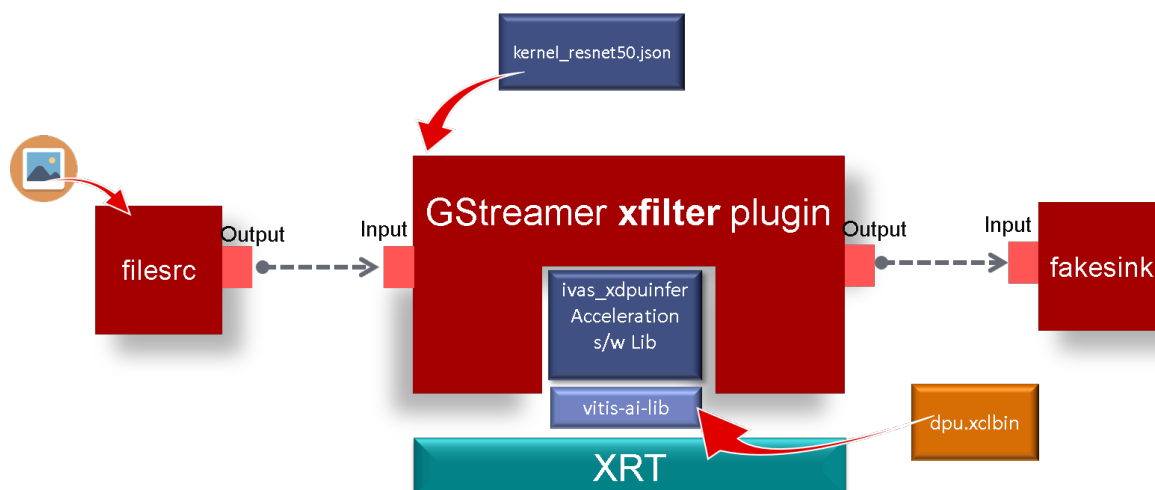
The Vitis AI library provides a way to read model parameters by reading the configuration file. It facilitates uniform configuration management of model parameters. The configuration file is located in the model-path of the JSON file with [model\_name].prototxt. These parameters are model specific. For more information on model parameters, refer to the Vitis AI Library User Guide ([UG1354](#)).

### Example GStreamer Pipelines

This section describes a few example GStreamer pipelines.

#### Classification Example Using Resnet50

The following pipeline performs ML using a Resnet50 model. DPU configuration uses kernels- config = ./json\_files/kernel\_resnet50.json for the ivas\_xdpuinfer. The output of the ivas\_xfilter is passed to fakesink along with the metadata.



The GStreamer command for the example pipeline:

```
gst-launch-1.0 filesrc location="<PATH>/001.bgr" blocksize=150528 numbuffers=1
! videoparse width=224 height=224 framerate=30/1 format=16
! ivas_xfilter name="kernel1" kernels-config="<PATH>/kernel_resnet50.json"
! fakesink
```

The JSON file for the ivas\_xdpuinfer to execute resnet50 model based classification pipeline is described below.

```
{
  "ivas-library-repo": "/usr/local/lib/ivas/",
  "element-mode": "inplace",
  "kernels" : [
    {
      "library-name": "libivas_xdpuinfer.so",
      "config": {
        "model-name" : "resnet50",
        "model-class" : "CLASSIFICATION",
        "model-format" : "BGR",
        "model-path" : "/usr/share/vitis_ai_library/models/",
        "run_time_model" : false,
        "need_preprocess" : true,
        "performance_test" : true,
        "debug_level" : 2
      }
    }
  ]
}
```

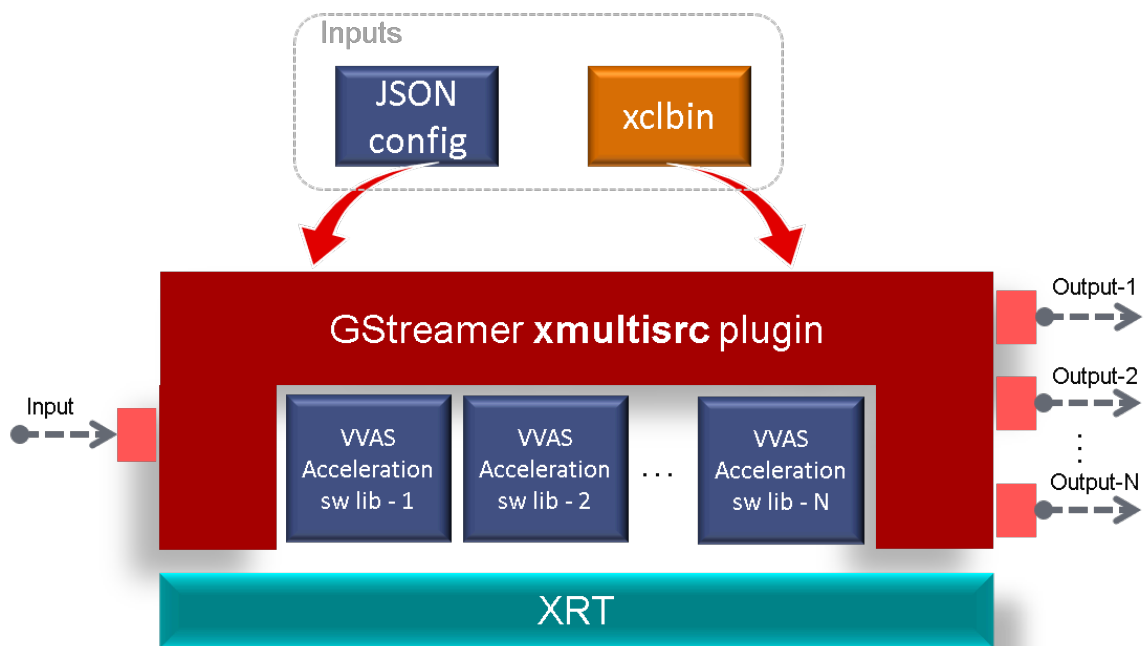
**Note** If “need\_preprocess” = false, then pre-processing operations like, Normalization, scaling must be performed on the frame before feeding to ivas\_xfilter/ivas\_xdpuinfer otherwise results may not be

as expected.

### ivas\_xmultisrc

The GStreamer `ivas_xmultisrc` plug-in can have one input pad and multiple-output pads. This single plug-in supports multiple acceleration kernels, each controlled by a separate acceleration software library.

For `ivas_xmultisrc` implementation details, refer to [ivas\\_xmultisrc source code](#)



### Input and Output

Input and output accepts buffers with the following color formats on input GstPad and output GstPad.

- GRAY8
- NV12
- BGR
- RGB
- YUY2
- r210
- v308
- GRAY10\_LE32
- ABGR
- ARGB

The formats listed are the Xilinx IP supported color formats. To add other color formats, update the `ivas_kernel.h` and `ivas_xfilter` plug-ins.

### Control Parameters and Plug-in Properties

Table 11: Plug-in Properties

Property Name	Type	Range	Default	Description
Kconfig	String	N/A	NULL	Path of the JSON configuration file based on the VVAS acceleration software library requirements. For further information, refer to B-JSON-File-Schema

#### JSON File for ivas\_xmultisrc

This section covers the format of JSON file/string to be passed to the ivas\_xmultisrc plug-in.

#### Example JSON File

The following example file describes how to specify two kernels that are being controlled by the single instance of the ivas\_xmultisrc plug-in. Modify this json file as per your kernels and acceleration software library. The next section describes each field in this example file.

```
{
  "xclbin-location":"/usr/lib/binary_1.xclbin",
  "ivas-library-repo": "/usr/lib/",
  "kernels" :[
    {
      "kernel-name":"resize:resize_1", <----- kernel 1
      "library-name":"libivas_xresize.so",
      "config": {
        x : 4,
        y : 7
      }
    }
    {
      "kernel-name":"cvt_rgb:cvt_rgb_1", <----- kernel 2
      "library-name":"libcvt_bgr.so",
      "config": {
        name = "xilinx",
        value = 98.34
      }
    }
  ]
}
```

Table 12: JSON Properties

Property Name	Type	Range	Default	Description
xclbin-location	String	N/A	NULL	The path of xclbin including the xclbin name. The plug-in downloads this xclbin and creates an XRT handle for memory allocation and programming kernels.
ivas-library-repo	String	N/A	/usr/lib	The library path for the VVAS repository for all the acceleration software libraries.
kernels	N/A	N/A	N/A	The JSON tag for starting the kernel specific configurations.
kernel-name	String	N/A	NULL	Name and instance of a kernel separated by ":" as mentioned in xclbin.
library-name	String	N/A	NULL	The acceleration software library name for the kernel. This is appended to ivas-library-repo for an absolute path.
config	N/A	N/A	N/A	The JSON tag for kernel specific configurations that depends on the acceleration software library.

#### Src Pad Naming Syntax

For single output pad naming is optional. For multiple pads, the source pads names shall be as mentioned below, assuming the name of the plug-in as sc.

```
sc.src_0, sc.src_1 .....sc.src_n
```

### Example Pipelines

#### Single Output Pad

The following example demonstrates the `ivas_xmultisrc` plug-in configured for one input and one output. A test video pattern is generated by the `videotestsrc` plug-in and passed to the `ivas_xmultisrc` plug-in. Depending on the kernel being used, `ivas_xmultisrc` uses `kernel.json` to configure the kernel for processing the input frame before passing it to the `fakesink`.

```
gst-launch-1.0 videotestsrc
! "video/x-raw, width=1280, height=720, format=BGR"
! ivas_xmultisrc kconfig="/root/jsons/<kernel.json>"
! "video/x-raw, width=640, height=360, format=BGR"
! fakesink -v
```

The following is an example `kernel.json` file having `mean_value` and `use_mean` as kernel configuration parameters. Modify this as per your kernel requirements.

```
{
  "xclbin-location": "/usr/lib/dpu.xclbin",
  "ivas-library-repo": "/usr/lib/ivas",
  "kernels": [
    {
      "kernel-name": "<kernel-name>",
      "library-name": "libivas_xresize_bgr.so",
      "config": {
        "use_mean": 1,
        "mean_value": 128
      }
    }
  ]
}
```

#### GstIvasBufferPool

The GStreamer VVAS buffer pool holds the pool of video frames allocated using the GStreamer allocator object. It is derived from the GStreamer base video buffer pool object (`GstVideoBufferPool`).

The VVAS buffer pool:

- Allocates buffers with stride and height alignment requirements. (e.g., the video codec unit (VCU) requires the stride to be aligned with 256 bytes and the height aligned with 64 bytes)
- Provides a callback to the GStreamer plug-in when the buffer comes back to the pool after it is used.

The following API is used to create the buffer pool.

```
GstBufferPool *gst_ivas_buffer_pool_new (guint stride_align, guint
elevation_align)
```

#### Parameters:

```
stride_align - stride alignment required
elevation_align - height alignment required
```

#### Return:

```
GstBufferPool object handle
```

Plug-ins can use the following API to set the callback function on the IVAS buffer pool and the callback function is called when the buffer arrives back to the pool after it is used.

```
Buffer release callback function pointer declaration:
typedef void (*ReleaseBufferCallback) (GstBuffer *buf, gpointer user_data);
```



```
void gst_ivas_buffer_pool_set_release_buffer_cb (GstIvasBufferPool *xpool,
ReleaseBufferCallback release_buf_cb, gpointer user_data)
```

Parameters:

xpool - IVAS buffer pool created using `gst_ivas_buffer_pool_new`  
 release\_buf\_cb - function pointer of callback  
 user\_data - user provided handle to be sent **as** function argument **while** calling `release_buf_cb()`

Return:

**None**

### *GstIvasAllocator*

The GStreamer IVAS allocator object is derived from an open source `GstAllocator` object designed for allocating memory using XRT APIs. The IVAS allocator is the backbone to the VVAS framework achieving zero-copy (wherever possible).

### *Allocator APIs*

GStreamer plug-in developers can use the following APIs to interact with the IVAS allocator. To allocate memory using XRT, the GStreamer plug-ins and buffer pools require the `GstAllocator` object provided by the following API.

```
GstAllocator* gst_ivas_allocator_new (guint dev_idx, gboolean need_dma)
```

Parameters:

dev\_idx - FPGA Device index on which memory **is** going to allocated  
 need\_dma - will decide memory allocated **is** dmabuf **or not**

Return:

`GstAllocator` handle on success **or** NULL on failure

..note:: In PCIe platforms, the DMA buffer allocation support is not available. This means that the `need_dma` argument to `gst_ivas_allocator_new()` API must be false.

Use the following API to check if a particular `GstMemory` object is allocated using `GstIvasAllocator`.

```
gboolean gst_is_ivas_memory (GstMemory *mem)
```

Parameters:

mem - memory to be validated

Return:

true **if** memory **is** allocated using IVAS Allocator **or** false

When GStreamer plug-ins are interacting with hard-kernel IP or soft-kernel, the plug-ins need physical memory addresses on an FPGA using the following API.

```
guint64 gst_ivas_allocator_get_paddr (GstMemory *mem)
```

Parameters:

mem - memory to get physical address

Return:

valid physical address on FPGA device **or** 0 on failure

Use the following API when plug-ins need an XRT buffer **object** (BO) corresponding to an VVAS memory **object**.

```
guint gst_ivas_allocator_get_bo (GstMemory *mem)
```

Parameters:

mem - memory to get XRT BO

```
Return:
    valid XRT BO on success or 0 on failure
```

## JSON Schema

This section covers the JSON schema for the configuration files used by the infrastructure plug-ins. For more details, refer to JSON Schema

## VVAS Inference Meta Data

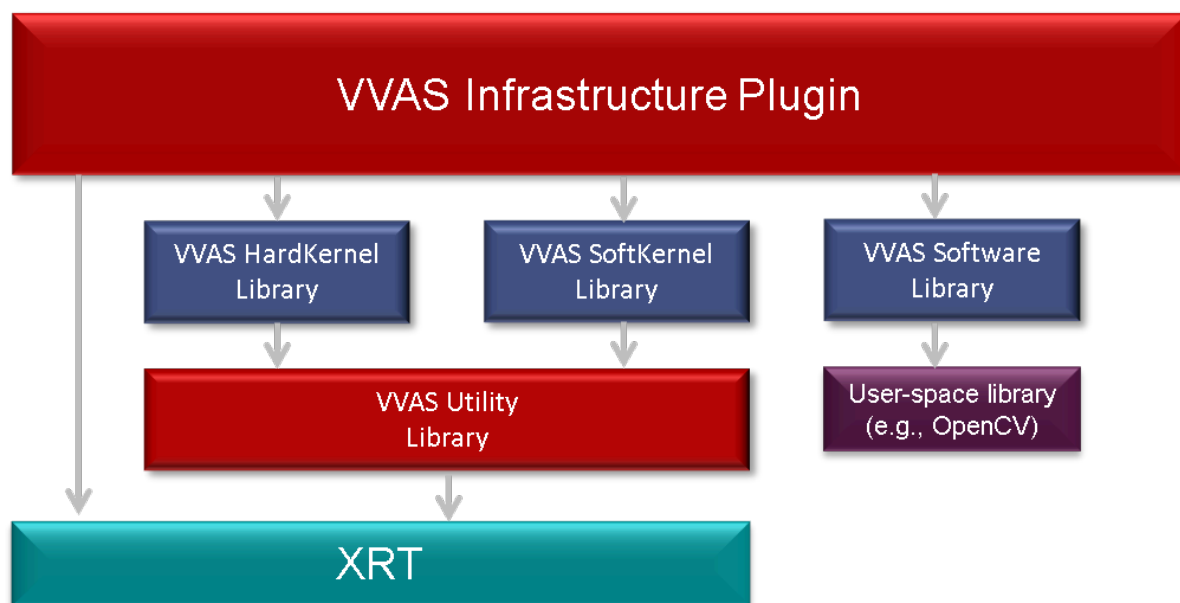
This section covers details about inference meta data structure used to store meta data. For more details, refer to VVAS Inference Meta Data

## 1.3 Development Guide

For advanced developers who wants to develop their Kernel or VVAS acceleration s/w library for their Kernel, this section covers detailed description and steps to achieve that. Acceleration s/w library implements the logic to control the kernel. Each acceleration s/w lib must implement four APIs that are then called by VVAS Infrastructure plugins to interact with the kernel.

### 1.3.1 Acceleration Software Library Development Guide

One of the objectives of VVAS is to define a methodology for advanced developers to develop their own kernel and integrate it into GStreamer-based applications. This section defines the steps required to develop an acceleration software library to control the kernel from the GStreamer application. The acceleration software library manages the kernel state machine as well as provide hooks to control the kernel from the GStreamer plug-ins or an application. There are three types of kernels.



### Types of Kernels

There are three types of kernels as mentioned below.

#### Hard Kernel

A hard kernel is made up of HLS and RTL kernels.

#### Softkernel

These are special kernels that executes on the application Processor of the device and are controlled

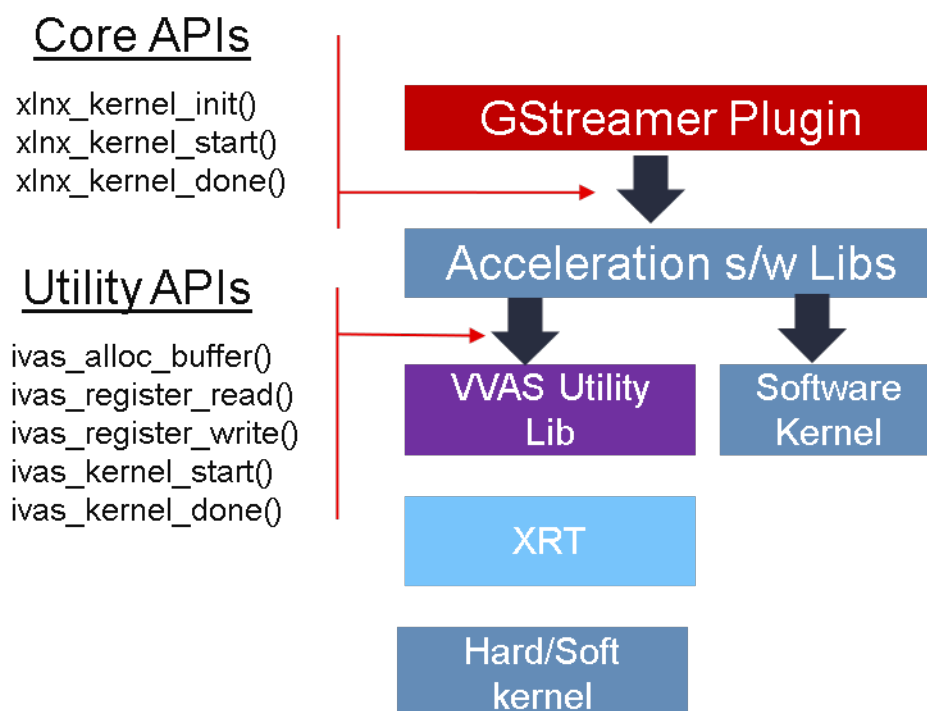
over PCIe interface from the Server. These kernels are not for Embedded platforms.

#### *Software Kernel*

Software kernels are software processing modules that execute fully on the host CPU. Exposing software kernel through the acceleration software library interface allows them to integrate into the GStreamer framework without an in-depth knowledge of the GStreamer framework.

#### Interfaces for Acceleration Software Libraries

To be able to be controlled from an application or GStreamer plug-ins, each acceleration software library must expose three core API functions.



To interact with the kernel, a set of utility APIs are provided.

#### *ivas-utils*

This section covers details about the utility infrastructure required to develop the kernel libraries for a new kernel and to integrate these acceleration software libraries into the GStreamer framework.

The VVAS acceleration software library APIs are C-based APIs designed for ease of use. A developer does not need knowledge of the GStreamer plug-ins. The acceleration software libraries are developed using the following APIs and can be used by the GStreamer VVAS infrastructure plug-ins to realize plug and play functionality of various processing blocks and filters.

The utility sources are hosted in the ivas-utils repository/folder.

```
.
└-- ivas-utils
    |-- README.md
    |-- meson.build
    |-- meson.cross
    └-- utils
        |-- ivas
        |   |-- ivas_kernel.h
        |   |-- ivaslogs.h
        |   └-- ivasmeta.h
        |-- ivas_kernel_utils.c
        |-- meson.build
        |-- xrt_utils.c
        └-- xrt_utils.h
```

## VVAS Datastructures

The following sections list the core structures and enumerations.

### *IVASKernel*

The IVASKernel is handle to kernel context and is created and passed to the core APIs by the GStreamer infrastructure plug-ins (for example: the ivas\_xfilter).

```
typedef struct _ivas_kernel IVASKernel;

struct _ivas_kernel {

void *xcl_handle; /* XRT handle provided by GStreamer

infrastructure plug-in */

uint32_t cu_idx; /* compute unit index of IP/soft- kernel */

json_t *kernel_config; /* kernel specific config from app */ void
*kernel_priv; /* to store kernel specific

structures */

xrt_buffer *ert_cmd_buf; /* ERT command buffer used to submit

execution commands to XRT */

size_t min_offset; size_t max_offset;

IVASBufAllocCBFunc alloc_func; /* callback function to allocate

memory from GstBufferPool by GStreamer infrastructure plug-in */
```

```

IVASBufFreeCBFunc free_func; /* callback function to free memory
allocated by alloc_func */ void *cb_user_data; /* handle to be
passed along with
alloc_func & free_func callback */
ivaspads *padinfo; #ifdef XLNX_PCIE_PLATFORM
uint32_t is_softkernel; /* true when acceleration s/w library is for
#endif
soft-kernel in PCIe platforms only
*/
uint8_t is_multiprocess; /* if true, ERT command buffer will
be used to start kernel. else, direct register programming will be
used */
};

```

#### IVASVideoFormat

The IVASVideoFormat represents the video color formats supported by the VVAS framework. The GStreamer infrastructure plug-ins supports the mapping of the following formats and corresponding GStreamer color formats.

```

typedef enum {
IVAS_VMFT_UNKNOWN = 0,
IVAS_VFMT_RGBX8,
IVAS_VFMT_YUVX8,
IVAS_VFMT_YUYV8, // YUYV
IVAS_VFMT_ABGR8,
IVAS_VFMT_RGBX10,
IVAS_VFMT_YUVX10,
IVAS_VFMT_Y_UV8,
IVAS_VFMT_Y_UV8_420, // NV12
IVAS_VFMT_RGB8,
IVAS_VFMT_YUVA8,
IVAS_VFMT_YUV8,
IVAS_VFMT_Y_UV10,
IVAS_VFMT_Y_UV10_420,
IVAS_VFMT_Y8,
IVAS_VFMT_Y10,
IVAS_VFMT_ARGB8,
IVAS_VFMT_BGRX8,
IVAS_VFMT_UYVY8,
IVAS_VFMT_BGR8, // BGR
IVAS_VFMT_RGBX12,
IVAS_VFMT_RGB16
} IVASVideoFormat;

```

#### IVASFrame

The IVASFrame stores information related to a video frame. The GStreamer infrastructure plug-ins allocate the IVASFrame handle for input and output video frames and sends them to the VVAS kernel processing APIs. Also, the IVASFrame can be allocated by kernel libraries for internal memory requirements (i.e., memory for filter coefficients).

```

typedef struct _ivas_frame_props IVASFrameProps;

```

```
typedef struct _ivas_frame IVASFrame;

// frame properties holds information about video frame
struct _ivas_frame_props {
uint32_t width;
uint32_t height;
uint32_t stride;
IVASVideoFormat fmt;
};
struct _ivas_frame {
uint32_t bo[VIDEO_MAX_PLANES]; // ignore : currently not used
void *vaddr[VIDEO_MAX_PLANES]; // virtual/user space address of
                                //video frame memory
uint64_t paddr[VIDEO_MAX_PLANES]; // physical address of video frame
uint32_t size[VIDEO_MAX_PLANES];
void *meta_data;
IVASFrameProps props; /* properties of video frame */
/* application's private data */
void *app_priv; /* assigned to GstBuffer by GStreamer infrastructure plugin */
IVASMemoryType mem_type;
/*number of planes in props.fmt */
uint32_t n_planes; // number of planes based on color format
};
```

#### Other VVAS Declarations

```
#define MAX_NUM_OBJECT 512 /* max number of video frames/objects
handled by VVAS */
#define MAX_EXEC_WAIT_RETRY_CNT 10 /* retry count on xclExecWait failure */
#define VIDEO_MAX_PLANES 4
typedef enum {
IVAS_UNKNOWN_MEMORY,
IVAS_FRAME_MEMORY, /* use for input and output buffers */
IVAS_INTERNAL_MEMORY, /* use for internal memory of IP */
} IVASMemoryType;
typedef struct buffer {
unsigned int bo; /* XRT Buffer object */
void* user_ptr; /* userspace/virtual pointer */
uint64_t phy_addr; /* physical address */
unsigned int size; /* size of XRT memory */
} xrt_buffer;
```

VVAS acceleration software libraries APIs are broadly categorized into two API types, **Core API** and **Utility API**.

#### Core API

Core APIs are exposed by the VVAS acceleration software library developer through a shared library. The GStreamer VVAS infrastructure plug-ins (ivas\_xfilter and ivas\_xmultisrc) call the APIs to perform operations on the kernel. The following is a list of core APIs.

##### Initialization API

The acceleration software library must perform one-time initialization tasks, such as private handles and internal memory allocations.

```
int32_t xlnx_kernel_init (IVASKernel * handle)
Parameters:
    handle - VVAS kernel handle which has kernel context
Return:
    0 on success or -1 on failure
```

##### Process API

The acceleration software library must perform per frame operations such as updating IP registers or

calling the processing function of the user space library. If a acceleration software library is developed for IP (HardKernel), then this API must call the `ivas_kernel_start()` utility API to issue a command to process the registers using XRT.

```
int32_t xlnx_kernel_start (IVASKernel * handle, int start, IVASFrame *
input[MAX_NUM_OBJECT], IVASFrame * output[MAX_NUM_OBJECT])
```

Parameters:

- `handle` - VVAS kernel handle which has kernel context
- `start` - flag to indicate start of the kernel. Mainly useful in streaming kernel mode
- `input[MAX_NUM_OBJECT]` - Array of input frames populated in IVASFrame handle
- `output[MAX_NUM_OBJECT]` - Array of output frames populated in IVASFrame handle

Return:

- 0 on success or -1 on failure

Note:

1. MAX\_NUM\_OBJECT is 512 and same is assigned in `ivas_kernel.h`
2. Input and output of array is NULL terminated to know number of input & output frames received to start function

The acceleration software library can use the following API to wait for completion of the task that was started using the `xlnx_kernel_start()` API. In the case of a memory-memory IP acceleration software library, this API can leverage the `ivas_kernel_done()` API to check whether an issued command to XRT is completed.

```
int32_t ivas_kernel_done (IVASKernel * handle, int32_t timeout)
```

Parameters:

- `handle` - VVAS kernel handle which has kernel context
- `timeout` - max. time to wait for "kernel done" notification from the kernel.

Return:

- 0 on success or -1 on failure

#### De-Initialization API

The acceleration software library must perform de-initialization tasks such as freeing private handles and internal memory allocation as part of the library initialization process.

```
int32_t xlnx_kernel_deinit (IVASKernel * handle)
```

Parameters:

- `handle` - VVAS kernel handle which has kernel context

Return:

- 0 on success or -1 on failure

#### Utility APIs

For ease of use, the utility APIs are abstracted from the XRT APIs. The following is the list of utility APIs.

##### Memory Management API

The following API must be used to allocate XRT memory for video frames as well as for internal memory requirements (i.e., memory for filter coefficients to be sent to IP). If video frames are requested using `IVAS_FRAME_MEMORY`, then the callback function invoked to the GStreamer VVAS infrastructure plug-ins, like the `ivas_xfilter`, will allocate frames from `GstVideoBufferPool` and `GstIvasAllocator` to avoid memory fragmentation or the memory will be allocated using direct XRT APIs.

```
IVASFrame* ivas_alloc_buffer (IVASKernel *handle, uint32_t size,
IVASMemoryType mem_type, IVASFrameProps *props)
```

**Parameters:**

handle - VVAS kernel handle which has kernel context  
 size - memory size to be allocated  
 mem\_type - memory can be IVAS\_FRAME\_MEMORY or IVAS\_INTERNAL\_MEMORY  
 props - required when requesting IVAS\_FRAME\_MEMORY

**Return:**

IVASFrame handle on success or NULL on failure

The following API is free memory that is allocated using the `ivas_alloc_buffer()` API.

```
void ivas_free_buffer (IVASKernel * handle, IVASFrame *ivas_frame)
```

**Parameters:**

handle - VVAS kernel handle which has kernel context  
 ivas\_frame - IVASFrame handle allocated using `ivas_alloc_buffer()` API

**Return:**

None

### Register Access APIs

The register access APIs are used to directly set or get registers of an IP or ERT command buffer that is sent to XRT. The following API is used write into the registers of an IP or to write into the ERT command buffer that is sent to XRT while starting the kernel execution.

```
void ivas_register_write (IVASKernel *handle, void *src, size_t size,
size_t offset)
```

**Parameters:**

handle - VVAS kernel handle which has kernel context  
 src - pointer to data to be written at offset in ERT command buffer or register at offset from base address of IP  
 size - size of the data pointer src  
 offset - offset at which data to be written

**Return:**

None

The following API used to read from the registers of an IP. This API is not required when `is_multiprocess` enabled in the IVASKernel handle.

```
void ivas_register_read (IVASKernel *handle, void *src, size_t size, size_t
offset)
```

**Parameters:**

handle - VVAS kernel handle which has kernel context  
 src - pointer to data which will be updated after read  
 size - size of the data pointer src  
 offset - offset from base address of an IP

**Return:**

None

### Execution APIs

Execution APIs are used to start kernel execution and wait for the completion of the kernel. These APIs are only used when `is_multiprocess` is enabled in the IVASKernel handle. Use the following API to start IP/kernel execution.

```
int32_t ivas_kernel_start (IVASKernel *handle)
```

**Parameters:**



```

    handle - VVAS kernel handle which has kernel context

Return:
    0 on success -1 on failure

```

Use the following API to check whether the IP or kernel has finished execution. This function internally loops for MAX\_EXEC\_WAIT\_RETRY\_CNT times until a timeout before returning an error.

```

int32_t ivas_kernel_start (IVASKernel *handle)

Parameters:
    handle - VVAS kernel handle which has kernel context

Return:
    0 on success or -1 on failure

int32_t xlnx_kernel_init (IVASKernel * handle)

Parameters:
    handle - VVAS kernel handle which has kernel context

Return:
    0 on success or -1 on failure

```

### Acceleration Software Library for Hard Kernels

This section covers the steps to develop an acceleration software library for hard kernels.

**Note** It is assumed that hard kernel work only on physicle address. Hence Infrastructure plugins will only provide physical address for the input/output buffers. If for any reason one wants to access the input/output buffers in s/w accel lib, then need to map the buffer and get the virtual address.

Virtual address is populated by infrastructure plugins only in case of s/w accel lib for “software only” kernels.

#### Memory Allocation

A hard kernel works on the physically contiguous memory. Use the ivas\_alloc\_buffer API to allocate physically contiguous memory on the device (FPGA).

#### Controlling Kernel

There are two ways to control a kernel, manual mode and automatic mode.

#### Automatic Control Mode

In this mode, VVAS is relying on the underlying XRT framework to write to the kernel registers to start the kernel. The underlying XRT framework ensures that the kernel is not accessed simultaneously by multiple users at any time. This is the recommended mode of operation for kernels that operate on memory buffers. However, there is one limitation. This mode is not suitable for streaming kernels, where kernels need to be started in auto-restart mode. For starting kernels in auto-restart mode, you must use the manual mode.

**Note** To operate a acceleration software library in automatic mode, set the is\_multiprocess flag to True in the kernel initialization API (xlnx\_kernel\_init).

- 

#### Programming Registers

Use the ivas\_register\_write APIs to program the kernel register. In this mode, the registers are not immediately written to. The register value is updated in an internal buffer. The actual registers are updated in response to the kernel start request, described in the following section.

- - **Starting Kernel**

When all the register values are programed, the acceleration software library calls the `ivas_kernel_start` API. The kernel registers are programed and the kernel is started using the XRT command internally by the `ivas_kernel_start` API implementation.
  - **Check Kernel Done Status**

The acceleration software library calls the `ivas_kernel_done` API. The acceleration software library can specify the `time_out` value before returning from this API.

#### *Manual Control Mode*

The manual control mode is used when you need to start the kernel in auto-restart mode, for example, in streaming kernels. The XRT framework does not support this mode. This is achieved by directly writing into the control registers of the kernel. In this mode, you must ensure that the acceleration software library (GStreamer plug-in or application) does not allow the kernel to be accessed simultaneously by more than one thread or process at any time. It can cause unpredictable results.

- - **Programming Registers**

In the manual control mode, use the `ivas_register_read/ivas_register_write` APIs to read from or write to the kernel register. The register is accessed immediately for reading and writing.
  - **Starting Kernel**

Start the kernel by directly writing the appropriate value in the kernel control register.
  - **Check Kernel Done Status**

In this mode, the acceleration software library must either continuously poll the kernel status register using `ivas_register_read`, or to wait on an interrupt to know if the kernel is finished processing.

#### *Acceleration Software Library for the Software Kernel*

Software kernels are software modules that run on the application processor. The acceleration software library for these processing modules do not interact with the XRT interface. The interface APIs that abstract the XRT interface are not needed. You must implement the core API in the acceleration software library for use in the GStreamer application through VVAS infrastructure plug-ins.

#### **Capability Negotiation Support in the Acceleration Software Library**

Kernel capability negotiation is an important functionality that should be accepted between the upstream element and infrastructure plug-ins to achieve an optimal solution. Because the infrastructure plug-ins are generic, the acceleration software library is responsible to populate the required kernel capability during `xlux_kernel_init()`, which is negotiated between the infrastructure plug-ins and the upstream element. The infrastructure plug-in suggests a format on its sink pad and arranges the recommendation in priority order as per the kernel capability in the result from the CAPS query that is performed on the sink pad. Only the `ivas_xfilter` plug-in is currently supporting the kernel specific capability negotiation.

The following section explains the data structures exchange between acceleration software libraries and the infrastructure plug-ins for capability negotiation.

```
typedef struct caps
{
    uint8_t range_height; /* true/false if height is specified in range */
    uint32_t lower_height; /* lower range of height supported,
    range_height=false then this value specified the fixed height supported
    */
}
```

```

uint32_t upper_height; /* upper range of height supported */
uint8_t range_width; /* true/false if width is specified in range */
uint32_t lower_width; /* lower range of width supported,
range_width=false then this value specified the fixed width supported */
uint32_t upper_width; /* upper range of width supported */
uint8_t num_fmt; /* number of format support by kernel */
IVASVideoFormat *fmt; /* list of formats */
} kernelcaps;

typedef struct kernelpad
{
uint8_t nu_caps; /* number of different caps supported */
kernelcaps **kcaps; /* list of caps */
} kernelpads;

```

Below mentioned user friendly APIs are provided for kernel to set the above mentioned capabilities.

API to create new caps with input parameters

```

ivas_caps_new() - Create new caps with input parameters
range_height
    - true  : if kernel support range of height
    - false : if kernel support fixed height
lower_height : lower value of height supported by kernel
               if range_height is false, this holds the fixed value
upper_height : higher value of height supported by kernel
               if range_height is false, this should be 0
range_width  : same as above
lower_width  :
upper_width  :

               : variable range of format supported terminated by 0
               : make sure to add 0 at end otherwise it
               : code will take format till it get 0

kernelcaps * ivas_caps_new (uint8_t range_height,
                           uint32_t lower_height,
                           uint32_t upper_height,
                           uint8_t range_width,
                           uint32_t lower_width,
                           uint32_t upper_width, ...)

```

API to add new caps to sink pad. Only one pad is supported in this release.

```

bool ivas_caps_add_to_sink (IVASKernel * handle, kernelcaps * kcaps, int
sinkpad_num)

```

API to add new caps to src pad. Only one pad is supported as on today.

```

bool ivas_caps_add_to_src (IVASKernel * handle, kernelcaps * kcaps, int
sinkpad_num)

```

### 1.3.2 Acceleration Hardware

This section covers details about the HLS/RTL acceleration kernels. The kernels are HLS code that can be built using Vitis HLS and the compiled kernels are provided as .xo files. These compiled kernels then can be used with any Vitis platform.

#### Kernel Build Steps

The following steps are used to build the kernel.

1. Setup the Vitis HLS 2021.1 software. Refer to [Vitis Software Development Platform 2021.1](#).
2. Edit the makefile to point the PLATFORM\_FILE to any Vitis 2021.1 platform.

3. Edit the options in <kernel\_folder>/kernel\_config.h.
4. Make <kernel name>.

### Multiscaler Kernel

VVAS releases the full HLS source code of the Multiscaler 2.0 IP, it is based on the descriptor approach for programming. This kernel produces a multiscaler kernel `xo/v_multi_scaler.xo`.

#### Kernel Configuration

The multiscaler kernel configuration can be edited by changing the <VVAS\_SOURCES>/ivas-accel-hw/-multiscaler/v\_multi\_scaler\_config.h file. The parameters in the following table can be changed as required. i Table 14: Multiscaler Kernel Configuration

Parameter Macro	Possible Values	Description
H SC_SAMPLES_PER_CLOCK	1, 2, 4	Pixels per clock
HSC_MAX_WIDTH	64 to 7680	Maximum width of the resolution supported
HSC_MAX_HEIGHT	64 to 4320	Maximum height of the resolution supported
HS C_BITS_PER_COMPONENT	8, 10	Bits per component
HSC_SCALE_MODE	0: Bilinear 1: Bicubic 2: Polyphase	Scaling algorithm
HAS_RGBX8_YUVX8	0: Disable 1: Enable	RGBX8 and YUVX8 color format support
HAS_YUYV8	0: Disable 1: Enable	YUYV8 color format support
HAS_RGBA8_YUVA8	0: Disable 1: Enable	RGBA8 and YUVA8 color format support
HAS_RGBX10_YUVX10	0: Disable 1: Enable	RGBX10 and YUVX10 color format support
HAS_Y_UV8_Y_UV8_420	0: Disable 1: Enable	Y_UV8 and Y_UV8_420 color format support
HAS_RGB8_YUV8	0: Disable 1: Enable	RGB8 and YUV8 color format support
H AS_Y_UV10_Y_UV10_420	0: Disable 1: Enable	Y_UV10 and Y_UV10_420 color format support
HAS_Y8	0: Disable 1: Enable	Y8 color format support
HAS_Y10	0: Disable 1: Enable	Y10 color format support
HAS_BGRA8	0: Disable 1: Enable	BGRA8 color format support
HAS_BGRX8	0: Disable 1: Enable	BGRX8 color format support
HAS_UYVY8	0: Disable 1: Enable	UYVY8 color format support
HAS_BGR8	0: Disable 1: Enable	BGR8 color format support
HAS_R_G_B8	0: Disable 1: Enable	R_G_B8 color format support
HAS_Y_U_V8_420	0: Disable 1: Enable	Y_U_V8_420 color format support

### Steps to Build Kernel

1. Source the Vitis HLS 2021.1 software.

2. Edit the makefile to point the PLATFORM\_FILE, (.xpfm) file, to any Vitis 2021.1 platform (tested using the ZCU104 base platform).
3. Edit the options in the multiscaler/v\_multi\_scaler\_config.h.
4. Make v\_multi\_scaler.xo. The generated output will be in the xo folder as xo/ v\_multi\_scaler.xo.

Title	Description
<a href="#">Acceleration s/w Library Development Guide</a>	This section covers the interfaces exposed by VVAS framework to develop the acceleration s/w library. It also covers various types of Kernels supported and how to develop acceleration s/w lib for each type of kernels.
<a href="#">Acceleration H/W Kernels</a>	This section covers the H/W (HLS) kernels supported by the VVAS release. It explains how to change the kernel configuration parameters, how to build these kernels. Ready to use workspace is already created to build the kernel.

## 1.4 Platforms And Applications

This VVAS release supports Zynq MPSoc based Embedded platforms and it has been validated with KV260 SOM and zcu104 platforms. Various supported example applications using VVAS on these platforms are listed below.

### 1.4.1 SOM Examples

There are four Application specific SOM platforms supported. Click on the link below to know more about these.

#### Smart Camera

Smart Camera Application with face detection and display functionality. For more details refer to [Smart Camera](#)

#### AIBox-ReiD

Application demonstrating multistream tracking and Re-Identification. For more details, refer to [AIBox-ReiD](#)

#### Defect Detection

Application to detect defects in mangoes. For more details, refer to [Defect Detect](#)

#### NLP SmartVision

Application to demonstrate Natural Language Processing feature. For more details refer to [NLP SmartVision](#)

### 1.4.2 ZCU104 Examples

#### Multi Channel ML

This application demonstrates multi channel multi model Machine Learning capability using VVAS. [Multi Channel ML Tutorial](#) tutorial for beginners explains step by step approach to build this application/pipeline using VVAS.

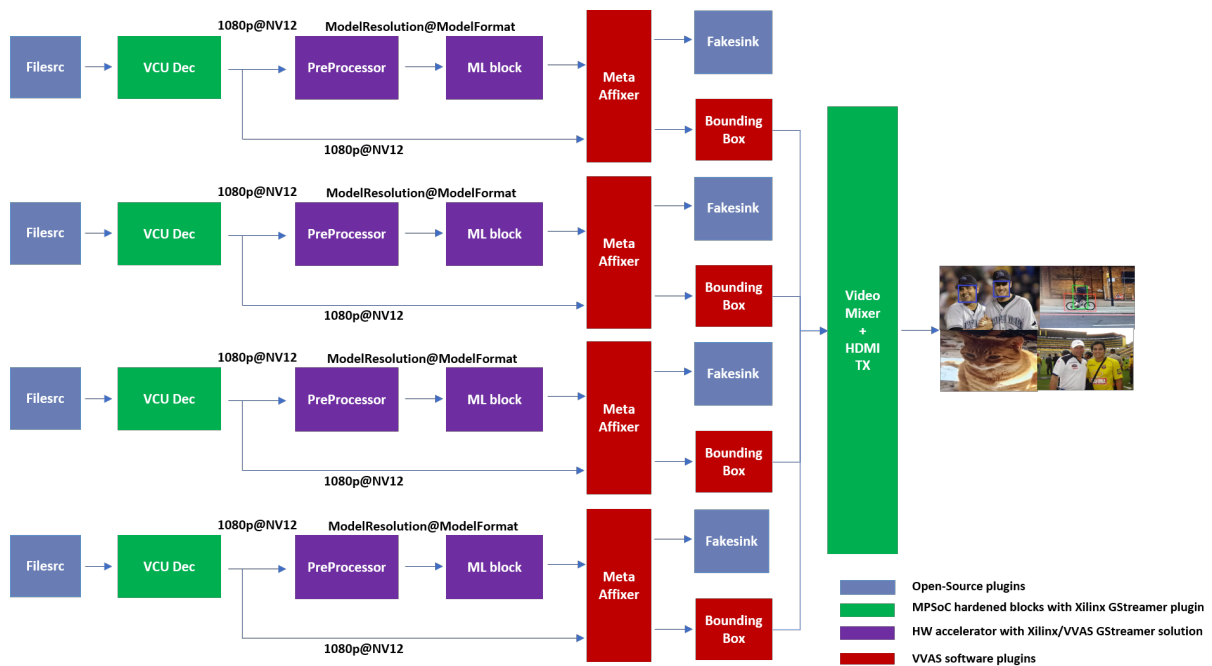
#### Smart Model Select

This application allows selection of ML model from 16 supported models, input source and output using command line and then it constructs and executes the pipeline. For more details, refer to [Smart ML Model Select](#).

#### Multichannel ML

Vitis Video Analytics SDK (VVAS) is a Xilinx framework to build different video analytics solutions on Xilinx platforms. This tutorial begins with building a single stream Machine learning pipeline using VVAS and then scales up to build four channel Machine learning pipeline. The final goal would be to run some ML model on the four H.264 decoded streams and mix the videos and display the four streams on HDMI Monitor. By the end of this tutorial, you should be able to run the following pipe-

line.



## Requirements

### Hardware Requirements

- ZCU104 Evaluation Board Rev 1.0
- Micro USB cable, connected to laptop or desktop computer for the terminal emulator
- MicroSD card, 8 GB or larger, class 10 (recommended)
- HDMI 2.0 supported Monitor with 3840x2160 as the native resolution
- HDMI 2.0 cable

### Software Requirements

(Refer [Vitis Unified Software Development Platform 2021.1 Documentation](#) for installation instructions)

- Vitis™ Unified Software Platform version 2021.1
- Petalinux tool version 2021.1
- Serial terminal emulator (for example, Tera Term)
- Git
- Host system with Ubuntu 18.04/20.04 (Recommended)
- Balena Etcher flashing tool

### System Requirements

- Board must have access to the internet and be accessible from your development system

### Platform

First, list all the components required to bring up the solution. If any of the components are available as a hardware block, select the specific platform.

Because there is a video codec unit (VCU) decoder, Video Mixer and HDMI Tx in the pipeline and these are hardened blocks, hence, a platform with these hard blocks is needed.

This tutorial uses the VVAS `zcu104_vcuDec_vmixHdmiTx` platform because it supports VCU decoder, Video mixer and HDMI Tx subsystem.

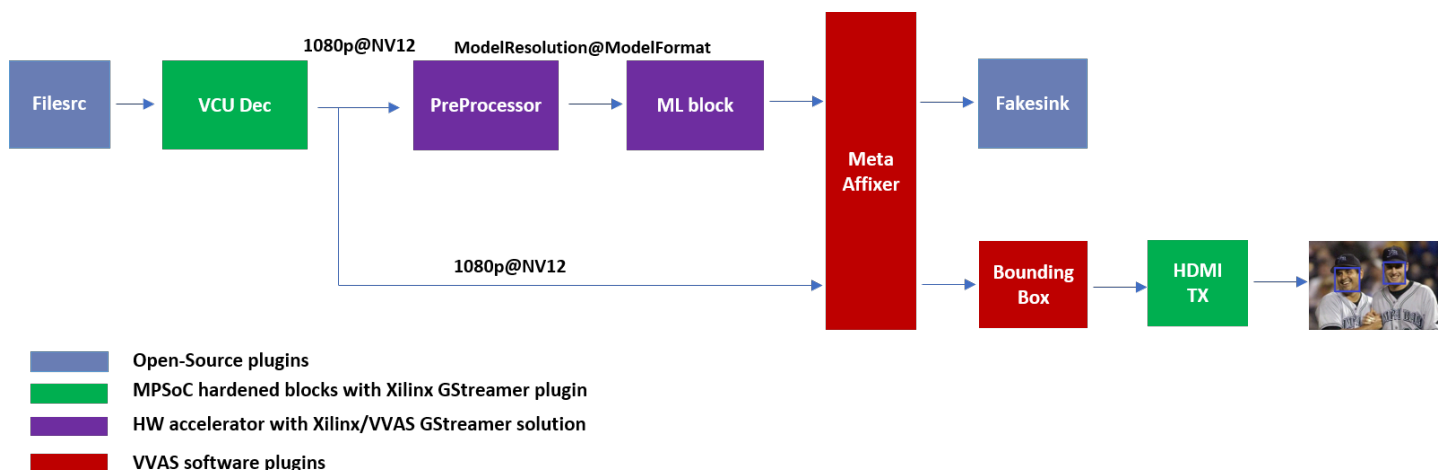
For more information on Vitis platforms, see [Vitis Software Platform](#).

**Note** VVAS platform `zcu104_vcuDec_vmixHdmiTx` may not be performance optimal. This platform is made available as reference and for tutorial demonstration.

**Note** VVAS platform `zcu104_vcuDec_vmixHdmiTx` adds patch to `irps5401` driver for `zcu104` board to support multi thread execution of VAI models. This [patch](#) shouldn't be applied to other boards and is not part of the official Xilinx released 2021.1 Petalinux.

### Building Blocks

Let us begin with constructing a single stream video pipeline based on the components selected.



We shall build the pipeline incrementally, starting from the source element and keep appending the pipeline per the use case.

First setup the ZCU104 board with steps outlined in [Board bring up](#). Facedetect model (`dense-box_320_320`) is used in constructing the single stream pipeline, hence choose a mp4 video file with human faces.

#### VCU Decoder block

A VCU Decoder block is required to decode the H.264/H.265 encoded stream and feed the decoded data to the ML block for inference. For good performance, the hardware VCU block is expected to be part of the Xilinx platform. The `zcu104_vcuDec_vmixHdmiTx` platform provides VCU as a hardware block as part of the design and the `omxh264dec` plugin for decoding. Refer to [pg252](#) for more information on the Xilinx VCU block.



Figure 1.1. VVAS solution for VCU block

Standalone VCU block can be tested with following pipeline:

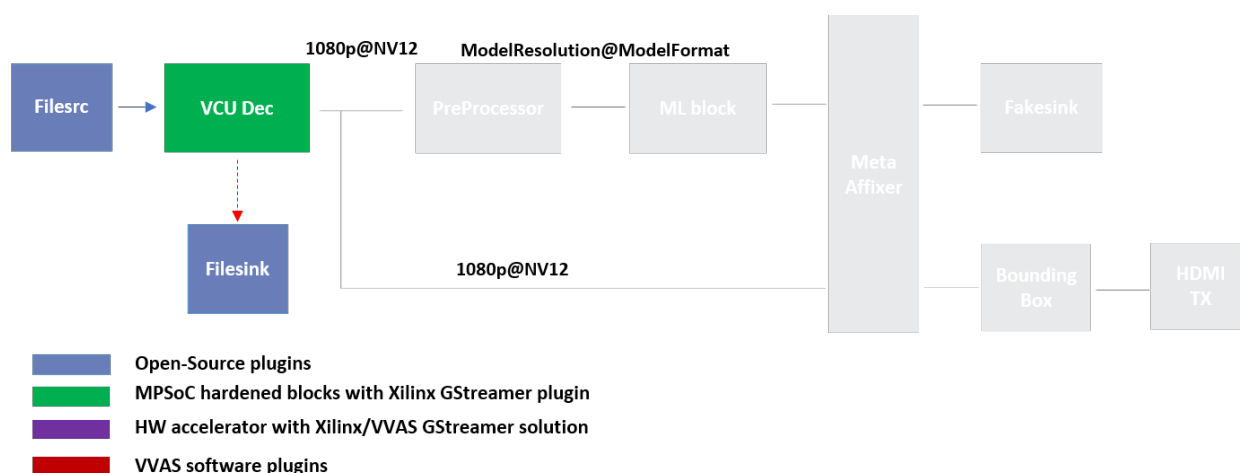


Figure 1.2. Sample video pipeline for VCU block

*Gstreamer command:*

```
gst-launch-1.0 filesrc location=/home/root/videos/face_detect.mp4 ! qtdemux !
h264parse ! omxh264dec internal-entropy-buffers=3 ! filesink
location=./vcu_out.nv12 -v
```

*Preprocessor block*

Different ML models supported by the DPU have different preprocessing requirements that can include resize, mean subtraction, scale normalization etc. Additionally, the DPU has a requirement to be fed with BGR/RGB images. The VCU decoder at the input of the DPU generates NV12 images. Depending on the model selected, the preprocessor block is expected to support the following operations:

- Resize
- Color space conversion
- Mean Subtraction
- Scale Normalization

Although all these operations can be achieved in software, the performance impact is substantial. VVAS support [Multiscaler hardware accelerator](#) using [ivas\\_xabrscler](#) gstreamer plugin.



Figure 1.3. VVAS solution for Preprocessor block

Different models have different requirements for mean and scale values, which can be configured via the plugin properties. Table 1 lists the plugin properties provided by [ivas\\_xabrscler](#) gstreamer plugin to configure mean and scale values. These properties are tested in the context of this tutorial only.

Table 1: [ivas\\_xabrscler](#) Plug-in Properties to configure mean and scale values



Property Name	Type	Range	Default	Description
alpha-b	float	0 to 128	0	Mean subtraction for blue channel
alpha-g	float	0 to 128	0	Mean subtraction for green channel
alpha-r	float	0 to 128	0	Mean subtraction for red channel
beta-b	float	0 to 1	1	Scaling for blue channel
beta-g	float	0 to 1	1	Scaling for green channel
beta-r	float	0 to 1	1	Scaling for red channel

Preprocessor block can be tested with following pipeline:

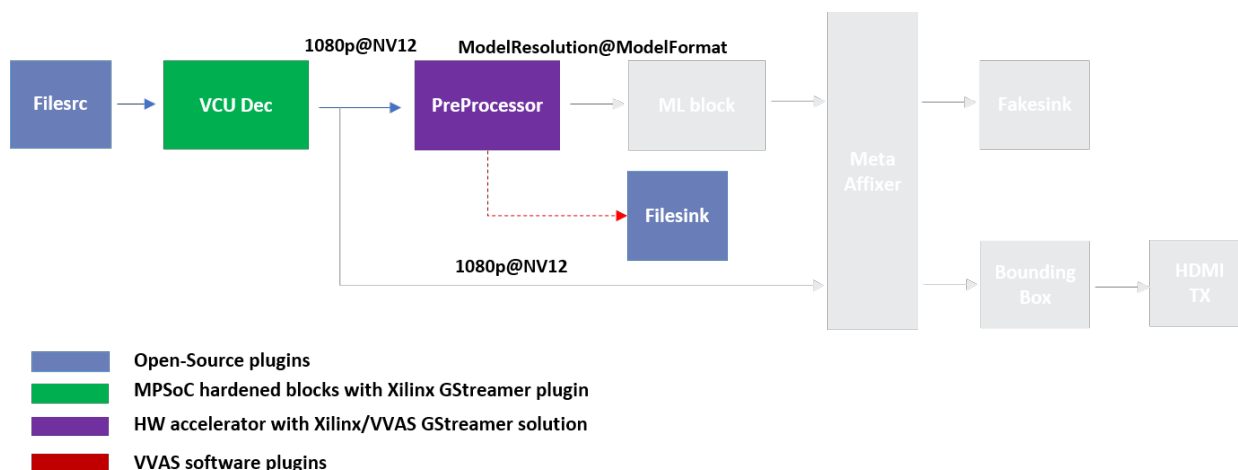


Figure 1.4. Sample Video Pipeline for VCU + Preprocessor block

Gstreamer command:

```
gst-launch-1.0 -v \
  filesrc location=/home/root/videos/face_detect.mp4 ! qtdemux ! h264parse !
  omxh264dec internal-entropy-buffers=3 ! \
  ivas_xabrscler xclbin-location=/media/sd-mmcbk0p1/dpu.xclbin
  kernel-name=v_multi_scaler:v_multi_scaler_1 ! \
  "video/x-raw, width=320, height=320, format=BGR" ! filesink
  location=./pre_proc.bgr
```

**Note** Check the `pre_proc.bgr` file in some raw BGR format reader tool to verify that the output of preprocessor is correct.

The following Gstreamer command is for pre-processing with a mean value of 128, which is required by the FACEDETECT class of DPU.

```
gst-launch-1.0 -v \
  filesrc location=/home/root/videos/face_detect.mp4 ! qtdemux ! h264parse !
  omxh264dec internal-entropy-buffers=3 ! \
  ivas_xabrscler xclbin-location=/media/sd-mmcbk0p1/dpu.xclbin
  kernel-name=v_multi_scaler:v_multi_scaler_1 alpha_r=128 alpha_g=128 alpha_b=128 !
  \
  "video/x-raw, width=320, height=320, format=BGR" ! filesink
  location=./pre_proc.bgr
```

#### Machine Learning (ML) block

Machine Learning inference is performed using DPU hardware accelerator and a gstreamer plug-in is used to control it. VVAS supports the DPU kernel released with Vitis-AI 1.4, and the VVAS infrastruc-

ture plugin `ivas_xfilter` is used along with the `ivas_xdpuinfer` accelerator software library. The beauty of this VVAS solution is that you do not need to figure out the resolution required for various DPU supported models, because the VVAS ML block identifies it dynamically based on the model requested, and negotiates the same resolution with its upstream element. In this case, the upstream element is the Preprocessor block, thus preprocessor converts the input image from the VCU as required by the model selected for the ML block. The model can be selected in the JSON, which is passed to `ivas_xfilter`.

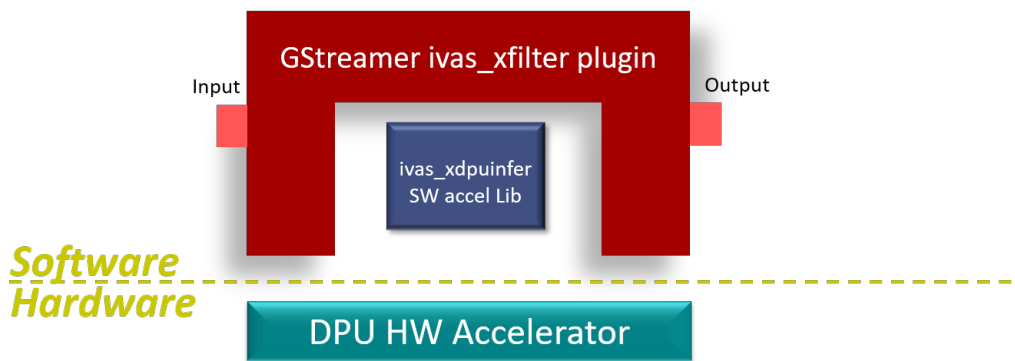


Figure 1.5. VVAS solution for ML block

ML block can be tested with following pipeline:

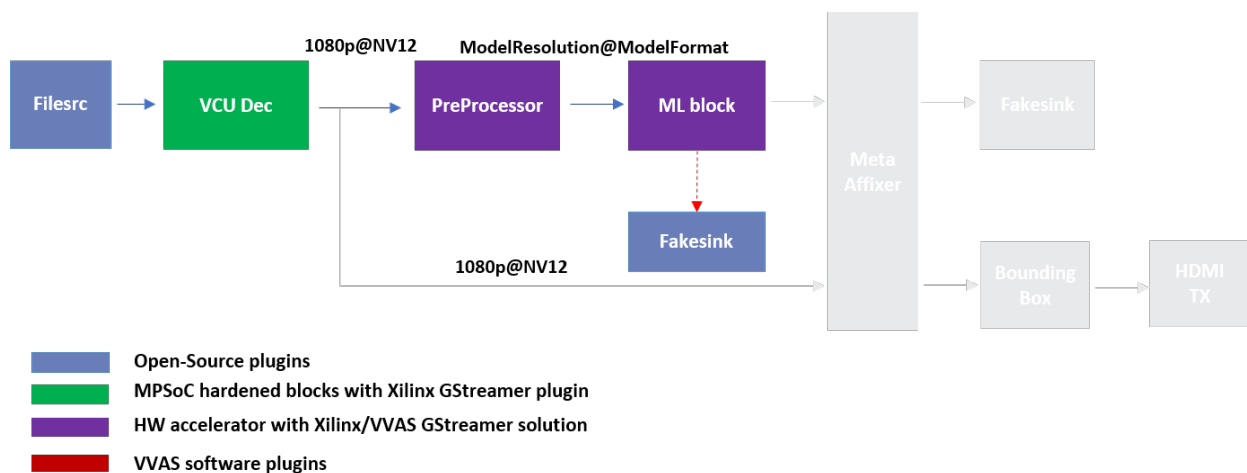


Figure 1.6. Sample Video Pipeline adding ML block

Gstreamer command:

```
gst-launch-1.0 -v \
  filesrc location=/home/root/videos/face_detect.mp4 ! qtdemux ! h264parse !
  omxh264dec internal-entropy-buffers=3 ! \
  ivas_xabrscler xclbin-location=/media/sd-mmcb1k0p1/dpu.xclbin
  kernel-name=v_multi_scaler:v_multi_scaler_1 alpha_r=128 alpha_g=128 alpha_b=128 !
  \
  ivas_xfilter
  kernels-config="/home/root/utls/jsons/dpu/kernel_densebox_320_320.json" !
  fakesink
```

You can observe that the caps mentioned after the `ivas_xabrscler` in the previous pipeline are removed now as the VVAS ML solution auto negotiates the caps based on the model selected. The following is a sample JSON `kernel_densebox_320_320.json` for running the `densebox_320_320` DPU

model that detects a human face.

```
{
  "xclbin-location":"/media/sd-mmcblk0p1/dpu.xclbin",
  "ivas-library-repo": "/usr/lib/",
  "element-mode":"inplace",
  "kernels" :[
    {
      "library-name":"libivas_xdpuinfer.so",
      "config": {
        "model-name" : "densebox_320_320",
        "model-class" : "FACEDETECT",
        "model-format" : "BGR",
        "model-path" : "/usr/share/vitis_ai_library/models/",
        "run_time_model" : false,
        "need_preprocess" : false,
        "performance_test" : false,
        "debug_level" : 1
      }
    }
  ]
}
```

---

**Note** In this pipeline, if the debug\_level of ivas\_xdpuinfer library is increased to 4, you can see the objects detected in logs. The debug level can be increased in the kernel\_densebox\_320\_320.json JSON file. The sample log output is shown below.

```
ivas_xfacedetect.cpp run:84] INFO: RESULT: 53.000000 265.000000 82.000000 309.000000
(0.996612)
ivas_xfacedetect.cpp run:88] DEBUG: prediction tree :

id : 1515,
enabled : True,
bbox : {
  x : 0
  y : 0
  width : 320
  height : 320
}
classes : [
],
predictions : [
{
  id : 1516,
  enabled : True,
  bbox : {
    x : 53
    y : 265
    width : 29
    height : 44
  }
  classes : [
    {
      Id : 1441
      Class : -1
      Label : (null)
      Probability : 0.996612
      Classes : 0
    }
  ],
  predictions : [
  ]
}
],
],
],
]
```

---

Once the correct detection is observed you can move to the next advanced blocks.

#### *Meta Affixer block and HDMI Tx*

In the previous section, the elementary ML pipeline is working but the output image from the preprocessor block might not be the best for display, as several preprocessing operations were done on this image before feeding it to the DPU. To have a good user experience, you must fork the output of the VCU decoder block into two streams, one for the ML block and other for the display. To get the scaled metadata for the original image you need to add one meta scale block, which converts the detection co-ordinates obtained by the ML model for its input resolution with respect to the original output stream from the VCU decoder. This can be done using the [ivas\\_xmetaaffixer](#) plugin which is implemented entirely in software.

You can add HDMI Tx using kmssink Gstreamer plugin along with the [ivas\\_xmetaaffixer](#) in the previous pipeline. This enables viewing video on HDMI monitor. You need to set DRM bus-id, plane-id and rendering position as kmssink properties.

The bus-id for the zcu104\_vcuDec\_vmixHdmiTx platform is fixed to `a0130000.v_mix`.

The video mixer in zcu104\_vcuDec\_vmixHdmiTx platform supports 9 planes of NV12 format, with plane-id starting from 34 to 42. You need to set the plane-id within this range to output the video stream on one of these planes.

The `render-rectangle` property sets the position of video stream on screen in the format “<x, y,

width, height>". Here, x, y represents the starting position of the image on screen, width represents width of the video image, and height represents height of the video image.

Sample video pipeline for adding meta affixer block and HDMI Tx is shown as below

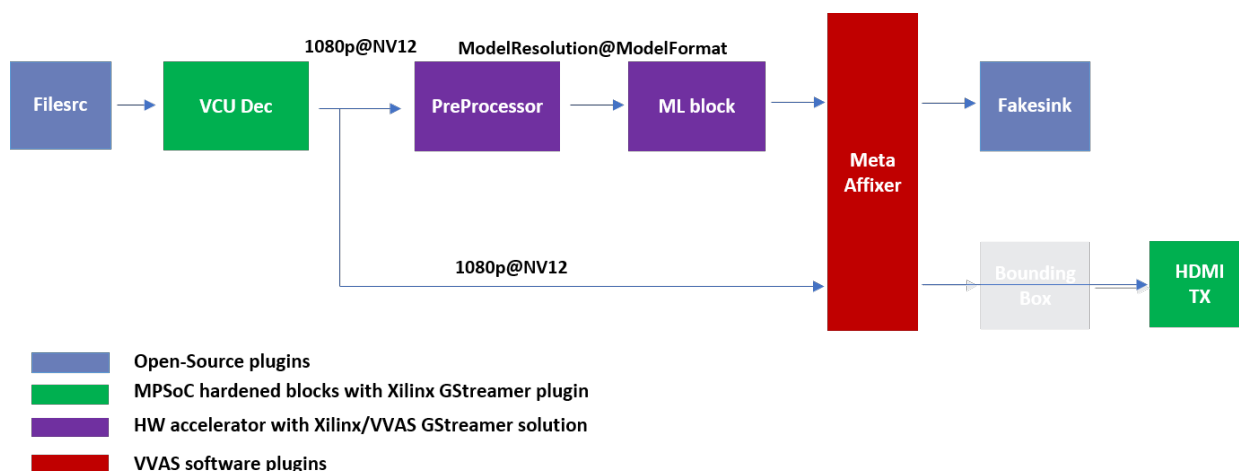


Figure 1.7. Sample video pipeline adding Meta Affixer and HDMI Tx blocks

Gstreamer command:

```

gst-launch-1.0 -v \
  filesrc location=/home/root/videos/face_detect.mp4 ! qtdemux ! h264parse !
  omxh264dec internal-entropy-buffers=3 ! \
  tee name=t0 \
    t0.src_0 ! queue ! \
      ivas_xabrscler xclbin-location=/media/sd-mmcbk0p1/dpu.xclbin
  kernel-name=v_multi_scaler:v_multi_scaler_1 alpha_r=128 alpha_g=128 alpha_b=128 !
  \
    ivas_xfilter
  kernels-config="/home/root/utls/jsons/dpu/kernel_densebox_320_320.json" ! \
    scalem0.sink_master ivas_xmetaaffixer name=scalem0 scalem0.src_master !
  fakesink \
    t0.src_1 ! queue ! \
      scalem0.sink_slave_0 scalem0.src_slave_0 ! queue ! \
      kmssink plane-id=34 bus-id="a0130000.v_mix"
  render-rectangle="<0,0,1920,1080>"
  
```

**Note** It is assumed that the video resolution of the input file sample.mp4 is 1080P.

**Note** Though you may not observe any ML output on monitor with this pipeline, but we should see the input image getting displayed in monitor by this pipeline.

#### Bouding Box block

To have an output of ML displayed on the monitor, you should draw the results on an image. The [ivas\\_xboundingbox](#) software acceleration library comes in handy in this case. This library along with VVAS infrastructure plug-in [ivas\\_xfilter](#) can provide the bounding box functionality.

Sample video pipeline for adding bounding box block is shown as below

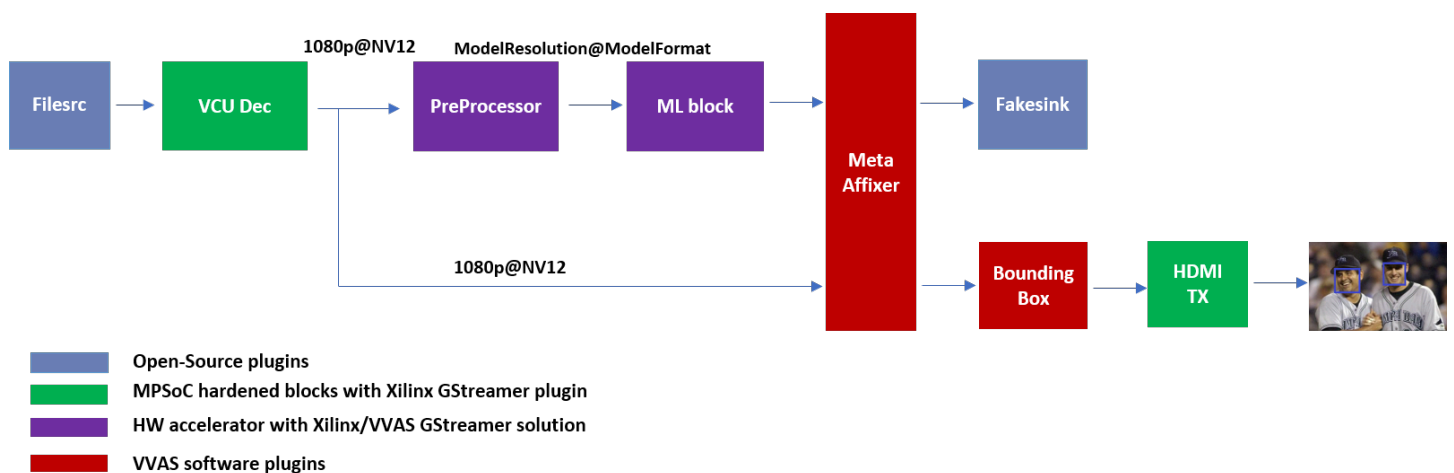


Figure 1.8. Sample Video Pipeline adding Bounding Box block

*Gstreamer command:*

```
gst-launch-1.0 -v \
  filesrc location=/home/root/videos/face_detect.mp4 ! qtdemux ! h264parse !
  omxh264dec internal-entropy-buffers=3 ! \
  tee name=t0 \
    t0.src_0 ! queue ! \
      ivas_xabrscler xclbin-location=/media/sd-mmcbk0p1/dpu.xclbin
  kernel-name=v_multi_scaler:v_multi_scaler_1 alpha_r=128 alpha_g=128 alpha_b=128 !
  \
    ivas_xfilter
  kernels-config="/home/root/utlis/jsons/dpu/kernel_densebox_320_320.json" ! \
    scalem0.sink_master ivas_xmetaaffixer name=scalem0 scalem0.src_master !
  fakesink \
    t0.src_1 ! queue ! \
      scalem0.sink_slave_0 scalem0.src_slave_0 ! queue ! \
      ivas_xfilter
  kernels-config="/home/root/utlis/jsons/bbox/kernel_boundingbox_facedetect.json" !
  \
    kmssink plane-id=34 bus-id="a0130000.v_mix"
  render-rectangle="<0,0,1920,1080>"
```

The following sample JSON file `kernel_boundingbox_facedetect.json` is used to draw a bounding box on detected objects.

```
{
  "xclbin-location": "/media/sd-mmcbk0p1/dpu.xclbin",
  "ivas-library-repo": "/usr/lib/",
  "element-mode": "inplace",
  "kernels": [
    {
      "library-name": "libivas_xboundingbox.so",
      "config": {
        "model-name": "densebox_320_320",
        "display_output": 1,
        "font_size": 0.5,
        "font": 3,
        "thickness": 3,
        "debug_level": 1,
        "label_color": { "blue": 0, "green": 0, "red": 0 },
        "label_filter": [ "class", "probability" ],
        "classes": [
```

```

    }
  }
]
}

```

With addition of bounding box, your pipeline for single stream is complete.

#### *Four Channel ML pipeline*

Now, constructing a four channel pipeline is simply duplicating the above pipeline four times for different models and positioning each output video appropriately on screen on different plane-ids.

Below Vitis AI models are used as example in this tutorial. Refer [Vitis AI User Documentation](#) to compile different models using arch.json file from release package.

- densebox\_320\_320 (Face detection)
- yolov3\_adas\_pruned\_0\_9 (Object detection)
- resnet50 (Classification)
- refinedet\_pruned\_0\_96 (Pedestrian detector)

A reference pipeline for four channel ML is given below.

```

gst-launch-1.0 -v \
  filesrc location=/home/root/videos/face_detect.mp4 ! qtdemux ! h264parse ! \
  omxh264dec internal-entropy-buffers=3 ! queue ! \
  tee name=t \
    t.src_0 ! queue ! \
    ivas_xabrscaler xclbin-location=/media/sd-mmcbk0p1/dpu.xclbin
kernel-name=v_multi_scaler:v_multi_scaler_1 alpha_r=128 alpha_g=128 alpha_b=128 !
queue ! \
  ivas_xfilter
kernels-config="/home/root/utls/jsons/dpu/kernel_densebox_320_320.json" ! queue !
\
  scalem.sink_master ivas_xmetaaffixer name=scalem scalem.src_master ! queue
! fakesink \
  t.src_1 ! queue ! \
  scalem.sink_slave_0 scalem.src_slave_0 ! queue ! \
  ivas_xfilter
kernels-config="/home/root/utls/jsons/bbox/kernel_boundingbox_facedetect.json" !
\
  fpsdisplaysink video-sink="kmssink plane-id=34 bus-id=a0130000.v_mix
render-rectangle=<0,0,1920,1080>" text-overlay=false sync=false \
  filesrc location=/home/root/videos/yolov3.mp4 ! qtdemux ! h264parse ! \
  omxh264dec internal-entropy-buffers=3 ! queue ! \
  tee name=t2 \
    t2.src_0 ! queue ! \
    ivas_xabrscaler xclbin-location=/media/sd-mmcbk0p1/dpu.xclbin
kernel-name=v_multi_scaler:v_multi_scaler_1 alpha_r=0 alpha_g=0 alpha_b=0
beta_r=0.25 beta_g=0.25 beta_b=0.25 ! queue ! \
  ivas_xfilter
kernels-config="/home/root/utls/jsons/dpu/kernel_yolov3_adas_pruned_0_9.json" !
queue ! \
  scalem2.sink_master ivas_xmetaaffixer name=scalem2 scalem2.src_master !
queue ! fakesink \
  t2.src_1 ! queue ! \
  scalem2.sink_slave_0 scalem2.src_slave_0 ! queue ! \
  ivas_xfilter
kernels-config="/home/root/utls/jsons/bbox/kernel_boundingbox_yolov3_adas_pruned_0_9.json"
! \
  fpsdisplaysink video-sink="kmssink plane-id=35 bus-id=a0130000.v_mix
render-rectangle=<1920,0,1920,1080>" text-overlay=false sync=false \
  filesrc location=/home/root/videos/classification.mp4 ! qtdemux ! h264parse ! \

```

```

omxh264dec internal-entropy-buffers=3 ! queue ! \
tee name=t3 \
t3.src_0 ! queue ! \
ivas_xabrscaler xclbin-location=/media/sd-mmcbld0pl/dpu.xclbin
kernel-name=v_multi_scaler:v_multi_scaler_1 alpha_r=104 alpha_g=107 alpha_b=123
beta_r=1 beta_g=1 beta_b=1 ! queue ! \
ivas_xfilter
kernels-config="/home/root/utls/jsons/dpu/kernel_resnet50.json" ! queue ! \
scalem3.sink_master ivas_xmetaaffixer name=scalem3 scalem3.src_master !
queue ! fakesink \
t3.src_1 ! queue ! \
scalem3.sink_slave_0 scalem3.src_slave_0 ! queue ! \
ivas_xfilter
kernels-config="/home/root/utls/jsons/bbox/kernel_boundingBox_resnet50.json" ! \
fpsdisplaysink video-sink="kmssink plane-id=36 bus-id=a0130000.v_mix
render-rectangle=<0,1080,1920,1080>" text-overlay=false sync=false \
filesrc location=/home/root/videos/refinedet.mp4 ! qtdemux ! h264parse ! \
omxh264dec internal-entropy-buffers=3 ! queue ! \
tee name=t4 \
t4.src_0 ! queue ! \
ivas_xabrscaler xclbin-location=/media/sd-mmcbld0pl/dpu.xclbin
kernel-name=v_multi_scaler:v_multi_scaler_1 alpha_r=104 alpha_g=117 alpha_b=123
beta_r=1 beta_g=1 beta_b=1 ! queue ! \
ivas_xfilter
kernels-config="/home/root/utls/jsons/dpu/kernel_refinedet_pruned_0_96.json" !
queue ! \
scalem4.sink_master ivas_xmetaaffixer name=scalem4 scalem4.src_master !
queue ! fakesink \
t4.src_1 ! queue ! \
scalem4.sink_slave_0 scalem4.src_slave_0 ! queue ! \
ivas_xfilter
kernels-config="/home/root/utls/jsons/bbox/kernel_boundingBox_refinedet_pruned_0_96.json"
! \
fpsdisplaysink video-sink="kmssink plane-id=37 bus-id=a0130000.v_mix
render-rectangle=<1920,1080,1920,1080>" text-overlay=false sync=false

```

The above command is available in the release package as `multichannel_ml.sh`.

Now, let's look into implementing the design and executing using Vitis AI and VVAS.

#### Implementation

**Release package** provides prebuilt binaries including SD card image that has the implemented design and required software, VAI models and scripts. Download the release package. Let the path where release package is downloaded be represented as `<RELEASE_PATH>`.

Below steps are required only if the platform and example design needs to be regenerated, else move to section [Board bring up](#) to try the released SD card image.

#### Platform

As previously described, this solution uses the VVAS `zcu104_vcuDec_vmixHdmiTx` platform. The first and foremost step is to build this platform from its sources.

The platform provides the following hardware and software components of the pipeline:

- VCU hardware
- Video Mixer and HDMI Tx hard block
- Opensource framework like Gstreamer, OpenCV
- Vitis AI 1.4 libraries
- Xilinx Runtime (XRT)
- omxh264dec Gstreamer plugin
- kmmsink Gstreamer plugin



- VVAS Gstreamer plugins and libraries
  - [ivas\\_xfilter](#) gstreamer plugin
  - [ivas\\_xabrscler](#) multiscaler accelerator plugin
  - [ivas\\_xdpuinfer](#) software accelerator library
  - [ivas\\_xboundingbox](#) software accelerator library
  - [ivas\\_xmetaaffixer](#) gstreamer plugin

Steps for building the platform:

1. Download the VVAS git repository. Let the path where VVAS repo is downloaded be represented as <VVAS\_REPO>.

```
git clone https://github.com/Xilinx/VVAS.git
```

2. Setup the toolchain

```
source <2021.1_Vitis>/settings64.sh
source <2021.1_Petalinux>/settings.sh
source <2021.1_XRT>/setenv.sh
```

3. Change directory to the platform

```
cd <VVAS_REPO>/VVAS/ivas-platforms/Embedded/zcu104_vcuDec_vmixHdmiTx
```

4. Build the platform

```
make
```

After the build is finished, the platform is available at <VVAS\_REPO>/VVAS/ivas-platforms/Embedded/zcu104\_vcuDec\_vmixHdmiTx/platform\_repo/xilinx\_

Let the path to platform be represented as <PLATFORM\_PATH>.

*Vitis Example Project*

A Vitis build is required to stitch all the discussed hardware accelerators to the platform design. The hardware accelerators required are:

1. DPU (Xilinx ML IP)
2. Multiscaler (Xilinx Preprocessing IP)

The Xilinx deep learning processor unit (DPU) is a configurable computation engine dedicated for convolutional neural networks. Refer to [DPU-TRD](#) for more information and compiling the DPU accelerator.

The `multichannel_ml` example design adds two instances of B3136 DPU configuration and a single instance of Multiscaler to the `zcu104_vcuDec_vmixHdmiTx` platform.

Steps for building Vitis example project:

1. Download Vitis-AI. Let the path where Vitis-AI is downloaded be represented as <VITIS\_AI\_REPO>.

```
git clone https://github.com/Xilinx/Vitis-AI.git
cd Vitis-AI/
git checkout tags/v1.4 -b v1.4
```

2. Change directory to example project

```
cd <VVAS_REPO>/VVAS/ivas-examples/Embedded/multichannel_ml/
```

3. Compile the project

```
make PLATFORM=<PLATFORM_PATH>/xilinx_zcu104_vcuDec_vmixHdmiTx_202110_1.xpfm
DPU_TRD_PATH=<VITIS_AI_REPO>/Vitis-AI/dsa/DPU-TRD/
HW_ACCEL_PATH==<VVAS_REPO>/VVAS/ivas-accel-hw/
```

---

**Note** Depending on the build machine capacity, building this example project can take about 3 or more hours to compile.

---

Once the build is completed, you can find the sdcard image at <VVAS\_REPO>/VVAS/ivas-examples/Embedded/multichannel\_ml/binary\_container\_1/sd\_card.img. *Board bring up*

1. Burn the SD card image sd\_card.img (Either from [Release package](#) or generated) using a SD card flashing tool like dd, Win32DiskImager, or Balena Etcher. Boot the board using this SD card.

2. Once the board is booted, resize the ext4 partition to extend to full SD card size.

```
resize-part /dev/mmcblk0p2
```

3. From the host system, copy the video files on the board.

```
mkdir -p ~/videos
scp -r <Path to Videos> root@<board ip>:~/videos
```

---

**Note** Password for root user is root.

---

---

**Note** Video files are not provided as part of release package.

---

4. Copy the model json files and scripts on the board

```
scp -r <RELEASE_PATH>/vvas_multichannel_ml_2021.1_zcu104/utils/ root@<board ip>:~
```

5. Copy the Vitis-AI model files on board

```
mkdir -p /usr/share/vitis_ai_library/models
scp -r <RELEASE_PATH>/vvas_multichannel_ml_2021.1_zcu104/vai_models/*
/usr/share/vitis_ai_library/models/
```

6. Execute four channel Gstreamer pipeline script

```
sh ~/utils/scripts/multichannel_ml.sh
```

You can now see the 4 channel mixed video on the HDMI monitor.

### Known Issues

1. Sometimes on reboot, the HDMI screen remains blank and following error log is observed during the boot. Power OFF and ON (Hard reboot) the board to fix the issue.

```
xlnx-drm-hdmi a0100000.v_hdmi_tx_ss: tx-clk not ready -EPROBE_DEFER
```

### References

1. <https://github.com/Xilinx/Vitis-AI>
2. [https://www.xilinx.com/html\\_docs/vitis\\_ai/1\\_4](https://www.xilinx.com/html_docs/vitis_ai/1_4)
3. <https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design>
4. <https://www.xilinx.com/products/boards-and-kits/zcu104.html>
5. [https://www.xilinx.com/support/documentation/ip\\_documentation/vcu/v1\\_2/pg252-vcu.pdf](https://www.xilinx.com/support/documentation/ip_documentation/vcu/v1_2/pg252-vcu.pdf)

6. <https://gstreamer.freedesktop.org>
7. <https://www.kernel.org/doc/html/v4.13/gpu/drm-kms.html>
8. <https://gstreamer.freedesktop.org/documentation/kms/index.html>

## Smart Model Select

Smart Model Select is an example application to demonstrate inferencing using 16 Machine Learning (ML) models supported by VVAS. User can select the input source, ML model to be used and the output option, like display on monitor or dump the results to file.

Smart Model Select application design is built on zcu104 development board which, along with VVAS, provides a complete framework for building and customizing video platforms with different pipelines stages. zcu104 development board can support below mentioned pipeline stages:

- Capture Pipelines
- Video Processing Pipelines
- Acceleration Pipelines
- Output Pipelines

Smart Model Select design doesn't support Capture Pipeline.

### Features

Smart Model Select application supports below mentioned features

- Supports 16 ML Models - resnet50 - resnet18 - mobilenet\_v2 - inception\_v1 - ssd\_adas\_pruned\_0\_95 - ssd\_traffic\_pruned\_0\_9 - ssd\_mobilenet\_v2 - ssd\_pedestrian\_pruned\_0\_97 - tiny\_yolov3\_vmss - yolov3\_voc\_tf - yolov3\_adas\_pruned\_0\_9 - refinedet\_pruned\_0\_96 - yolov2\_voc - yolov2\_voc\_pruned\_0\_77 - densebox\_320\_320 - densebox\_640\_360
- Display Port for display
- H264/H265 Decode
- Input can be from file, rtsp source

### Prerequisites

Before attempting the execution of application, please ensure that all the pre-requisites mentioned below are fulfilled.

### Hardware Requirements

- ZCU104 Evaluation Board Rev 1.0
- Micro USB cable, connected to laptop or desktop computer for the terminal emulator
- Micro SD card, 8 GB or larger, class 10 (recommended)
- Display port 1.2 supported Monitor with 1920x1080 as max resolution
- Display port 1.2 cable

### Software/Tools Requirements

- Serial terminal emulator (for example, Tera Term)
- Balena etcher or equivalent flashing tool to flash the SD Card image

### System Requirements

- Board must have access to the internet and be accessible from your development system

### Application Design

Smart Model Select application design has a platform and integrated accelerator functions. Platform is nothing but collection of hardware as well as software components required to build a solu-

tion. Prebuilt design for `Smart Model Select` has been provided as part of this VVAS release. You may download this ready to use design by following the link [Download pre-built binaries](#) and directly jump to [Preparing Setup](#) section describing how to prepare the setup and execute the example application.

If for some reason, one wants to build the design from scratch, then refer to the section [Build Design](#) that describes the steps to build the platform, accelerator functions and finally integrate these to create complete Design for *Smart Camera Select* application.

---

**Note** VVAS platform `zcu104_vcuDec_DP` may not be performance optimal. This platform is made available as reference along with *Smart Model Select Application*.

---

---

**Note** VVAS platform `zcu104_vcuDec_DP` adds patch to `irps5401` driver for `zcu104` board to support multi thread execution of VAI models. This [patch](#) shouldn't be applied to other boards and is not part of the official Xilinx released 2021.1 Petalinux.

---

#### *Pre-built binaries*

Ready to use Prebuilt binaries are provided with this VVAS release. You can download these binaries and required supporting files and quickly run the example application.

Create a folder, say `PREBUILT_BINARIES`, where pre-built binaries are to be downloaded.

Click on [Smart Model Select Prebuilt Binaries](#). This will ask for few credentials and then start downloading.

Unzip the downloaded package `vvas_smart_model_select_2021.1_zcu104.zip`.

```
unzip vvas_smart_model_select_2021.1_zcu104.zip
```

Prebuilt binaries package includes

- `sd_card.img`: Image to be flashed on the SD Card on the `zcu104` board.
- `sdk.sh`: This is required to generate `sysroot`. `Sysroot` is required **only** if one wants to build the VVAS plugins and libraries. You do not need this if you simply wants to execute the pre-built application.
- `models`: Supported DPU models on this platform.
- `app`: Contains application executable and input configuration/json files.
- `arch.json`: Represents DPU architecture.

Once you have downloaded the prebuilt binaries, you need to prepare the setup to execute the application, as shown in the next section.

#### *Preparing the setup*

It is assumed that all the pre-requisites are fulfilled and we are ready to setup the board and execute the example application. There are few steps that are required only for the first time when the `zcu104` board is not flashed with the `sd_card` image for `smart_model_select` application. You may skip these steps if `zcu104` board has already been flashed with the required `sd_card` image.

#### *One time setup*

- Flash the SD Card with the `sd_card.img` using any SD card flashing tool like `dd`, `Win32DiskImager`, or `BalenaEtcher`.

If using pre-built binaries, then `sd_card.img` is located as

```
<PREBUILT_BINARIES>/vvas_smart_model_select_2021.1_zcu104/sd_card.img
```

If you have built the platform yourself, then `sd_card.img` would be located at

```
<VVAS_SOURCES>/VVAS/ivas-examples/Embedded/smart_model_select/binary_container_1/sd_card.img
```

- Insert this SD card in the SD card slot on the zcu104 board and boot the board.
- After booting up the board, run below command to extend the root filesystem partition to its 100% size. This is useful in copying input video streams and storing output files from example application.

```
resize-part /dev/mmcblk0p2
```

- Copy the <PREBUILT\_BINARIES>/vvas\_smart\_model\_select\_2021.1\_zcu104/app folder of the application onto *home* folder of the board. .. code-block:

```
scp -r <PREBUILT_BINARIES>/vvas_smart_model_select_2021.1_zcu104/app
root@<board ip>:~/
```

- After copying, `chmod ~/app/setup.sh` and `~/app/smart_model_select` to make them executable on the board. .. code-block:

```
chmod 777 ~/app/smart_model_select
chmod 777 ~/app/setup.sh
```

- Create `/usr/share/vitis_ai_library/models` folder on the board and copy the Vitis-AI models into it

```
mkdir -p /usr/share/vitis_ai_library/models
scp -r <PREBUILT_BINARIES>/vvas_smart_model_select_2021.1_zcu104/models/*
root@<board ip>/usr/share/vitis_ai_library/models/
```

### Running the application

This section will elaborate on the usage of the application and various options with it.

Every time the board is booted, execute the steps mentioned below

- Run `setup.sh` on the board, this will set the alpha channel of the display port and copy label json files related to each model.

```
cd /home/root/app/
./setup.sh
```

- Export the environment variable "XCLBIN\_PATH" with path pointing to xclbin. If its not exported, `/media/sd-mmcblk0p1/dpu.xclbin` will be the default xclbin path.
- Run the command to execute the application .. code-block:

```
./smart_model_select
```

When the application starts executing, you can observe the Fig 1 coming up in the display. This menu displays various models supported by the application and options on either side of the image are for input sources and output sinks supported. All these options carry an index number alongside, which user need to enter in sequence to create the pipeline of choice.

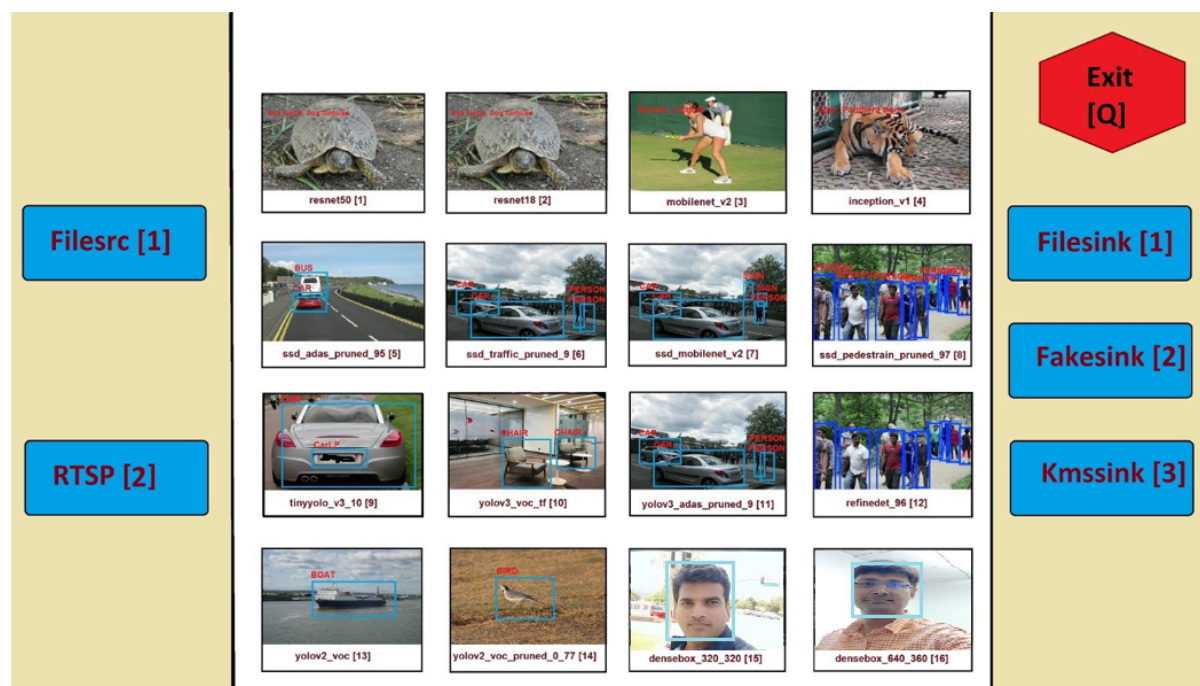


Fig 1: Menu image of the application

Below command line prompt will appear on console/command prompt when the application starts, which will accept the input options for creating the Gstreamer pipeline of choice. As described below, user has to enter four options in the sequence of input source, ML model to be used, output sink and a field to enable/disable performance mode. Example the sequence "1,2,3,0", tells that the source is "filesrc", ML model to be used is "resnet18", sink is "kmsink" and 0 is to disable performance mode.

```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Menu displayed on the monitor shows various options available
for input source, ML model, output sink. Each option carry an
index number along side.
Select elements to be used in the pipeline in the sequence of
"input source, ML model, output sink and performance
mode flag" seperated by commas.
eg input: 1,1,3,0
Above input will run "filesrc" input, "resnet50" model
"kmssink" used as output sink and performance mode disabled.
Enter 'q' to exit
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```

**Note** If performance mode is enabled then the sink type should always be fakesink. Otherwise pipeline will not execute.

If performance mode is enabled, there will be 4 ML pipelines executed simultaneously feeding DPU from multiple threads. This results in improved performance compared to when using DPU from single thread.

Followed by the selection of ML model, source and sink elements, next step is the option to provide input filename or RTSP URL as applicable, to be processed.

If the selected input is file source, then by default application will look for the input video files under *app/videos* folder. Create the folder */home/root/app/videos* and copy the video files here with names given below.

```

videos/
+-- CLASSIFICATION.mp4

```

```

+-- FACEDETECT.mp4
+-- REFINEDET.mp4
+-- SSD.mp4
+-- YOLOV2.mp4
+-- YOLOV3.mp4

```

If the file is not available in this folder then application will prompt for the input file. All files must be named after ML model type as given below. User has to enter the input file location in response to the below message prompt.

```
Enter the input filename to be processed
```

If the selected input source is “RTSP”, then application will prompt for entering “RTSP” URL.

```
Enter the RTSP url to be processed
```

The application supports RTSP input with RTP packets containing H264 payload of resolution 1920x1080. One can download and setup Gstreamer RTSP server or VLC can also be used to serve RTSP data. Follow below steps to compile Gstreamer RTSP server. For successful compilation of Gstreamer RTSP server, Gstreamer framework must be installed as a prerequisite.

```

1. wget
   https://gstreamer.freedesktop.org/src/gst-rtsp-server/gst-rtsp-server-1.16.2.tar.xz
2. tar -xvf gst-rtsp-server-1.16.2.tar.xz
3. cd gst-rtsp-server-1.16.2/
4. ./autogen.sh --disable-gtk-doc
5. make

```

Examples in `gst-rtsp-server-1.16.2/examples` can be used to serve RTSP data. Refer below example

```

cd gst-rtsp-server-1.16.2/examples
./test-launch "filesrc location=<File with H264 1080p in MP4 format> ! qtdemux !
h264parse ! rtph264pay name=pay0 pt=96"

```

Streaming starts on the URL `rtsp://<RTSP server ip address>:8554/test`. Enter the same URL as input to the application.

Application supports multiple sink options as well. If `kmssink` is used, output video will be rendered on the display monitor connected. If `filesink` is chosen the output will get dumped to a file by name “output.nv12” in the current directory. On the other hand, `fakesink` acts a black hole for the data with no overhead.

Below Fig 2 is the pictorial depiction of a typical pipeline that is created by the application.

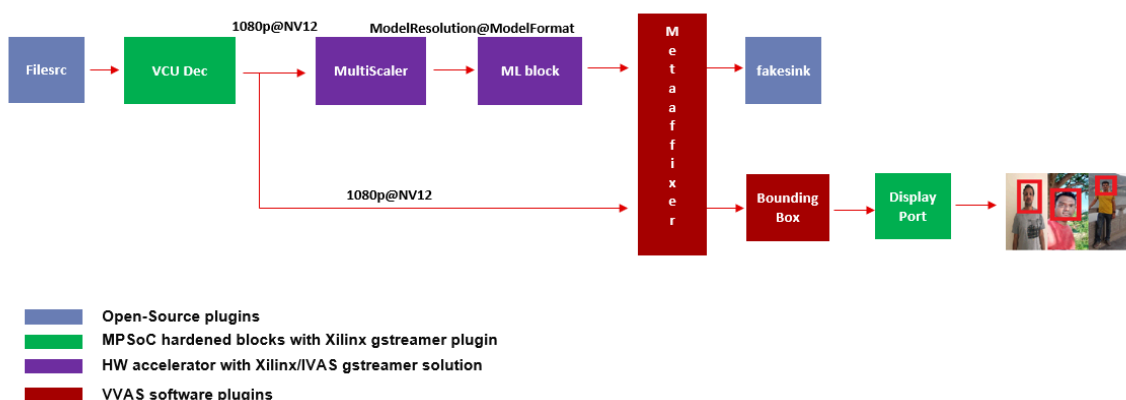


Fig 2: Typical Gstreamer pipeline that application creates  
*Build Design*

The Design consists of a base platform and integrated accelerator functions (Kernels).



### *Base Platform*

Smart Model Select application requires video decoding, resizing the decoded frames, Machine Learning and finally display the output. Hence we need a platform that fulfills these requirements. This VVAS release has zcu104\_vcuDec\_DP base platform that meets the requirements of decode and display. This platform has hardware accelerated video decoding IP, Video Codec Unit VCU. For display, this platform supports Display Port.

In addition to the above mentioned hardware components, zcu104\_vcuDec\_DP supports following software components

- omxh264dec Gstreamer plugin
- Opensource framework like Gstreamer, OpenCV
- Vitis AI 1.4 libraries
- Xilinx Run Time (XRT)

### *Compiling base platform*

Complete sources along with workspace to build the platform is provided as part of this release. Follow the steps mentioned below to compile the platform.

1. Navigate to the folder, say VVAS\_SOURCES, where you want to clone the VVAS source tree.
2. Clone VVAS repo

```
git clone https://github.com/Xilinx/VVAS.git
```

3. Setup tool chain environment:

```
source <2021.1_Vitis>/settings64.sh
source <2021.1_Petalinux>/settings.sh
source <2021.1_XRT>/setenv.sh
```

4. Navigate to zcu104\_vcuDec\_DP platform folder

```
cd <VVAS_SOURCES>/VVAS/ivas-platforms/Embedded/zcu104_vcuDec_DP
```

5. Compile the platform

```
make
```

After build is finished, platform will be available <VVAS\_SOURCES>/VVAS/ivas-platforms/Embedded/zcu104\_vcuDec\_DP/platform\_repo/xilinx\_zcu104\_vcuDec\_DP\_202110\_1/export/xilinx\_zcu104\_vcuDec\_DP\_202110\_1/ location.

### *Hardware Accelerators (Kernels)*

Smart Model Select application's requirements of Machine Learning and Resize operations are fulfilled by below mentioned accelerators (Kernels):

- DPU (Deep Learning Processing Unit) for Machine Learning.
- Multiscaler for Preprocessing operation

### *Compiling Hardware Accelerators (Kernels)*

The sources for hardware accelerators required for Smart Model Select application can be made available as mentioned below:

1. Navigate to <VVAS\_SOURCES>
2. DPU Kernel sources can be cloned from

```
git clone https://github.com/Xilinx/Vitis-AI.git
cd Vitis-AI/
git checkout tags/v1.4 -b v1.4
```



3. Multiscaler kernel sources are part of VVAS source tree and are located at

```
<VVAS_SOURCES>/VVAS/ivas-accel-hw/multiscaler
```

Kernels may have different configurations for different application requirements. Hence it is recommended to build the Kernels from the application design workspace with the required Kernel configuration for that application. Each application design workspace provided with this VVAS release has the required Kernel configurations for that application. In case one wants to change the kernel configuration for some reason, do these changes in the configuration files mentioned below. Compilation of Kernels is initiated from the build process of the final design for the application. Hence kernel compilation steps are not covered separately here.

- Configuration of DPU

```
<VVAS_SOURCES>/VVAS/ivas-examples/Embedded/smart_model_select/dpu_conf.vh
```

- Configuration of Multiscaler

```
<VVAS_SOURCES>/VVAS/ivas-examples/Embedded/smart_model_select/v_multi_scaler_config.h
```

You may modify the kernel configuration as per your requirements in these files.

#### *Creating SD Card image*

Once platform is available and kernels are built, next step is to stitch the required hardware accelerators (kernels) into the platform and generate final SD Card image using Vitis Flow.

VVAS sources already has ready to build example Vitis workspace for smart\_model\_select Application. This workspace uses Vitis Flow that stitches kernels into the platform and generates final SD card image. Follow below mentioned steps to build the final image.

```
cd <VVAS_SOURCES>/VVAS/ivas-examples/Embedded/smart_model_select
make PLATFORM=<PLATFORM_PATH> DPU_TRD_PATH=<DPU_PATH>
HW_ACCEL_PATH=<MULTISCALER_PATH>

PLATFORM_PATH =
<VVAS_SOURCES>/VVAS/ivas-platforms/Embedded/zcu104_vcuDec_DP/platform_repo/xilinx_zcu104_vcuDec_
DPU_PATH = <VVAS_SOURCES>/Vitis-AI/dsa/DPU-TRD/
MULTISCALER_PATH = <VVAS_SOURCES>/VVAS/ivas-accel-hw
```

Once above build is done, final sdcard image is available at `./binary_container_1/sd_card.img` location.

#### *Build VVAS Plug-ins and Libraries*

VVAS Plugins and libraries are part of petalinux bsp and are built along with building platform. So no need to build again. Still if one wants to build these for some reason, follow the steps mentioned below,

#### *Setting Sysroot*

Sysroot is required to build the VVAS GStreamer plugins. Sysroot installer location depends on whether you are using pre-built binaries or you have built the platform from scratch.

If you have downloaded the pre-built binaries in folder, say `PREBUILT_BINARIES`, then you can find the Sysroot installer at

```
<PREBUILT_BINARIES>/vvas_smart_model_select_2021.1_zcu104/sdk.sh
```

If you have built the platform yourself, then Sysroot installer is available at

```
<VVAS_SOURCES>/VVAS/ivas-platforms/Embedded/zcu104_vcuDec_DP/platform_repo/tmp/sw_components/sdk
```

One need to install the sysroot. Create a folder, say **sysroot** in `VVAS_SOURCES`. Command for sysroot generation is

```
<path to sdk.sh>/sdk.sh -y -d VVAS_SOURCES/sysroot/
```

Now sysroot is installed. You are ready to build plugins and applications.

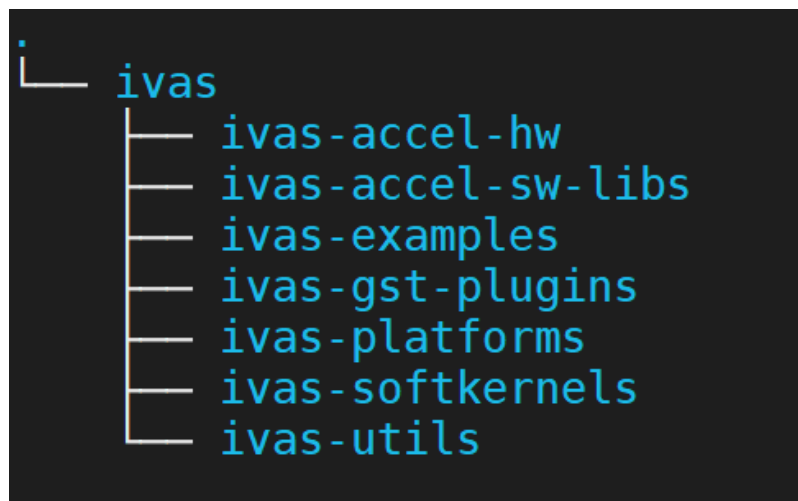
#### *Build Plugins and Libraries*

Get the VVAS Sources if not done already. Follow the steps mentioned below.

- Navigate to the folder, say VVAS\_SOURCES, where you want to clone the VVAS source tree.
- Clone VVAS repo

```
git clone https://github.com/Xilinx/VVAS.git
```

VVAS Source tree structure is described below:



- **ivas-utils:** This folder contains the source code for the VVAS kernel interface to be exposed by the acceleration software libraries, abstraction APIs on top of the Xilinx Runtime (XRT) tool, and common headers (for example, metadata).
  - **ivas-gst-plugins:** This folder contains all the VVAS GStreamer plug-ins, including the infrastructure plug-ins and the Custom plug-ins.
  - **ivas-accel-sw-libs:** Contains source for the acceleration software libraries that are called from the GStreamer infrastructure plug-ins.
  - **ivas-accel-hw:** This folder contains the hardware kernel source code.
  - **ivas-examples:** This repository hosts example solutions.
  - **ivas-platforms:** This folder contains the reference platforms for different applications.
- Navigate to VVAS\_SOURCES/VVAS folder
  - Unset the LD library path environment variable.

```
unset LD_LIBRARY_PATH
```

- Set the sysroot path

```
source VVAS_SOURCES/sysroot/environment-setup-cortexa72-cortexa53-xilinx-linux
```

You must have write permission to the sysroot.

- Build VVAS plugins and libraries

```
./build-ivas-essential.sh Edge
```

- When above step is complete, VVAS plugins are generated as **install/ivas\_installer.tar.gz**.
- Copy `install/ivas_installer.tar.gz` onto the root folder (/) of the target board and untar it. VVAS Plugins are now installed on the target device.

#### Build Smart Model Select Application

The example application (`smart_model_select`) is available in the “ivas-example” section of VVAS repository. Follow below steps to compile the application.

```
cd <VVAS_SOURCES>/VVAS/ivas-examples/Embedded/smart_model_select/src
unset LD_LIBRARY_PATH
source <sysroot path>/environment-setup-cortexa72-cortexa53-xilinx-linux
make SYSROOT= <sysroot path>/sysroots/cortexa72-cortexa53-xilinx-linux
```

## 1.5 VVAS GStreamer Plug-ins for Embedded Platforms

The following section describes GStreamer plugins for Embedded platforms. The plug-ins source code is available in the `ivas-gst-plugins` repository/folder of the VVAS sources tree. The following table lists the GStreamer plug-ins.

Table 1: GStreamer Plug-ins

Plug-in Name	Functionality
<code>omxh264dec/omxh265dec</code>	GStreamer plug-in to perform hardware accelerated h264/h265 video decoding.
<code>omxh264enc/omxh265enc</code>	GStreamer plug-in to perform hardware accelerated h264/h265 video encoding.
<code>ivas_xroigen</code>	Plug-in to generate region of interest metadata.

### 1.5.1 OMX Encoder/Decoder Plug-in

“`omxh264decoder/omxh265decoder`” are GStreamer Plugins for hardware accelerated H264/H265 Video decoding using VCU IP. Similarly “`omxh264encoder/omxh265encoder`” are GStreamer Plugins for hardware accelerated H264/H265 Video encoding using VCU IP. For more details about these plugins, refer to the [VCU PG252](#) Chapter 23: Encoder and Decoder Software Features.

### 1.5.2 Region of Interest Plug-in

The region of interest (ROI) GStreamer `ivas_xroigen` plug-in generates `GstVideoRegionOfInterestMeta` metadata information, which is expected by GStreamer OMX encoder plug-ins to encode raw frames with the desired quality parameters (QP) values/ level for specified ROIs. The `ivas_xroigen` plug-in prepares the `GstVideoRegionOfInterestMeta` metadata by parsing an IVAS supported list of metadata objects (`GstInferenceMeta`).

For implementation details, refer to [ivas\\_xroigen source code](#).

**Tip** This plug-in is only useful with embedded platforms because its main job is to generate metadata required for the GStreamer OMX encoder plug-in that exists on embedded platforms.

GStreamer ROI metadata information is attached with each outgoing buffer based on the following plug-in properties.

#### Input and Output

Accepts buffers with any of the GStreamer supported video formats on input and output `GstPads`.

### Control Parameters and Plug-in Properties

The following table lists the GObject properties exposed by ivas\_xroigen plug-in.

Table 4: ivas\_xroigen Plug-in Properties

Property Name	Type	Range	Default	Description
class-filters	GstValueArray	None	None	Array of desired inference classes. ivas_xroigen prepares ROI metadata only for class-filters array of strings out of classes present in IVAS supported metadata objects. Default behavior: Plug-in allows all classes when this property is not set. e.g., class-filters =<"person", "car">
insert-roi-sei	Boolean	true, false	false	When 'true', it sends custom events to OMX encoder to insert ROI information as SEI packet.
resolution-range	GstValueArray	None	None	Array of integers that represent the minimum and maximum range of ROI resolutions. Plug-in uses this resolution range to filter out ROIs. Set this property: <min-width, min-height, max-width, max-height>) Default behavior: Plug-in allows all ROIs. e.g., resolution-range=<0,0,320,240>
roi-max-num	Unsigned integer	0 to 4294967295	4294967295	Maximum number of ROI metadata to be attached per buffer
roi-qp-delta	Integer	-32 to 31	0	QP delta to be used for ROI in encoder
roi-qp-level	Enum	0, 1, 2, 3, 4	0	QP level to be used for ROI in encoder 0: high (delta QP of -5) 1: medium (delta QP of 0) 2: low (delta QP of +5) 3: don't-care (maximum delta QP value) 4: intra (region all LCU encoded with intra prediction mode)

roi-type	Enum	0, 1, 2	0	Type to be used to generate ROI metadata 0: default (ROI without encoder QP information) 1: qp_level (ROI encoder QP level) 2: qp_delta (ROI encoder QP delta)
----------	------	---------	---	--

### Example Pipelines

The following pipeline takes input video in an MP4 container from the filesrc plug-in. The qtdemux extracts the H.264 elementary stream from the MP4 container and pass it to the omxh264dec plug-in for decoding. The output of the decoder goes to the ivas\_xmultisrc plug-in for resizing to a resolution of 640 x 360 and color-space conversion to BGR format. If your design has other kernels, like multiscaler, for re-size and colorspace conversion, then you may use it along with ivas\_xabrscler plug-in. The output from ivas\_xmultisrc goes to the ivas\_xfilter plug-in for object detection using the densebox model. The inference operation generates the metadata and bounding box, for each detected object. The ivas\_xroigen plug-in creates the ROI metadata from the incoming bounding box information and passes this metadata to omxh264enc for ROI based encoding. This is an example pipeline to demonstrate ROI feature. This pipeline uses omxh264enc hence your design must have VCU encoder.

```
gst-launch-1.0 -v filesrc location=<filename>.mp4 \
! qtdemux \
! h264parse \
! omxh264dec internal-entropy-buffers=3 \
! queue \
! ivas_xmultisrc kconfig="/home/root/jsons/kernel_resize_bgr.json"
name=m2m_0 \
! video/x-raw, width=640, height=360, format=BGR \
! queue \
! ivas_xfilter kernels-config="/home/root/jsons/
kernel_densebox_640_360.json" \
! queue \
! ivas_xroigen roi-type=1 roi-qp-delta=-10 roi-max-num=10 resolution-
range="<0,0,500,300>" \
! queue \
! omxh264enc prefetch-buffer=true qp-mode=1
! queue
! fakesink
```

## 1.6 Tutorials

This section covers tutorials that will explain step by step approach to build GStreamer pipelines for different usecases using VVAS framework. Each tutorial explains what is the purpose of each plugin in the pipeline and how to configure, set properties and the corresponding json files wherever applicable.

### 1.6.1 Multi Channel ML

This tutorial explains how to build Machine Learning pipelines using VVAS. For more details, refer to the link [MultiChannelML](#).

## 1.7 Supported Platforms And Applications

For more information, contact marketing.

## 1.8 VVAS GStreamer Plug-ins for Data Center Platforms

For more information, contact marketing team.

## 1.9 Frequently Asked Questions

Yes.

VVAS Release has been covered by below mentioned licenses:

- Apache License
- Version 2.0
- 3-Clause BSD License
- GNU GENERAL PUBLIC LICENSE
- The MIT License

VVAS is tested on PetaLinux for embedded platforms. For more information about supported platforms, refer to [Platforms And Applications](#).

Below mentioned 16 models are supported.

- resnet50
- resnet18
- mobilenet\_v2
- inception\_v1
- ssd\_adas\_pruned\_0\_95
- ssd\_traffic\_pruned\_0\_9
- ssd\_mobilenet\_v2
- ssd\_pedestrian\_pruned\_0\_97
- tiny\_yolov3\_vmss
- yolov3\_voc\_tf
- yolov3\_adas\_pruned\_0\_9
- refinedet\_pruned\_0\_96
- yolov2\_voc
- yolov2\_voc\_pruned\_0\_77
- densebox\_320\_320
- densebox\_640\_360

Does this mean models not supported by Vitis AI? If the model is not in DPU deployable format, then it first needs to be converted into DPU deployable state. For this refer to [Vitis AI 1.4 documentation](#).

This VVAS release supports Vitis AI V1.4.

No, it has dependencies on Vitis AI 1.4.

The model name to be used for inferencing has to be provided in the JSON file for dpuinfer. For more details, see [DPU Infer](#).

while a pipeline is running, the model details cannot change. To change the model's details, stop the running pipeline, and then update the JSON file. Re-start the pipeline.

- H.264, H.265 encoded video streams
- Raw video frames in NV12, BGR/RGB formats

Receiving RTSP stream is supported by an open source plugin.

Yes, VVAS supports simultaneous execution of multiple instances of plugins to realize multistream decode and ML operations.

Refer to [Acceleration s/w development guide](#).

No. Using a platform that supports the required hardware/software components for the video analytics applications, you can directly use VVAS to realize your video analytics application with several reference solutions. Refer [Platforms And Applications](#).

Yes. The [Section 1.5.2](#) that generates ROI data required for encoders.

Yes. The `ivas_xabrscler` plug-in controls the `multiscaler` kernel to generate up to 8 different resolutions for one input frame. This plugin, along with `resize`, can also do colorspace conversion.

No.

Yes. There are sample accelerated platforms and applications provided that you can execute by following a few steps. Start at [Platforms And Applications](#).

One can connect multiple instances of `ivas_xdpuinfer` one after another to implement multi-stage cascading network and each ML instance will generate its own inference data separately. This is already supported in this release. However accumulation of inference data from several ML instances in a pipeline into a single meta data structure is not yet supported by plug-ins and this has to be done by the application.

VVAS is based on GStreamer framework. It relies of debugging tools supported by GStreamer framework. For more details, you may refer to [GStreamer Debugging Tools](#).

Using GStreamer's native fps display mechanism.

Refer to [Vitis AI 1.4 documentation](#).

Refer to [Building VVAS Plugins and Libraries](#).

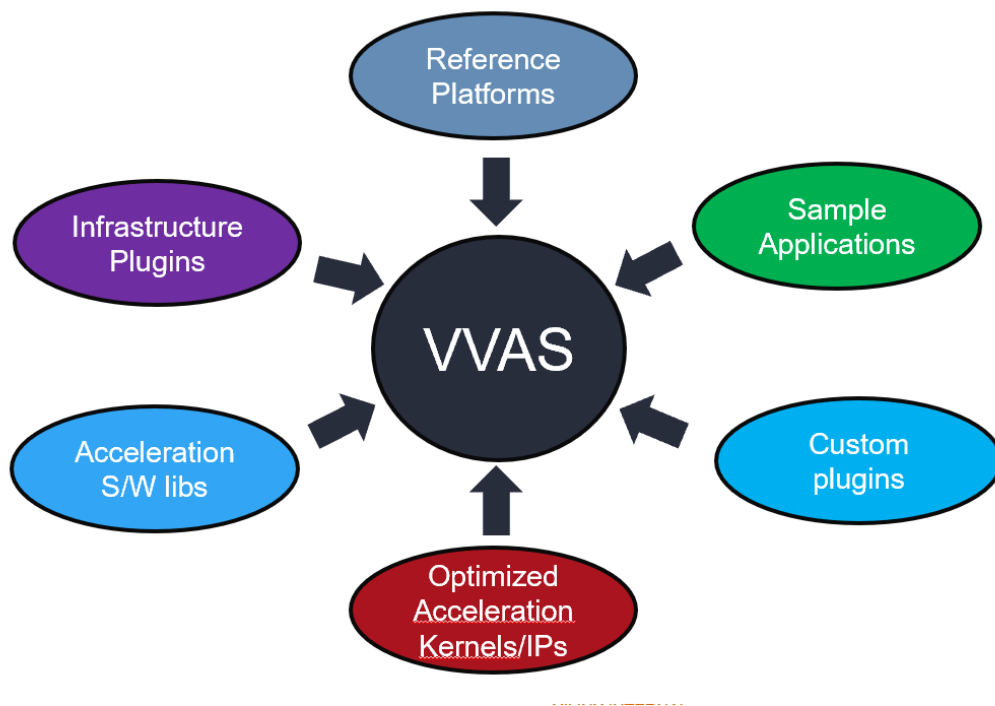
Contact support.

## 1.10 Why VVAS?

Application developers can build seamless streaming pipelines for AI-based video and image analytics, complex Adaptive Bitrate Transcoding pipelines and several other solutions using VVAS without having any understanding about FPGA or other development environment complexities. VVAS ships with several hardware accelerators for various functionalities, highly optimized GStreamer plugins meeting most of the requirements of the Video Analytics and transcoding solutions. For advanced developers, VVAS provides an easy to use framework to integrate their own hardware accelerators/kernels in to Gstreamer framework based applications. VVAS provide AI model support for popular object detection and classification models such as SSD, YOLO etc. All these infrastructure gives the flexibility for rapid prototyping to full production level solutions by significantly reducing time to market for the solutions on Xilinx platforms.

## 1.11 VVAS Core Components

This section gives more deeper insight into VVAS. VVAS comprises following core components.



### 1.11.1 Custom Plug-ins

These are highly optimized GStreamer plug-ins developed to provide very specific functionality using optimized Kernels and IPs on Xilinx Platform. Refer to [Section 1.2.1](#) for more details about how to use these plug-ins.

### 1.11.2 Infrastructure Plug-ins

These are generic infrastructure GStreamer plug-ins being developed to help users to directly use these plug-ins to integrate their Kernels into GStreamer framework. User need not have in-depth understanding of the GStreamer framework. Refer to [Section 1.2.2](#) for more details about how to use these plug-ins. These plug-ins are part of “ivas-gst-plugins” repository.

### 1.11.3 Acceleration S/W Libs

These are optimized Acceleration s/w libs developed to manage the state machine of the acceleration Kernels/IPs and expose the interface so that these Acceleration s/w libs can be hooked into VVAS Generic Infrastructure Plug-ins. These can be used as reference to develop a new Acceleration s/w lib based on VVAS framework. Details about the vvas acceleration s/w libs and how can these be used with infrastructure plug-ins are explained in [Section 1.2.2](#) section.

### 1.11.4 Acceleration H/W (Kernels/IPs)

These are highly optimized Kernels being developed by Xilinx. Details of these are captured in this section. Refer to VVAS Acceleration H/W [VVAS Acceleration H/W](#) for more details.

### 1.11.5 Reference Platforms And Applications

There are different requirements of different applications. VVAS provides several reference platforms catering to different applications/solutions needs. Embedded platforms and sample application details can be found in [Platforms And Applications](#) section.





