UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

Design of an image acquisition and processing system using configurable devices

A Master's Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Miquel López Muñoz

In partial fulfilment

of the requirements for the degree of

MASTER IN ELECTRONICS ENGINEERING

Advisor: Juan Manuel Moreno Arostegui

Barcelona, June 2021

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

telecos BCN

Title of the thesis: **Design of an image acquisition and processing system using configurable devices**

Author: **Miquel López Muñoz**

Advisor: **Juan Manuel Moreno Aróstegui**

Abstract

This thesis consists of the evaluation of the possibility to implement a Neural Network in an FPGA instead on the more used GPU. Theoretically, an FPGA is a better choice in terms of processing power, latency, or flexibility but its configuration is harder.

In this report, the implementation process for an FPGA is followed, including the creation of an embedded Operating System, video capture and display pipelines, testing of the chosen model and the final implementation of the model in the board.

As a result of the evaluation, the conclusion is that nowadays the way to implement a neural network in an FPGA is not mature enough to compete with GPU alternative. The tools needed to achieve this implementation are very limited and the process is confusing. In the other hand, the GPU implementations has a huge catalogue of HW options and one can choose the better solution for its model.

Acknowledgements

This project would not have been possible without the help of my Idneo partners Rubén Veloso and Aitor Sanchez, as well as my supervisor Martí Cobos. With special thanks to Ruben for proposing me this topic and for helping me in the first steps of the project to understand and install all the required tools.

I must thank also my Idneo partners Joan Sintes and Biel Tura for their patience in explaining me the basis of Machine Learning from my absolute lack of knowledge in these field.

I must also thank Ignacio Espinel and Stefano Tabanelli from Avnet for providing me a large amount of Documentation, Trainings and project examples to manage the ZCU104 Development board and all the Xillinx development tools.

Finally, I have to thank my project supervisor Juan Manuel Moreno Aróstegui to help me to understand the complexity of the project and advise me in the steps of the development project and its structure. Also, to the project follow up which it has been exactly the kind of tracing that I need.

Revision history and approval record

Revision	Date	Purpose
0	13/05/2021	Document creation
1	30/08/2021	Document corrected
2	03/10/2021	Document revised
3	05/10/2021	Document approved

Written by:		Reviewed and approved by:	
Date	13/05/2021	Date	05/10/2021
Name	Miquel López	Name	Juan Manuel Moreno
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables

1. Introduction

The main objective of this project is to explore the possibility to implement a Convolutional Neural Network in an FPGA instead of in the most used solution GPU, testing the availability and the difficulty of the process and apprise if it could be a better solution.

This project is realized in collaboration between ETSETB-UPC and Idneo Technologies Vision team. Vision is an engineering team that is focused on computer video applications, developing from the automotive cameras until Machine Learning algorithms to process the captured video. In this context, it is reasonable to explore the possibility to get a more efficient way to implement the Convolutional Neural Network that applies the algorithm. However, since this new implementation is so recent and still under development and because there is nobody in the company with an expertise regarding FPGA, there is no know-how about the needed knowledge of this project. This has caused that the objectives of the project and its process are splitted into steps.

The main scope of the project is to develop an image capture system embedded in a development board with an embedded Linux-based OS and process this acquired video with a Deep Learning algorithm. Since the main field of work of Vision team is the automotive cameras, it has been decided to implement a Deep Learning algorithm that can be used in autonomous driving such as Semantic Segmentation of a video captured with a car front camera.

The first implementation is based in an embedded Linux-based OS in a development board that captures video with a USB camera and stores a video file that is then processed with a CNN in the host PC. The second and final implementation consists on implement this CNN in the FPGA of the development board and write a C++ application running in the board that captures the video, process each frame in the CNN and shows the resulting video in an external monitor.

To do so, numerous Xilinx trainings and project examples which are mentioned in the Bibliography, have been followed. As mentioned, this kind of implementation is under development so it is not as flexible as they should be, and this has been caused some troubles and delays.

Figure 1-1 shows the original Gantt diagram of the project.

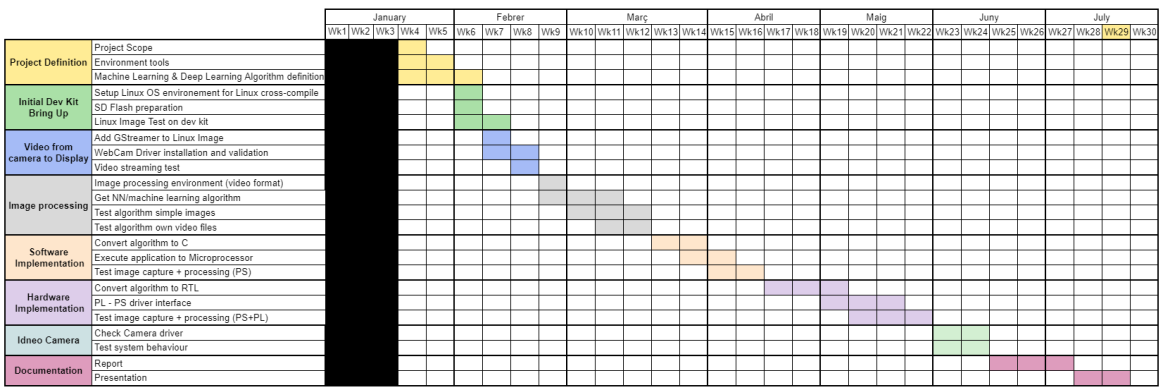


Figure 1-1: Gantt diagram of the project.

At first, the project is intended to be finished in one semester, but due to the complexity of each step and the lack of know-how regarding every configuration to be performed, remarkable delays have been added. Despite this, the project could be finished in time, but as already mentioned, this kind of implementation is under development and this implies that the process works fine in the trainings and examples but when trying something different, unexpected errors occur which do not have a known solution. These issues caused that the project could not be finished in a single semester and an additional one has been needed to realize some workarounds.

The chapters that this thesis document contains are briefly described below:

State of the art of the technology used or applied in this thesis

In this step, the context of the field of study regarding Machine Learning and the most used implementation are introduced. Also, the alternative of implement the NN with FPGA and the set of tools used to explore this possibility are explained.

Project development

This is the core chapter of the thesis. In this chapter there is first a description of the architecture of the system. Then, each step of the project development is described. These steps are: Embedded PetaLinux Image; Capture, display and store video image from USB camera; Processing image in host PC using a Convolutional Neuronal Network; Implementing Convolutional Neuronal Network in embedded system.

Results

The results of the study are collected in this chapter, this includes screen captures and reports. To represent a video in this report, it has been decided to make a collage with a set of the frames of the video.

Budget

In this chapter, a summary of the required budget is included in a table.

Conclusions and future development

The conclusions of the project are described in this chapter including a brief discussion of the availability to implement the NN in an FPGA instead of a GPU. Also, some future developments are commented to motivate future thesis or research projects.

At the end of the project, there is detailed the literature consulted in this thesis, as well as the project examples and trainings followed in the Bibliography chapter. And finally, there are a set of Appendixes and a section defining the Acronyms and Glossary appearing in the report.

2. State of the art of the technology used or applied in this thesis

Machine Learning Algorithms are mathematical algorithms that, with the help of a training process, learn how to perform a specific task. Nowadays, this field of study is growing and every day is used more to develop applications. These algorithms have the advantage in front of the traditional algorithms that the developer does not need the way to reach the solution. In this project, Convolutional Neural Network machine learning algorithms are used. This kind of algorithms are a multi-layer bidimensional matrix algorithm used in the Deep Learning algorithms specifically to process images.

Most of the video processing applications realized nowadays are implemented in a Graphics Processing Unit (GPU) to free the CPU from the large number of calculations needed by the algorithm. The two main suppliers of Deep Learning platforms are NVIDIA and AMD. They have development boards to use one or more GPUs along with the CPU, to implement neural networks in the board or in the cloud, training all frame kind of models with all the available frameworks, etc…

Since the implementation in the GPU is so advanced, most of the main frameworks like TesnorFlow, Caffe, PyTorch and so on, are intended to be used with Python. Also, the computer vision libraries like OpenCV or CUDA works better with Python. Even if they have C++ version, they are not working as good as they do in Python.

In this field lead by Graphics Processing Unit, it arises the possibility to implement the Neural Network in a FPGA instead. Since both solutions fulfil the main objective which is free the CPU to the large amount of processing, the idea to use a FPGA instead of GPU is quite interesting. Since the FPGA is programable HW, it has some advantages in front of a GPU that could make them a better solution to implement the Neural Network. FPGA can process more operations per second, is more flexible since the HW can be changed by programming so it can be adapted better to each application, the latency is lower because it is not instruction-based execution but parallel HW execution and the connection between components and peripherals can be more efficient since is programmed by the developer. The drawback is that configuring an FPGA is much more difficult than a GPU. In addition, the GPU implementation is so stablished that it is easier to find a suitable GPU for your certain application.

This is the reason why nowadays almost every developer is using a GPU instead of the theoretically better solution FPGA. Nevertheless, Xilinx is trying to launch an easy way to implement the NN in an FPGA and perform a Deep Learning Video Processing application, but it is still under development, and it is not as robust and as flexible as advertised. There are a lot of incompatibilities between software versions, and a lot of tools to each task and it is not clear what can be done and what is not compatible.

The following tools are used in this project. The version of each tool has to be the specified or it would be incompatible with the rest:

- Host Operating System: Ubuntu 16.04 LTS
- FPGA development software: Vivado Design Suite v2018.3

- Embedded OS: PetaLinux v2018.3
- Software to compile NN model to be used by the FPGA IP: DNNDK 3.0
- FPGA IP to implement model: DPU IP v2.0
- Software environment to compile Application: Xilinx SDK (XSDK) 2018.3

And the Hardware used in the project is the following:

- Host PC: Lenovo ThinkPad L560
- Development board: Zynq UltraSCALE+MPSoC ZCU104
- USB Camera: See3CAM_CU30_CHL_TC_BX e-con Systems
- SD card: SanDisk Ultra 16GB
- DP Monitor: BenQ GW2780 27 LED IPS Eye Care

3. Project development

The development of the project is divided into different steps in order to easily focus each part of the whole system and make it work itself as a smaller system. This way, at the end of the project, the whole system will be merged using the acquired know-how from the previous steps. This working methodology has been selected because of the complexity of the work field and the lack of knowledge about it from the company.

Accordingly, this section is composed by first a description of the final system and its architecture and then all the steps required to complete the work.

3.2. Embedded PetaLinux Image,

3.3. Capture, display and store video image from USB camera

3.4. Processing image in host PC using a Convolutional Neuronal Network

3.5. Implementing Convolutional Neuronal Network in embedded system

3.1. System architecture and description

3.1.1. System architecture

The architecture of the final system is described in *Figure 3-1* below:

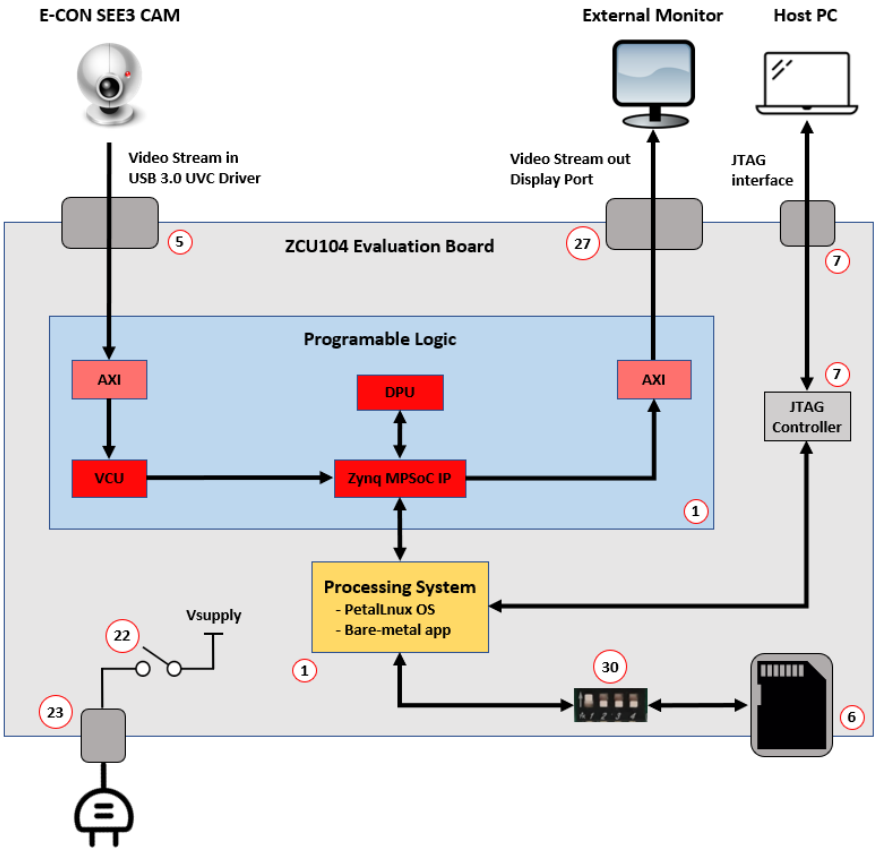


Figure 3-1: System architecture draft.

The reference numbers in *Figure 3-1* are the same than in *Figure 3-3* to easily relate these components to their descriptions in section below.

In Host PC, PetaLinux Embedded Image is configured and build and properly mounted in a boot SD card. This SD card is introduced in its ZCU104 Board slot (6) and SW6 (30) is set to realize the system boot from the SD card.

After powering the development board trough Power connector (23), being power switch (22) in ON position, the Processing system of the Zynq UltraScale+ MPSoC (1) boots from SD card and launches de PetaLinux OS and the bare-metal application.

System can be monitoring by the host PC through JTAG connection using a software such as Minicom. This way we can introduce the user and password of the OS in the host PC terminal.

Using the Linux drivers and the open-source multimedia framework GStreamer, a Video pipeline for the video stream can be defined. The first half of his pipeline starts from the USB Camera sending the video trough USB Video Class driver, this stream is received in the Programable Logic and either routed to the VCU to be encoded and stored as a file or passed through the VCU to the PS.

The C++ code running in the PS sends each frame to the DPU in the PL to be processed by Segnet CNN. The processed frame returns to the PS and it is sent to the second half of the video pipeline, in which the processed video is sent to an external monitor via Display Port.

3.1.2. HW Components description

Apart of the host PC which is Lenovo Think Pad L560 with Ubuntu 16.04 LTS OS and an external monitor with Display Port input, there are two basic HW components needed in the project:

1. USB Camera: See3CAM_CU30_CHL_TC_BX e-con Systems [11] (picture in *Figure 3-2*)

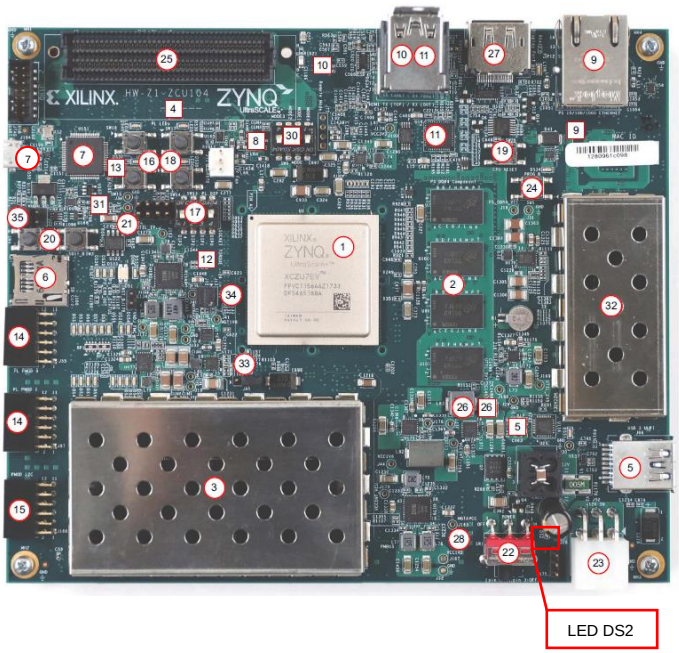
Figure 3-2: See3CAM CU30_CHL_TC_BX Cam. Source: [12]

- 1/3” On Semiconductor AR0330 CMOS + Image Signal Processor (ISP)
- 3.4 Mp, color. Uncompressed UYVY format and Compressed MJPEG format
- USB 3.1 Gen 1 Super Speed[1]
- USB type-C connector
- Operating Voltage: 5V ± 5%, Current: 433mA
- High compatible with Linux OS. Plug and Play V4L2 driver.

[1] This nomenclature is a little bit confusing. USB 3.1 Gen 1 is also known as USB 3.0 (up to 5 Gbit/s), may not be confused with USB 3.1 Gen 2 which allows data transfers up to 10 Gbit/s.

2. Development Board**:** Xilinx ZCU104 (picture in *Figure 3-3*)

This is a very complete board with a large number of capabilities intended to be used in a lot of different applications. Only the most relevant for this project are listed below.



1. Zynq UltraScale+ XCZU7EV MPSoC
5. USB 3.0 Transceiver and USB 2.0 ULPI PHY
6. SD Card Interface connector
7. Programmable Logic JTAG
22. SW1: power on/off
23. Power Connector
27. Display Port connector
30. SW6: U1 MPSoC PS bank 503 4-pole mode DIP switch

Figure 3-3: ZCU104 development board main components for this project. Source: [7]

1 The ZCU104 board is populated with the Zynq UltraScale+ XCZU7EV-2FFVC1156 MPSoC, which combines a powerful processing system (PS) and programmable logic (PL) in the same device. The PS in a Zynq UltraScale+ MPSoC features the Arm® flagship Cortex®-A53 64-bit quad-core processor and Cortex-R5 dual-core real-time processor [7].

PS

The Zynq UltraScale+ MPSoC PS block has three major processing units [7]:

- Cortex-A53 application processing unit (APU)-Arm v8 architecture-based 64-bit quad core multiprocessing CPU.
- Cortex-R5 real-time processing unit (RPU)-Arm v7 architecture-based 32-bit dual real-time processing unit with dedicated tightly coupled memory (TCM).
- Mali-400 graphics processing unit (GPU)-graphics processing unit with pixel and geometry processor and 64 KB L2 cache.

PL

The Xilinx® UltraScale™ architecture in the programmable logic (PL) provides an extensive set of functions and resources. The Zynq® MPSoC devices include several peripherals controllers and functional units [8]. The relevant ones for this project are:

- DisplayPort Video and Audio Interfaces

- USB Interfaces
- Video Codec Unit (VCU). It provides multi-standard video encoding and decoding, including support for the high-efficiency video coding (HEVC) H.265 and advanced video coding (AVC) H.264 standards. The main features are:
    - H.264 and H.265 standards encoding/decoding.
    - Up to eight simultaneous streams.
    - 8K x 4K at a reduced frame rate.
    - Progressive video only (no interlace support).
    - I, IP, and IPB encoding/decoding.
    - 8-bit and 10-bit color depth, YCbCr 4:2:2 and 4:2:0 video formats, and up to a 4K x 2K@60/8K x 2K@15 Hz rate.
- DPU. The Xilinx® Deep Learning Processor Unit (DPU) is a programmable engine dedicated for convolutional neural network. The unit contains a register configure module, a data controller module, and a convolution computing module. There is a specialized instruction set for DPU, which enables DPU to work efficiently for many convolutional neural networks [12].

USB Controller

The USB 3.0 controller consists of two independent dual-role device (DRD) controllers. Both can be individually configured to work as host or device at any given time. The USB 3.0 DRD controller provides an eXtensible host controller interface (xHCI) to the system software through the advanced eXtensible interface (AXI) slave interface. An internal DMA engine is present in the controller, and it utilizes the AXI master interface to transfer data. The three dual-port RAM configurations implement the RX data FIFO, TX data FIFO, and descriptor/register cache. The AXI master port and the protocol layers access the different RAMs through the buffer management unit [8]. Its most important features are:

- Two USB 2.0/3.0 controllers.
- Supports a 5.0 Gb/s data rate.
- 64-bit AXI master port with built-in DMA.
- Supports 12 endpoints (six out and six in).

Display Port Controller

The DisplayPort controller can source data from memory (non-live input) or the (live input) programmable logic (PL). The DisplayPort processes data and sends it out through the DisplayPort source-only controller block to external display devices or to the PL (live output). The DisplayPort pipeline consists of the DisplayPort direct memory access (DMA) for fetching data from memory, a centralized buffer manager, a display rendering block, an audio mixer block, and the DisplayPort source controller, along with the PS-GTR block, which contains the multi-gigabit transceivers that provide high-speed communication link between the media access controllers of the peripherals and their link partners outside the board. The DisplayPort pipeline supports an ultra-high definition (UHD) aggregate video bandwidth of 30 Hz [8].

It provides support for the following video formats:

- Resolution up to 4K x 2K at 30Fps.
- Y-only, YCbCr444, YCbCr422, YCbCr420, and RGB video formats.
- 6, 8, 10, or 12 bits per color components.
- Progressive video.
- A 36-bit native video input interface to capture live video.
- Non-live video from frame buffers using local DPDMA.

5 The ZCU104 board uses a Standard Microsystems Corporation USB3320 USB 2.0 ULPI transceiver at U116 to support a USB connection to the host computer (see *Figure 3-4*). In this project, the USB 3.0 (USB 3.1 gen 1) connection to Zynq UltraScale USB GTR is used [7].

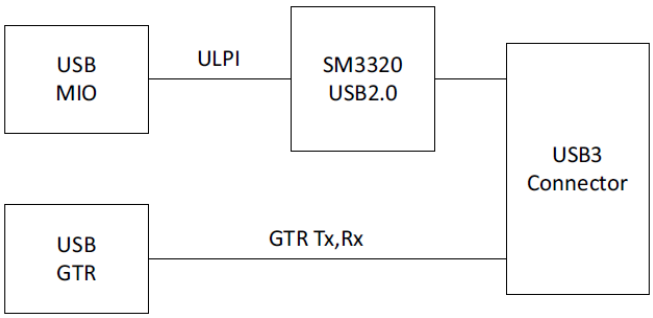


Figure 3-4: USB interface. Source: [7]

6 The ZCU104 board includes a secure digital input/output (SDIO) interface to provide access to general purpose non-volatile SDIO memory cards and peripherals. The ZCU104 SD card interface supports the SD1_LS configuration boot mode documented in the *Zynq UltraScale+ MPSoC Technical Reference Manual (UG1085)* [8][7].

7 The ZCU104 board JTAG chain is shown in *Figure 3-5.*

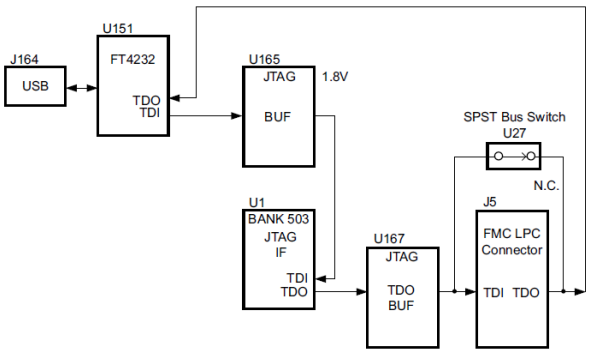


Figure 3-5: JTAG Chain Block Diagram. Source: [7]

22 23 The ZCU104 board power switch is SW1. Sliding the switch actuator from the off to the on position applies 12V power from J52, a 6-pin mini-fit connector. Green LED DS2 illuminates when the ZCU104 board power is on [7].

27 The Zynq UltraScale+ MPSoC provides a VESA DisplayPort 1.2 source-only controller that supports up to two lanes of main link data at rates of 1.62 Gb/s, 2.70 Gb/s, or 5.40 Gb/s. The DisplayPort standard defines an auxiliary channel that uses LVDS signaling at a 1 Mb/s data rate. [7]

30 Configuration switch with 4-bit position. Boot mode is selected setting the 4 stitches as shown in table *Table 3-1*.

Boot Mode	Mode Pins [3:0]
JTAG	0x0 - 0000
QSPI32	0x2 - 0010 (default)
SD1	0xE - 1110

Table 3-1: Switch Configuration. Source: [7]

3.2. Embedded PetaLinux Image

As shown in *3.1. System architecture and* description, Zynq UltraSCALE + MPSoC ZCU104 board is intended to be driven by an embedded OS called PetaLinux. So, the first step is focused in understanding and mounting an embedded OS image in an SD card and boot the ARM Cortex-A53 from it. The OS used in this step is composed with the minimal features and applications for simplicity. The chosen embedded Operating System is PetaLinux, it is the one recommended by Xilinx® due to its compatibility with the HW and the rest of the tools.

PetaLinux is an Embedded Linux System Development Kit targeting Xilinx® FPGA-based System-on-Chip designs [2]. It uses Yocto Project, which allows the developer to customize the embedded image and adapt it to a certain HW architecture.

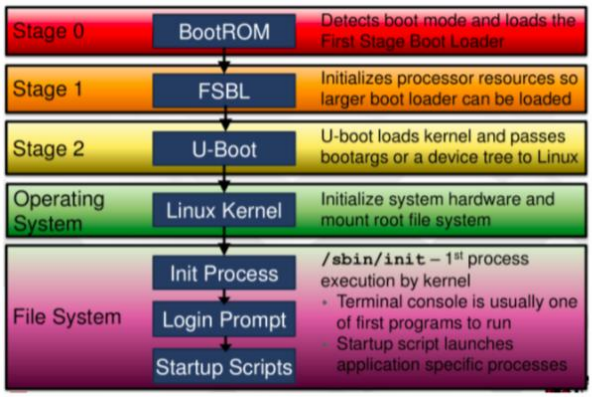
PetaLinux contains [2]:

- Yocto Extensible SDK (e-SDK): allows the user to add new libraries and apps in the project. E-SDK consists of all the layers for the architecture (core, meta-oe…), sstate-cache that allows incremental project builds and the sysroots of the embedded OS.
- Minimal downloads: use of premirrors to download the needed source files.
- XSCT and tool chains: The PetaLinux tool uses XSCT underneath for all embedded SW apps. Linux tool chain for all architectures is from Yocto.
- PetaLinux CLI tools: This contains all the required PetaLinux commands.

The followed steps to successful build a PetaLinux Image to be used in Zynq UltraSCALE + MPSoC ZCU104 board are described below. However, to understand these steps it is necessary to know the structure of the Embedded OS Image and its required files, as well as the boot process of a Linux OS.

The basic boot process of a Linux OS starts with the named Stage 0. In this Stage, the ROM code detects boot mode and loads the executable code of First Stage Boot Loader (FSBL) from selected interface, in this case the SD card. Then, in Stage 1, the FSBL initializes an external memory and system clocks allowing to load a larger bootloader file, in this case U-Boot. In Stage 2 the U-Boot file loads the Linux Kernel and passes him the device tree. Finally, the Kernel initializes the Hardware and mounts the root file system. [25]

Figure 3-6 shows a summary of the basic Linux boot process



Boot Stage	Operations	User Configurable	Zynq-7000 Terminology
Stage 0	ROM code detects desired boot mode (NAND, QSPI, JTAG) and loads executable code of First Stage Boot Loader (FSBL) from selected peripheral/interface	No	BootROM
Stage 1	Initializes external memory and system clocks so that a larger, more advanced boot loader (in our case U-boot) can be loaded	Yes	FSBL
Stage 2	Used to locate, load, and execute the Linux kernel and is also responsible for passing a device tree to the kernel	Yes	U-Boot

Figure 3-6: Embedded Linux boot process. Source: [25]

To go through these stages, there are some files that needs to be generated and stored in a certain mode in the SD card. These files are:

- PL Bitstream: Stored in *boot.bin* container. Definition of the PL (Programable Logic) exported by Vivado in the *wrapper_file.hdf*. After adding it to PetaLinux project, the file is automatically stored with the name: *system.bit*
- FSBL: Stored in *boot.bin* container. Generated by building PetaLinux project with name: *zynq_fsbl.elf*.
- PMU (Platform Management Unit) Firmware: Stored in *boot.bin* container. This is an optional file only for Zynq UltraScale+ MPSoC that controls the power-up, reset, and monitoring of resources within the system. Generated by building PetaLinux project with name: *pmufw.elf*.
- ATF (Arm Trusted Firmware): Stored in *boot.bin* container. It provides a reference to secure software for ARMv8-A architecture, and it provides implementations of various interface standards and Secure monitor code for interfacing to Normal world software. This firmware is for Zynq UltraScale+ MPSoC only. Generated by building PetaLinux project with name: *bl31.elf*.
- U-Boot: Stored in *boot.bin* container. Generated by building PetaLinux project with name: *u-boot.elf*.
- Kernel: Stored in *Image.ub.* Generated by building PetaLinux project.
- Root file System: Generated by building PetaLinux project with name: *rootfd.tar.gz*.
- Device tree: Stored in *Image.ub.* Generated by building PetaLinux project.

Some of these files are stored in a bigger file called *boot.bin* container. After the PetaLinux project build, when all the required files have been generated, this container is mounted. This is the last step of the embedded PetaLinux Image.

Figure 3-7 Shows the structure of the *boot.bin* container.

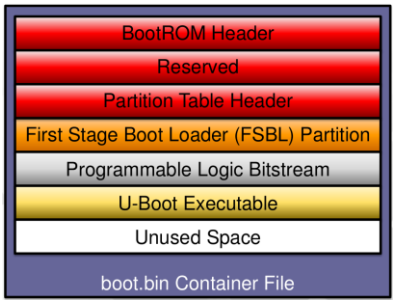


Figure 3-7: Boot.bin container file diagram. Source: [25]

Once it is known the structure of a PetaLinux Image, the required steps to create, configure and build a PetaLinux project can be detailed.

In *Figure 3-8* a summary of the steps to go through and the flow of the generated files is shown:

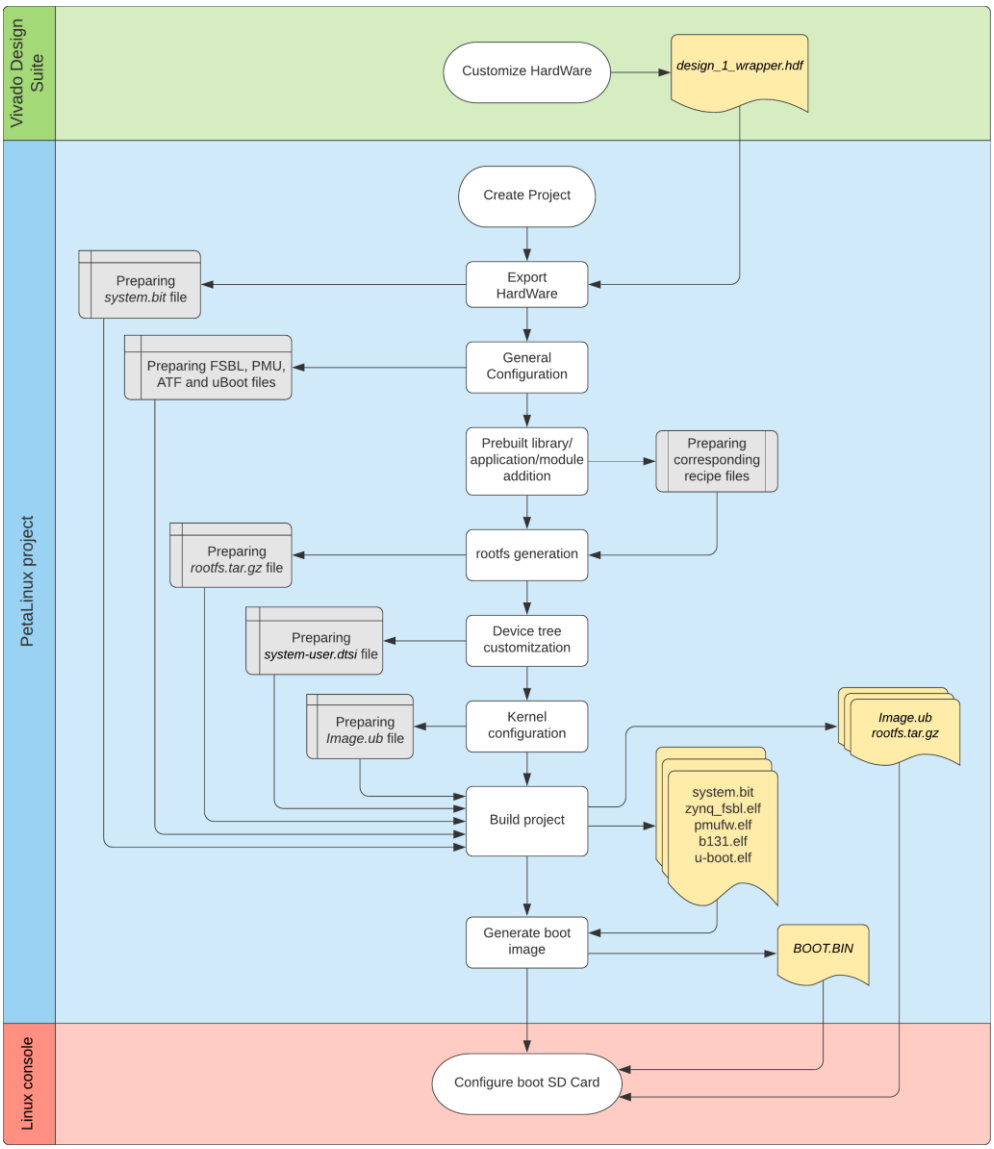


Figure 3-8: PetaLinux Embedded Image project steps diagram.

In this project step, the simplest PetaLinux image has been generated because the objective at this point is to learn the basic needs of the embedded PetaLinux and how to create the image, properly configure the SD boot and be able to have the OS running in the board. Therefore, some of the previous steps are not followed in this project step, the scheme above is the full project process. Specifically in this step there is no custom library, application or module addition, and default kernel and device tree are used.

3.2.1. Basic Programable Logic

As already mentioned, this project step is kept as simple as possible. No peripherals are used, or PL functionality is required. Only the processing system IP (Zynq UltraScale+ MPSoC) is added to Vivado project as shown in *Figure 3-9*.

Figure 3-9: Basic Programable Logic Vivado Block Design.

After creating an HDL wrapper, compile, build and generate Bitstream, it is necessary to export HW to use it in the next step.

3.2.2. Create PetaLinux project, export HW and set general configuration

Once the programmable logic is defined and exported, the PetaLinux project can be created. Due to the difficulties at download and install all the tools with the certain versions it is highly recommended to use a virtual environment. In this project, the default Python virtual environment for Linux has been used.

Before start using PetaLinux, the file *settings.sh* from the PetaLinux installation file needs to be sourced to Ubuntu using this command line:

```
$: source <path to petalinux .sh file>/settings64.sh
```

Then the system is able to use PetaLinux commands. Then, the easiest way is to create a project from a template using the next command (parameters explained in *Table 3-2*):

```
$: petalinux-create --type project --template zynqMP --force –name
<project name>
```

Option	Functional Description	Value Range	Default Value
`-t,--type TYPE`	Specify the TYPE of object to create. This is required.	• project • apps • modules	None
`-n,--name NAME`	Create object with the specified NAME. This is optional when creating a project from a BSP source. Otherwise, this is required.	User-specified	None
`-p,--project PROJECT`	PetaLinux project directory path. This is optional.	User-specified	Current Directory
`--force`	Overwrite existing files on disk. This is optional.	None	None
`-h,--help`	Display usage information. This is optional.	None	None

Table 3-2: petalinux-create Command Line Options. Source: [14]

A new folder with the name “project name” will be created. After entering this folder, we can export the HW writing the following command:

```
$: petalinux-config --get-hw-description=<path to .hdf directory in Vivado project folder>
```

A new window with the HW configurations will open and the following parameters must be specified (see APPENDIX A):

- `DTG Settings → (zcu104-revc) MACHINE_NAME`
- `u-boot Configuration → (xilinx_zynqmp_zcu104_revC_defconfig) u-boot config target`
- `Image Packaging Configuration → Root file System type → SD card`
- `Yocto Settings → (zcu104-zynqmp) YOCTO_MACHINE_NAME`

In this way, we are configuring properly the HW configurations with our development board and telling PetaLinux that rootfs will be stored in a SD card. When pressing the *ok* button, the window automatically closes itself and PetaLinux may configure all and generate the *system.bit* file to be later included in the BOOT.BIN container. Also, FSBL, ATF, PMU and uboot files are prepared to be built. These files are configured by default, so no special configuration needs to be specified apart of u-boot configuration.

3.2.3. Rootfs generation

As already mentioned, in this first PetaLinux image there is no custom application or module, so directly we can configure rootfs using this command:

```
$: petalinux-config -c rootfs
```

-c term indicates that the configuration to be open refers to a component. A component can be rootfs, kernel, bootloader, device-tree, etc.

A new window will open to customize the rootfs of the PetaLinux image (see APPENDIX A). At this point, we don't need any specific library or application because it is only a test to set-up the PetaLinux image, but to display the default GUI in a monitor we activate the following:

- `filesystem Packages → misc → python3 → all`
- `filesystem Packages → misc → python3-async`
- `filesystem Packages → misc → python3-git`
- `filesystem Packages → misc → python3-gitdb`
- `filesystem Packages → misc → python3-setuptools`
- `filesystem Packages → misc → python3-smmap`
- `filesystem Packages → x11 → base → libdrm → libdrm`
- `filesystem Packages → x11 → base → libdrm → libdrm-tests`
- `filesystem Packages → x11 → base → libdrm → libdrm-kms`
- `Petalinux Package Groups → packagegroup-petalinux-matchbox`
- `Petalinux Package Groups → packagegroup-petalinux-x11`

After pressing *ok* in the window, PetaLinux prepares rootfs to be built.

3.2.4. Build project and generate Boot Image

When all the files are ready and well configured, the following command launches the build of the whole project:

```
$: petalinux-build
```

This will take a significant amount of time, the larger the project, the slowest the building.

Once the building finishes, the generated files will appear in their corresponding folders inside the project folder. These files are: *system.bit, zynq_fsbl.elf, pmufw.elf, b131.els, u-boot.els, image.ub* and *rootfs.tar.gz.*

The first five ones are then combined to form the *boot.bin* container using the following command:

```
$: petalinux-package --boot --format BIN –fsbl
images/linux/zynqmp_fsbl.elf --u-boot images/linux/u-boot.elf --pmufw
images/linux/pmufw.elf –atf images/linux/bl31.elf --fpga
images/linux/system.bit –force
```

When the process finishes, *boot.bin* is also generated and ready to mount the boot SD card.

3.2.5. Configure boot SD card

To boot from SD card, it is necessary to properly configure it and store the files. In this project, 2 partitions are needed for the SD card:

- *boot* partition: FAT32 format. ≥ 60 MB. *boot.bin* and *image.ub* files.
- *root* partition: Ext4 format. ≥ 3 GB. *rootfs.tar.gz* file.

Files in the *boot* partition stores required files for booting the image and in *root* partition, the rootfs will be mounted.

Finally, in the zcu104 board, the SW6 should have a specific configuration indicated in *Table 3-1* of section 3.1.2. HW Components description in order to boot from SD card. This is 1110 (ON, ON, ON, OFF) as shown in *Figure 3-10*.

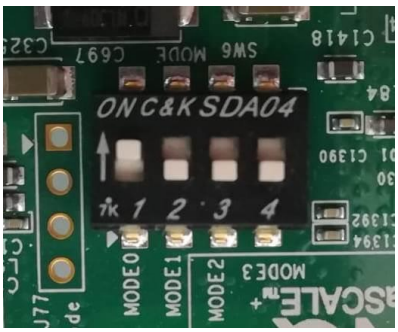


Figure 3-10: SW6 SD boot configuration.

3.3. Capture, display and store video image from USB camera

This second project step is focused on modifying the PetaLinux Embedded Image to be able to capture image from a USB camera, display a Video stream in an external monitor via Display Port, and store a video file to be processed in the host PC.

With this objective, it is necessary to develop a complete HW design in the FPGA to capture video, process it in the VCU and Display the video. Also, some changes in the PetaLinux project may be introduced to add more functionality than in the basic one. Lastly, one need to know how to define video pipelines from the host PC terminal.

Since the camera is a USB camera compatible with UVC driver, we can use the V4L2 driver to enumerate and manage the device. Also, the external monitor can be managed with the GStreamer video sink kmssink so both devices and the VCU can be connected and configured using the open-source multimedia framework GStreamer.

3.3.1. Programable logic

In the previous project step, the definition of the HW in the FPGA was only the Zynq UltraScale+ MPSoC IP. In this step, it is necessary to add some peripherals in order to manage the video pipeline. *Figure 3-11* shows a simplified diagram of the PL design, and each part is explained below. This project is based on rdf0428-zcu106-vcu-trd-2018-3 and *zcu102-dpu-trd-2018-2-190531* [6] projects, but for simplification purposes only the main IPs needed to properly understand the system are shown in the diagram, omitting the reset circuitry, the AXI communication interconnections and an unused GPIO (see full diagram in APPENDIX C).

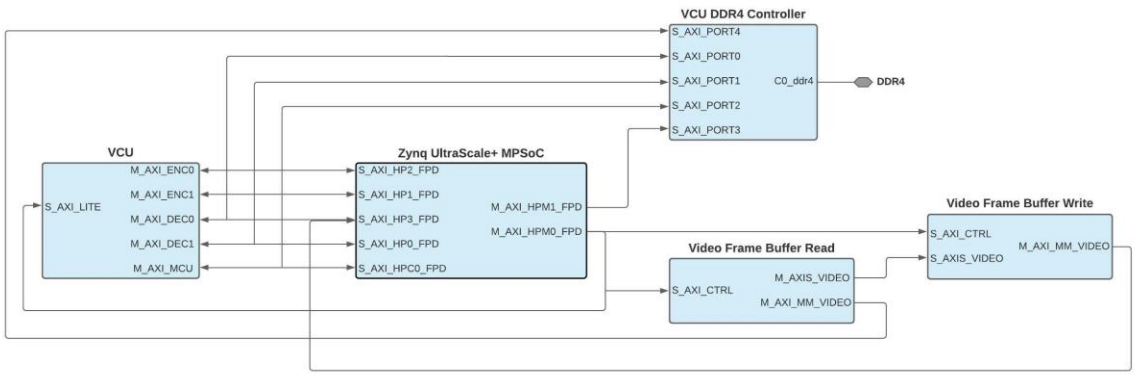


Figure 3-11: Image Capture PL Diagram.

The main IP is the Zynq UltraScale+ MPSoC where PetaLinux OS is running. In the PS, the USB3.0 camera is enumerated and configured, and the video stream is captured. This stream can be sent to the Display port through DMA Video Frame buffers or to VCU to encode it. Zynq UltraScale+ MPSoC has four bidirectional AXI interfaces with VCU to exchange video data, one Master AXI interface to control VCU and DMA buffers, one

Master AXI to control VCU DDR4 Controller and one Slave AXI interface coming from VCU.

VCU IP is composed by two modules, the Encoder and the Decoder and each implement a 32-bit MCU to run the firmware, control both blocks and handle interaction between hardware blocks and with APU [26].

- The Encoder module data path is built with two 128-bit AXI4 interconnections to fetch and write video data from external DDR memory attached to either the Processing System (PS) or the Programmable Logic (PL). [26]
- The Decoder module data path is built with two 128-bit AXI4 interconnections used to fetch video input data and store video output data from/to the system memory in the Processing System (PS) or in the Programable Logic (PL). [26]
- Control path for both modules is processed by MCU using two 32-bit AXI4 interconnections. One AXI-Lite slave used by the APU to control the MCU and one master interface used by MCU to communicate with the APU.

Finally, the video stream is sent to the Display Port Controller in two ways:

- Using DMA Video Frame Buffers. (Live-Video)
- Storing the frames in the DDR4 to be read by DPDMA, Display Port DMA. (Non-live Video)

3.3.2. PetaLinux project changes

Once the HW is defined and exported from Vivado, it is time to create and configure the PetaLinux project. The main difference between the previous PetaLinux image and the new one is in the rootfs libraries. In this new set of libraries, gstreamer, V4L2, OpenCV and OpenAmp functionalities are added to successfully build the desired video pipelines.

First, we create the project and export HW in the same way than in previous project step and with the same configuration:

```
$: petalinux-create --type project --template zynqMP --force –name
<project name>

$: petalinux-config --get-hw-description=<path to .hdf directory in
Vivado project folder>
```

Then, there are a set of VCU functions using GStreamer that needs to be added in a recipe file to make it selectable later in the rootfs configuration by means of this code line:

```
$: echo 'IMAGE_INSTALL_append = " gstreamer-vcu-examples"' >> <project
root>/project-spec/meta-user/recipes-core/images/petalinux-image-
full.bbappend
```

This way, when preparing the set of libraries of rootfs to be build, we can select all the needed functionalities.

Rootfs configurated is called as in previous section:

```
$: petalinux-config -c rootfs
```

In addition to all the libraries selected in the basic project, the following ones must be added too:

- filesystem Packages → libs → opencv → opencv, opencv - apps
- filesystem Packages → misc → gst-player
- filesystem Packages → misc → gst-plugins-base → base, apps
- filesystem Packages → misc → gst-plugins-good
- filesystem Packages → misc → gstreamer1.0-meta-base → base, video, x11
- filesystem Packages → misc → gstreamer1.0-plugins-bad
- filesystem Packages → misc → gstreamer1.0-plugins-base → base, apps
- filesystem Packages → misc → gstreamer1.0-plugins-good
- filesystem Packages → misc → openamp-fw-echo-testd
- filesystem Packages → misc → openamp-fw-mat-muld
- filesystem Packages → misc → openamp-fw-rpc-demo
- filesystem Packages → misc → v4l-utils → v4l-utils, libv4l, ir-keytable, media-ctl, rc-keymaps
- filesystem Packages → multimedia → all gstreamer
- Petalinux Package Groups → packagegroup-petalinux-gstreamer
- Petalinux Package Groups → packagegroup-petalinux-openamp
- Petalinux Package Groups → packagegroup-petalinux-opencv
- Petalinux Package Groups → packagegroup-petalinux-v4lutils
- user packages → gstreamer-vcu-examples

After configuring the rootfs, the project needs to be built and Image files needs to be packaged with the same commands:

```
$: petalinux-build
```

```
$: petalinux-package --boot --format BIN –fsbl
images/linux/zynqmp_fsbl.elf --u-boot images/linux/u-boot.elf --pmufw
```

```
images/linux/pmufw.elf -atf images/linux/bl31.elf --fpga
images/linux/system.bit -force
```

Then, the SD card can be configured and the new OS launched in the board.

3.3.3. GStreamer video pipelines

Using a text-based serial port communications program like Minicom, one can navigate into the root file system of the OS and manage some functions. To generate the video pipelines, GStreamer functions already added are needed among with V4L2 driver and KMSSINK plugin. Also, VCU is used to encode and/or decode the incoming video. The needed Software/Hardware stack is shown in *Figure 3-12*:

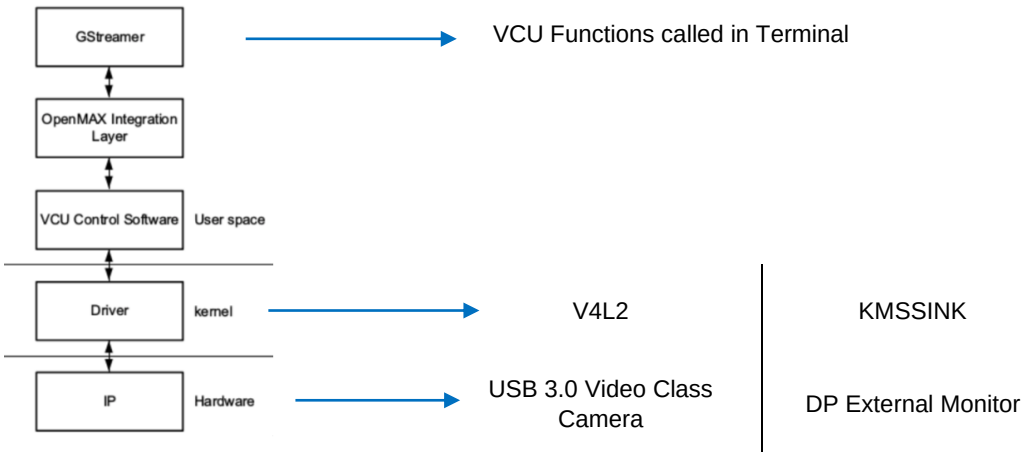


Figure 3-12: Software stack. Source: [26]

In the GStreamer VCU Demo functions there are two that can be used directly to display and store video (see APPENDIX C). The first one is *vcu-demo-camera-encode-decode-display.sh* in which the only parameter that may be changed is the resolution Full HD (1920x1080). This function is called by typing:

```
$: vcu-demo-camera-encode-decode-display.sh -s 1920x1080
```

The function directly enumerates the camera and the display, configures the devices parameters, and builds the pipeline. The Video Stream should be playing in the monitor after calling this function.

The second one is *vcu-demo-camera-encode-file.sh*. This function stores a video file called *camera_output.ts* in the current directory. In this function one can specify the resolution and the number of frames to be captured and stored.

```
$: vcu-demo-camera-encode-file.sh -s 1920x1080 -n 1000
```

Again, by calling the function, the camera and display are enumerated and the pipeline is mounted.

These two functions are useful but GStreamer allows the developer to directly build and configure the pipeline in the terminal. This way one can simplify or modify the parameters and devices used. For example, in the first function it is not needed to encode and decode the video and the second pipeline can be improved controlling the quality of the encoding. To do so, the developer will need to know how to enumerate the devices, the supported parameters of each one and how to build the pipeline.

To enumerate the USB camera and know its properties, two V4L2 commands can be used:

```
$: v4l2-ctl --list-devices        → to enumerate devices
$: v4l2-ctl -list-formats-ext     → to list devices input formats
```

To know the display modes supported for the decive one can consult the modes file using the command:

```
$: cat /sys/class/drm/card0-DP-1/modes
```

And check the kmssink capabilities with the command:

```
$: gst-launch-1.0 videotestsrc ! kmssink
```

Once the video source and the video sink are determined, the capabilities of the VCU encoder can be checked with the command:

```
$: gst-inspect-1.0 omxh264enc
```

Tables 3-3, 3-4 and *3-5* show the specifications summary of each device:

Camera	
Uncompressed YUV	**Compressed MJPEG**
VGA (640&480) @ 30 & 60 fps	VGA (640&480) @ 30 & 60 fps
HD (1280 x 720) @ 30 & 60 fps	HD (1280 x 720) @ 60 fps
FHD (1920 x 1080) @ 15, 30 & 60 fps	FHD (1920 x 1080) @ 60 fps
3MP (2304 x 1296) @ 15 & 30 fps	3MP (2304 x 1296) @ 60 fps
3.4MP (2304 x 1536) @ 12 & 24 fps	3.4MP (2304 x 1536) @ 48 fps
1920 x 1280 @ 25 & 50 fps	1920 x 1280 @ 50 fps
1152 x 768 @ 30 & 60 fps	1152 x 768 @ 60 fps
1280 x 960 @ 30 & 58 fps	1280 x 960 @ 58 fps
2048 x 1536 @ 21 & 42 fps	2048 x 1536 @ 50 fps

Table 3-4: Camera Video features. Source [11]

Video Coding Parameter	H.265 (HEVC)	H.264 (AVC)
Profiles	Main Main Intra Main10 Main10 Intra Main 4:2:2 10 Main 4:2:2 10 Intra	Baseline Main High High10 High 4:2:2 High10 Intra High 4:2:2 Intra
Levels	Up to 5.1 High Tier	Up to 5.2
Resolution and Frame Rate [1][2]	4096x2160p60 with specific device (-2, -3) 3840×2160p60 3840×2160p30 1920×1080p60 1920×1080p30 1280×720p60 1280×720p30	4096x2160p60 with specific device (-2,-3) 3840×2160p60 3840×2160p30 1920×1080p60 1920×1080p30 1280×720p60 1280×720p30
Bit Depth		
GStreamer	8-bit, 10-bit	8-bit, 10-bit
OMX	8-bit, 10-bit	8-bit, 10-bit
VCU Control Software	8-bit, 10-bit	8-bit, 10-bit
Chroma Format		
GStreamer	4:2:0, 4:2:2	4:2:0, 4:2:2
OMX	4:2:0, 4:2:2	4:2:0, 4:2:2

Table 3-3: VCU Encoder features. Source: [26]

Format	BPC/BPP	R	G	B	Cr	Y	Cb	Cr/Cb	Y
RGB	6/18	[35:30]	[23:18]	[11:6]					
RGB	8/24	[35:28]	[23:16]	[11:4]					
RGB	10/30	[35:26]	[23:14]	[11:2]					
RGB	12/36	[35:24]	[23:12]	[11:0]					
YCbCr444	6/18				[35:30]	[23:18]	[11:6]		
YCbCr444	8/24				[35:28]	[23:16]	[11:4]		
YCbCr444	10/30				[35:26]	[23:14]	[11:2]		
YCbCr444	12/36				[35:24]	[23:12]	[11:0]		
YCbCr422	8/16							[35:28]	[23:16]
YCbCr422	10/20							[35:26]	[23:14]
YCbCr422	12/24							[35:24]	[23:12]
YONLY	8/8								[35:28]
YONLY	10/10								[35:26]
YONLY	12/12								[35:24]

Table 3-5: Display Port Live Video Format. Source: [8]

The video specifications of the camera need to match with the supported specifications of the VCU and Display Port controller.

To display the video without encoder and decoder, it is only needed to list the video source (camera) and its settings and the video sink (external monitor). For this test, the video format chosen is 8-bit YUV 422 at FHD (1920x1080) at 60fps.

```
$: gst-launch-1.0 v4l2src device=/dev/video0 ! video/x-raw, \
> format=\(string\)UYVY, width=1920, height=1080, framerate=60/1 ! \
> kmssink bus-id=fd4a0000.zynqmp-display fullscreen-overlay=1
```

To improve the quality of the encoded video [26], the following pipeline has been used:

```
$: gst-launch-1.0 -e v4l2src device=/dev/video0 num-buffers=300 ! \
> video/x-raw,width=1920,height=1080,framerate=30/1 ! \
> videoconvert ! video/x-raw, format=\(string\)NV16 ! \
> omxh264enc control-rate=constant gop-mode=low-delay-p gdr-
mode=vertical \
> gop-length=0 target-bitrate=5500 ! \
> video/x-h264, profile=high-4:2:2 ! h264parse ! queue ! \
> mpegtsmux name=mux mux. ! filesink location=/home/improved.ts
```

In *Figure 3-13*, the pipeline is shown identifying each configuration with its meaning:

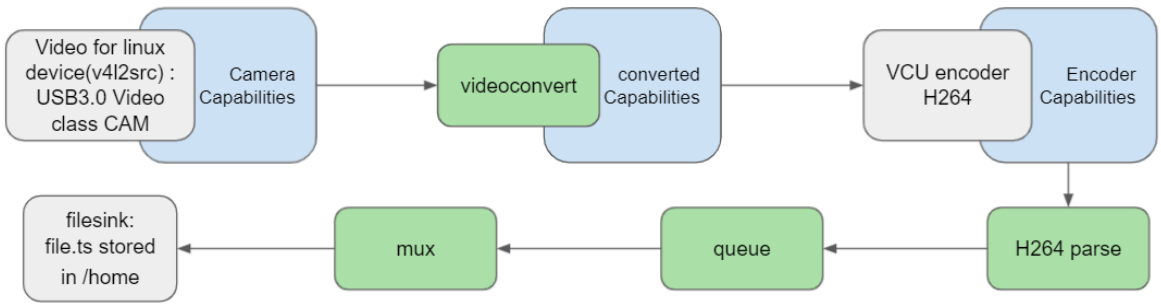


Figure 3-13: GStreamer custom capture-encode-store_file pipeline

- 300 frames of the video source set as default video format (YUYV 422 8-bit) at FHD, 30 fps are captured.
- These frames are converted into NV16 format: YUV file contains 4:2:2 8-bit video samples stored in planar format with all picture Luma (Y) samples followed by interleaved U and V Chroma samples [26].
- Encoder is configured using H.264 (AVC) high 4:2:2 profile and optimized parameters to be stored at maximum quality with the better performance:
    - Control rate = constant.
    - Gop-mode = low-delay-p. Specifies group of pictures configuration. low-delay-p (IPPPPP….). [26]

- Gdr-mode = vertical. When gop-mode is set to low_delay_p, gdr-mode specifies which scheme should be used for Gradual Decoder Refresh: vertical option means that a vertical bar moving from left to right is used. [26]
- Gop-length = 0. Distance between two consecutive Intra frames. Specify integer value between 0 and 1,000. Value 0 and 1 corresponds to Intra-only encoding. [26]

For better understanding of Gop-mode and Gop-length, some knowledge of H.264 Video Codec is required.

To reduce the size of a video, the frame is divided into independent sections called slices, that are also divided into smaller sections called Macro Blocks (MBs). These MBs are groups of blocks containing Luma pixels (Y) or croma pixels (Cb, Cr). See *Figure 3-14*

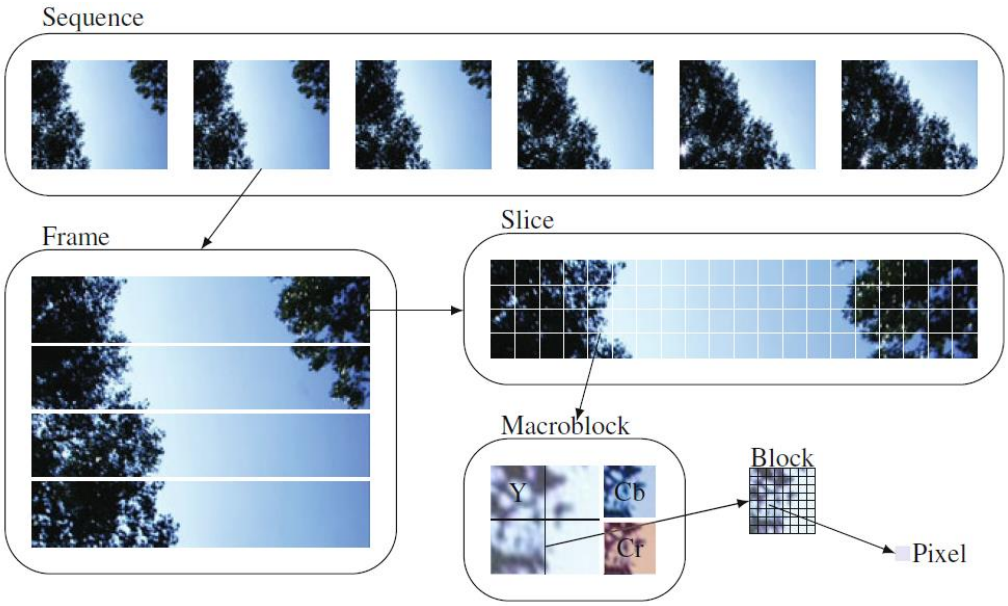


Figure 3-14: H.264 (AVC) frame distribution.

This way, MB of some frames can be predicted using different techniques instead of recording and storing each full frame.

The H.264 video codec defines three main types of frames which are I-frames, P-frames and B-frames. A Gop (group of pictures) is a set of combined frames of different type.

- I-frames: Intra prediction frames. Is independent of the other frames and its MBs are predicted based on adjacent MBs of the same frame.
- P-frames: Predicted frames. Uses Intra prediction and motion estimation to predict its MBs. It depends on previous frames that can be either I-, P- or B-frames.
- B-frames: Bidirectional predicted frames. It uses bidirectional motion estimation to predict its MBs. It depends on previous and/or future frames.

3.4. Processing image in host PC using a Convolutional Neuronal Network

This project step is focused on processing the captured images taken with the system configured in the previous one in the host PC to validate the CNN before implementing it in the development board.

Once the CNN is chosen, it is tested first with a single image to understand the performance. Then, the code is adapted to process a downloaded sample video and validated with own captured video.

The original code can be downloaded from the Cornell University website [40].

3.4.1. SegNet

Basing in the objectives and the scope of the project, any of the CNN may be used since the main issue is how to successfully implement a system running this NN in an FPGA. However, the main purpose is to make the first move to finally achieve an automotive application using FPGA in front of GPU to run ML algorithms. So, even if this final application is out of the scope of the project, it seems appropriate to work with a Semantic Segmentation Neural Network, for instance SegNet.

SegNet is a Deep fully convolutional Neural Network for semantic pixel-wise segmentation. The architecture of its core trainable segmentation engine consists of an encoder network with its corresponding decoder network followed by a final pixel-wise classification layer. (See *Figure 3-15*). The encoder network is built with 13 convolutional layers which corresponds to the first 13 convolutional layers in the VGG16 network [38] designed for object classification. Each encoder layer has a corresponding decoder layer, so the decoder network is also composed with 13 convolutional layers. The output of the decoder network is fed to a multi-class soft-max classifier to produce class probabilities for each pixel independently. [33]

Figure 3-15 shows the architecture of SegNet, its layer types and an example of input-output images:

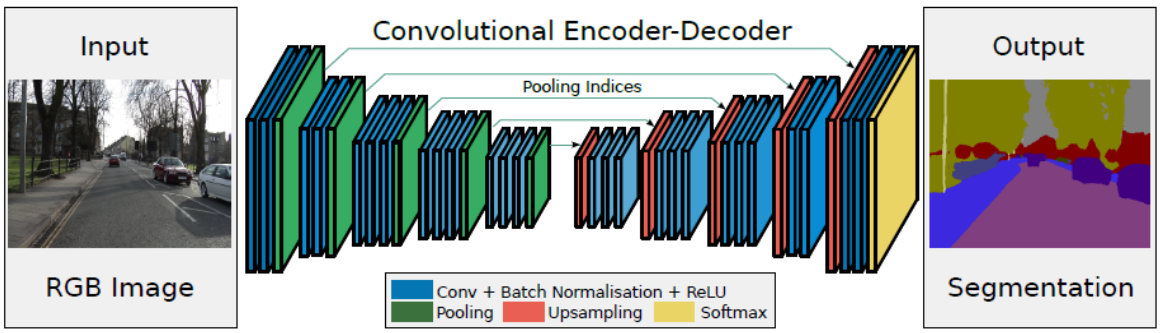


Figure 3-15: SegNet architecture. Source: [33]

3.4.2. SegNet Image processing code

To test the SegNet CNN, sample images have been processed in a python project. This project consists in a main file that reads the image from a folder and calls the SegNet to process it. Its file structure is shown in *Figure 3-16:*

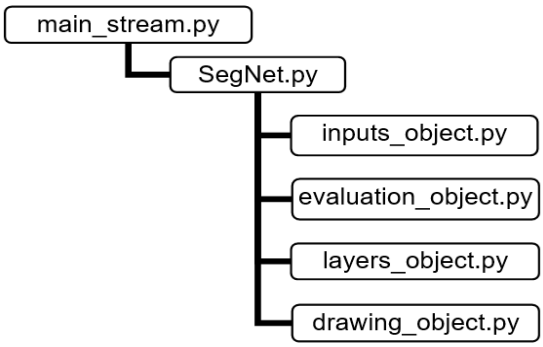


Figure 3-16: SegNet project code file architecture.

The main file reads images from a specified path using the OpenCV command *cv.imread()*. This command, by default, converts from the YUV colour space to BGR colour space. Since SegNet class expects RGB image, before calling it, the image should have another colour space transformation from BGR to RGB using the command *cv2.cvtColor(image, cv2.COLOR_BGR2RGB)*. Then, SegNet is called and the output image is stored in a specified folder.

The code of the main function is:

```
def main():
      for arg in sys.argv[1:]:
        print('Analysis of file: ',arg)
        if os.path.isfile(arg):
            im = cv2.imread(arg)
            im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
            pred_tot, var_tot, inference_time, im =
                SegNet().visual_results_external_image(im,
                FLAG_MAX_VOTE=False)
            cv2.imwrite('res_image/img.jpg', im)
        else:
            print('File ', arg, ' is not found.')
            exit()
if __name__ == "__main__":
    main()
```

The *SegNet.py* file is in charge to resize the image to its required size (480x360), process the image with the help of *inputs_object.py, evaluation_object.py* and *layers_object.py*, draw the output image with *drawing_objects.py* and return the image.

This code has been tested with sample images and works properly. This may be enough since in the ZCU104 development board, in the ideal case, it may work frame by frame. But since the processing time for each frame predictably will exceed the required timing to get real time operation, it may be useful to leave open the possibility to process a video file stored in the system.

To do so, first the code of main file needs to be modified to read and write a video file instead of a single image. Instead of opening a single image with *cv.imread()* function, now the video needs to be opened with the OpenCV function *cv2.VideoCapture()*. Then, go through the video frame per frame processing each and store all the frames in a selected folder. Finally, a custom function to build the video from the stored frames is called and the resulting video is stored. The modified code is as follows:

```
def main():
    for arg in sys.argv[1:]:
        print('Analysis of file: ',arg)
        if os.path.isfile(arg):
            cap = cv2.VideoCapture(arg)
            i = 1
            while (cap.isOpened()):
                ret, frame = cap.read()
                if ret == False:
                    break
                frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                pred_tot, var_tot, inference_time, im =
                   SegNet().visual_results_external_image(frame,
                   FLAG_MAX_VOTE=False)
                cv2.imwrite('res_frames/frame'+ str(i).zfill(4)
                   +'.jpg',im)
                i += 1
            cap.release()
            cv2.destroyAllWindows()
            createVideo('res_frames')
        else:
            print('File ', arg, ' is not found.')
            exit()
if __name__ == "__main__":
    main()

def createVideo(frames_path):
    img_array = []
    frame_array = glob.glob(frames_path+'/*.jpg')
    frame_array.sort()
    for filename in frame_array:
        img = cv2.imread(filename)
        height, width, layers = img.shape
        size = (width, height)
        img_array.append(img)

    out = cv2.VideoWriter('output.avi', cv2.VideoWriter_fourcc(*'DIVX'),
      15, size)

    for i in range(len(img_array)):
        out.write(img_array[i])
    out.release()
    return
```

Using the new code, two video samples are checked. The first one is a sample video downloaded from internet captured with a car front camera recording a journey through a city. The point to use this video is to simulate the output of the CNN when an autonomous driving camera is used.

Then, the second video is one recorded with the capture image system built in the previous step in order to verify that the output images from the capture pipeline are a proper input for the processing stage.

See the results in the section *4.3. CNN capture in host PC*

3.5. Implementing Convolutional Neuronal Network in embedded system

Finally, this is the last stage of the project. At this stage, all the previous subsystems and the acquired knowledge are merged and applied to build the final full system.

This stage may result in a complete embedded system that captures video from a USB camera, process either each frame in real time or a stored video file with a segmentation CNN and shows the image in a screen via display port. Nevertheless, it cannot be completed because the Xilinx process to implement the CNN in its DPU IP using its software DNNDK is not as robust as advertised. It is explained later but, in this report, the process has been followed properly and at the end an error that even the Xilinx specialist doesn't know how to fix occurs.

Even so, the process to implement an embedded PetaLinux operating system with a capture, process and display a video pipeline implementing a CNN in PL is the shown in *Figure 3-17* and explained below.

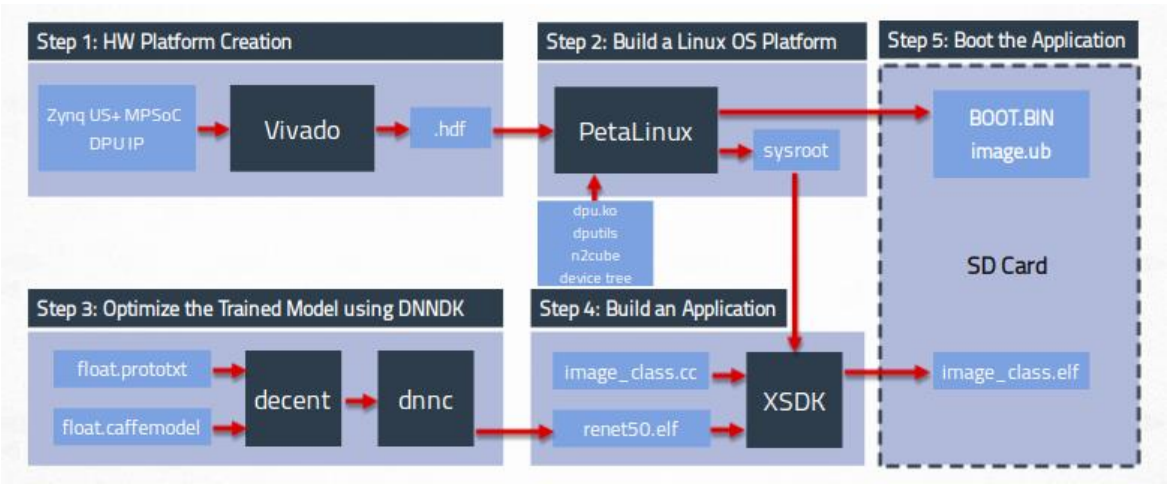


Figure 3-17: Development Flow of an Edge-based AI Application example. Source: [48]

The first step is, as in previous project stages, to define de HW in the PL side, and export it to be included in the PetaLinux project. Then, the OS image is built in a same way than done until now but adding the DPU capabilities and modifying the device tree. The main difference is that before configuring the rootfs, the CNN needs to be implemented in the

DPU using the DNNDK Xilinx software. Finally, the resulting files along with the bare metal code to call the Net are managed in the Xilinx SDK.

3.5.1. Full System Programable logic

The block design in the PL side is almost the same as that used in the Image Capture subsystem. All its functionalities have been kept and only the Deep Learning Processing Unit (DPU) has been added in order to implement the CNN and process the obtained images using that IP. *Figure 3-18* illustrates the final HW Diagram. (See Appendix D for the complete Block Design).

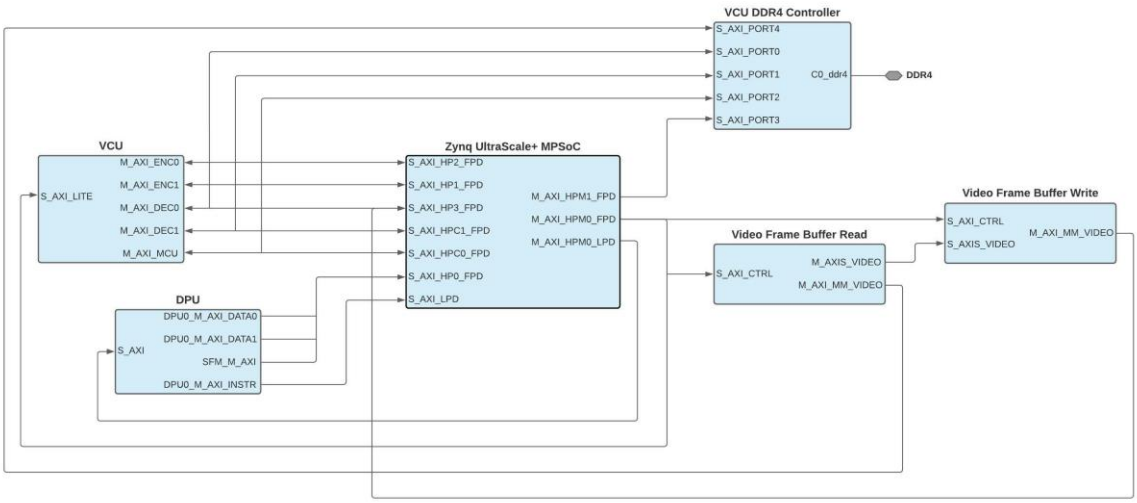


Figure 3-18: Full system PL Diagram.

When adding the DPU in the project, there are some aspects to consider.

1. The chosen CNN can be implemented in the DPU.

The DPU has a set of layer operations that are supported and not in any order. One needs to ensure that the layers in the chosen Network and its order are supported. In the case of this project, this verification has been done at the moment of selecting the SegNet network. The supported operations in the DPU are:

- Conv
  - Dilation
- Pooling
  - Max
  - Average
- ReLU/Leaky Relu/ Relu6
- Full Connected (FC)
- Concat
- Elementwise
- Deconv
- Depthwise conv
- Batch Normalization

- Mean scale
- Upsampling
- Split
- Reorg
- Softmax (additional core)

Since as shown in *Figure 3-15*, SegNet uses Conv, Batch Normalisation, ReLU, Pooling, Upsampling, Deconv and Softmax layers, it seems a suitable Net.

Regarding the order, *Table 3-6* can be consulted to check the availability of the operations order:

Layer Type \ Next Layer	Conv	Deconv	Depth-wise Conv	Inner Product	Max Pooling	Ave Pooling	BN	ReLU	LeakyReLU	Element-wise	Concat	As Input	As Output
Conv	●	●	○	●	●	○	●	●	○	●	●	●	●
Deconv	●	●	○	●	●	○	●	●	○	●	●	●	●
Depth-wise Conv	●	●	○	●	●	○	●	●	○	●	●	●	●
Inner Product	●	●	○	●	●	○	●	●	○	●	●	●	●
Max Pooling	●	●	○	●	●	○	○	×	×	●	●	●	●
Ave Pooling	○	○	○	○	○	○	○	×	×	○	○	○	○
BN	●	●	○	●	●	○	○	●	×	●	●	○	○
ReLU	●	●	○	●	●	○	○	×	×	●	●	---	●
LeakyReLU	○	○	○	○	○	○	○	×	×	○	○	---	○
Element-wise	●	●	○	●	●	○	○	●	○	●	●	---	●
Concat	●	●	○	●	●	○	○	×	×	●	●	---	●

● : Support ×: Not support ○ : Support when selecting additional features

Table 3-6: Supported layers order in DPU IP. Source: [48]

One can see that in these tables there are not all the available nets. It Is a resume of the more relevant ones but to be sure, the compatibility of SegNet with the DPU has been asked to the Xilinx contact and he confirms that the net is suitable also regarding the layers order.

2. The DPU configuration.

The DPU IP has a few parameters that can be configured when adding it to the block design. The more relevant are:

- The number of DPU cores: DPU can run with multiple cores (up to 3) to achieve higher performance. This also implies more resources consumption of the programable logic. In this case, considering the trade-off between the required performance and the available resources, only one core has been decided to use.
- The Arch of DPU: The DPU has different architecture configurations depending on the level of parallelism of the convolution unit. It depends also on the board used. In the case of the Zynq UltraScale +MPSoC and the performance required for SegNet, the chosen architecture is B2304.

- Number of SFM cores: The Softmax function is a layer of a CNN that converts the output vector of n real numbers into a vector of n real numbers that sum to 1. This operation is supported using an additional core to handle it, so since the SegNet uses Softmax operation, this parameter needs to be set as 1.

3. Connect DPU with Zynq UltraScale +MPSoC and Assign register address for DPU.

The DPU IP has only one slave interface and a variable number of master interfaces depending on the number of used cores. Each DPU core has three master interfaces, one for instruction fetch and two for data. The SFM core, if enabled, has one additional master interface.

In the project use case, only one DPU core is used and Softmax is enabled so there are one slave interface and four master interfaces in total.

Xilinx recommends connecting the slave port with the M_AXI_HPM0_LPD port of the Zynq UltraScale +MPSoC. This recommendation has been followed since that port was unused in the previous block design.

Regarding the Master interfaces, Xilinx recommends avoiding AXI_interconnections if possible because it may add some delay in comparison with direct connection with the PS. Nevertheless, due to the high number of interconnections it is not possible in this design. When this is not possible, Xilinx recommends connecting the master port for instruction fetching to the S_AXI_LPD port of the PS due to its low bandwidth requirements and the data master ports to the higher priority (low number) port of the PS high bandwidth ports. In this project, the three data master ports are connected with S_AXI_HP0_FPD port of the PS so both recommendations have been fulfilled.

In addition, it is important to properly connect the interruption, reset and clocking signals.

Once all the connections are well configured, the base address of the AXI slave interface must be assigned to any of the accessible address by the CPU in the Vivado address editor. Xilinx recommend setting the DPU slave interface to 0x8F00_0000 for MPSoC devices allowing a minimum space of 16 MB. This recommendation has been followed.

Finally, the HW needs to be exported to later adding it to PetaLinux project.

See Programable Logic Reports in section 4.4.Final PL Vivado Block Design reports

3.5.2. PetaLinux changes

This is the final PetaLinux project configuration for the complete system. The main difference with the previous one is that now some libraries and recipes are added to support DPU and DNNDK behaviour, and the Device tree may be modified. Also, the resulting sysroot will be exported to merge all the software using Xilinx SDK.

First, we create the project and export the new HW the same way as before.

Then, it is time to add all the new features:

1. DPU driver:

The DPU driver needs to be added to allow the kernel to handle it. To do so, a recipe for a new module must be included in *PetaLinux_project_folder/project-spec/meta-user/recipes-modules*. Thus, the system will include this module when building the project. The DPU module is sourced by Xilinx in the *xilinx-zcu104-v2018.3-final-v2.bsp* [4] and its content is that shown in *Figure 3-19*.

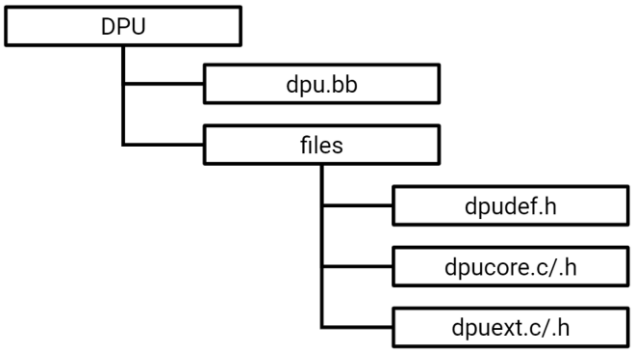


Figure 3-19: dpu module file structure. It includes the recipe and the source code files.

In addition to including the dpu module, the Linux kernel requires an indication to properly insert the DPU kernel driver (dpu.ko) at boot.

The file *PetaLinux_project_folder/project-spec/meta-user/recipes-kernel/linux/linux-xlnx_%.bbappend* should be modified by adding the line:

```
LINUX_VERSION_EXTENSION = "+"
```

2. DPU Device tree definition

Until now, the default device tree has been used. Now, it needs to be modified to add the details of the DPU core and DPU softmax core. The new device tree may have the *system-user-dtsi* name and must be placed in the folder *PetaLinux_project_folder/project-spec/meta-user/recipes-bsp/device-tree/files/*

The code to be added in the current device tree is the one bellow. This code is a template code provided by Xilinx for the DPU instantiation in the device tree. There are three memory addresses that needs to be changed to the defined in the programable logic. They are marked in black in the below code and they are the DPU base address and the interrupt values for both softmax and DPU cores.

```
amba {
	dpu {
		compatible = "xilinx,dpu";
		base-addr = <0x8f000000>;

		dpucore {
			compatible = "xilinx,dpucore";
			interrupt-parent = <&gic >;
			interrupts = <0x0 0x5C 0x1>;
			core-num = <0x1>;
		};

		softmax {
			compatible = "xilinx, smfc";
			interrupt-parent = <&gic>;
			interrupts = <0x0 0x5D 0x1>;
			core-num = <0x1>;
		};
	};
};
```

The DPU base address is the one selected in the Vivado address editor when adding the DPU IP in the block design. That is, the recommended by Xilinx for UltraScale +MPSoC devices 0x8F000000.

Regarding the interruptions, its value will depend on their connections on the Vivado block design. In the final design of the project, the dpu interruption *dpu_interrupt_[0:0]* and the softmax interruption *sfm_interrrupt* are connected through a Concat IP with other interruptions to the *pl_ps_irq0(3)* and *pl_ps_irq0(4)* respectively. Looking at the UG1058 (Zynq UltraScale+ Device Technical Reference Manual) [8] one can find the IRQ number (GIC) of the PL_PS_Group0 which corresponds to the port *pl_ps_irq0[7:0].* That numbers are from 121 to 128, DPU and softmax interruptions are connected to 3 and 4 respectively, if IRQ number for GIC are 124 and 125. However, the number to be placed in the device tree is not the GIC IRQ number but the PetaLinux IRQ number and to find these numbers one must subtract 32 to the GIC numbers (this is because reserved interruption numbers for the OS). So, the PetaLinux IRQ numbers for DPU and Softmax are 92 and 93, in hexadecimal 0x5C and 0x5D.

3. DPU run-time libraries and utilities

Finally, the dnndk app must be added as a recipe in the path *PetaLinux_project-folder/project-spec/meta-user/recipes-apps* to get the DPU utilities and libraries. The folder *dnndk* provided by Xilinx contains the file.bb that contains the recipe for including the software components required for the DPU core, including driver, header files, as well as utilities.

The file structure of the dnndk folder is shown in *Figure 3-20*

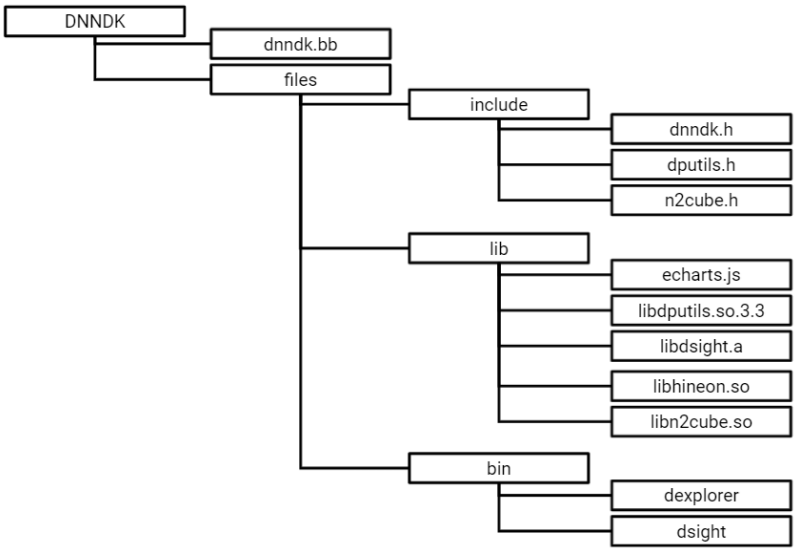


Figure 3-20: DNNDK app file structure. It includes the recipe, the DPU utilities and the required libraries.

Once these modifications are included in the meta-user folder, the recipes need to be included in *PetaLinux_project-folder/project-spec/meta-user/recipes-core/image-full.bbappend* to make them selectable in the rootfs configuration. To do so, the following commands are used:

```
$: echo 'IMAGE_INSTALL_append = " dpu"' >>./project-spec/meta-user/recipes-core/images/petalinux-image-full.bbappend
```

```
$: echo 'IMAGE_INSTALL_append = " dnndk"' >>./project-spec/meta-user/recipes-core/images/petalinux-image-full.bbappend
```

Then the rootfs should be called as in previous steps

```
$: petalinux-config -c rootfs
```

and select the new applications in the pop-up windows.

- modules → dpu
- user packages → dnndk

Then, build the project in the same way as explained and rebuilt again activating the sdk option to build Yocto e-SDK taping the following command:

```
$: petalinux-build –sdk
```

Finally, package the sysroot to be used by SDK with the command:

```
$: petalinux-package --sysroot -d ../sdk
```

3.5.3. CNN files generated with DNNDK

Next step consists of translating a pretrained model based on Caffe or TensorFlow frameworks into the needed files to be implemented in the DPU IP. The one in charge of that is the Xilinx software tool called Deep Neuronal Network Development Kit (DNNDK). DNNDK is a software to deploy deep learning inference applications on the DPU and it is composed by two models: the model compression called DECENT and the model compilation called DNNC.

The Deep Compression Tool DECENT has two stages. The first stage is an optional stage of pruning, in which the Neuronal Network is simplified to get a small one and to reduce the number of operations. This simplification is done by removing connections and nodes with near zero weights. After this stage, even if the resulting NN is equivalent to the original one, a retraining process is required to ensure the performance is actually equivalent. One has to paid to get available his first stage and since it is optional, it will be skipped in this project. The second stage is the Quantization stage, in which the 32-bit-floating-point weights used when training neuronal networks are converted into 8-bit integer (INT8) to allow the DPU to process it.

Figure 3-21 shows the DECENT flow.

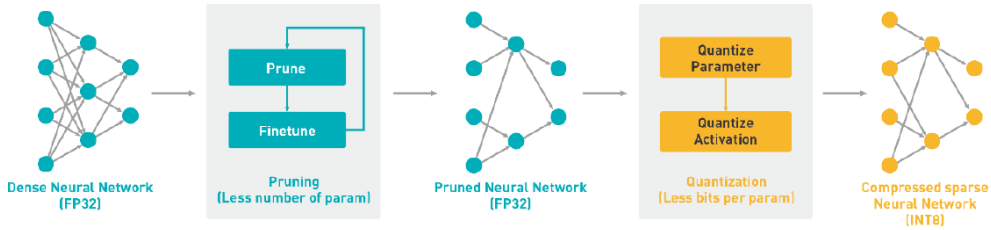


Figure 3-21: DECENT Pruning and Quantization Flow. Source: [46]

The Deep Neural Network Compiler DNNC is composed by three stages. A front-end parser that is responsible for parsing the Caffe/TensorFlow model and generates an intermediate representation (IR) of the input model. The optimizer, which handles optimizations based on the IR. And the code generator that maps the optimized IR to DPU instructions.

The architecture of the DNNC is represented in *Figure 3-22* below:

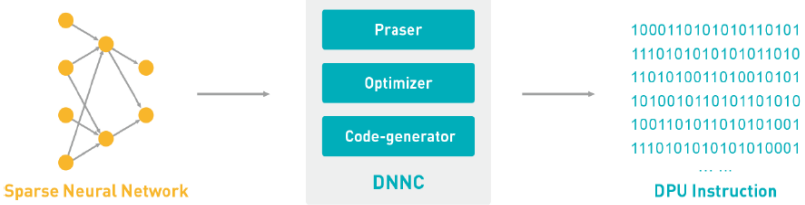


Figure 3-22: DNNC Components. Source: [46]

So, first step to be realized is the Quantization with DECENT. With the resultant fixed-point network model, the memory bandwidth requirement and the computing complexity are reduced, and it provides a faster speed and higher power efficiency than the original model. In this process, a calibration data set will be needed to analyse the distribution of activation values for calibration. This data set is a small set of images compared with the training data set.

The wotkflow of this process is shown in *Figure 3-23*. It takes a float model as input, in this project use case, a frozen GraphDef file since the used model is trained in TensorFlow. DECENT_Q does some pre-processing removing useless nodes or folding batchnorms and then it quantizes the weights/biases and activations. Then is when the calibration dataset is needed, DECENT runs some iterations of inference to calibrate the activation and improve the precision of the resulting quantized graph. Finally, the quantized model is transformed into a DPU deployable model named *deploy_model.pb*. ready to be used in the model compiler DNNC. [46]

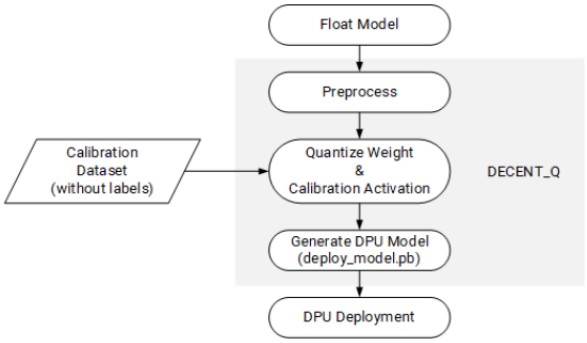


Figure 3-23: DECENT Workflow. Source: [46]

To execute the DECENT quantization and configure the required parameters, Xilinx provides a script called *decent_q.sh* with the following code:

```
decent_q quantize \
      --input_frozen_graph deployfinal.pb \
      --input_nodes input \
      --input_shapes ?,480,360,3 \
      --output_nodes conv/classifier/output \
      --method 1 \
      --input_fn SegNet_input_fn.calib_input \
      --gpu 0 \
      --calib_iter 100 \
      --output_dir ./quantize_results \
```

- In the input_frozen_graph, directory of the frozen GraphDef file of the TensorFlow model must be included as the input system.
- The calibration dataset is called by the function in the parameter input_fn and set it in the variable included in input_nodes parameter.
- The shape of the input images expected from the net must be specified in input_shapes.
- The output directory is specified in output_dir and the output node in output_nodes.
- Finally, one can choose if a gpu is used to accelerate the process, the number of iterations of the calibration and the followed method which is always leaved as default (1= min-diffs).

After that compilation an error due to the different version of TensorFlow between the trained model and the supported by DNNDK occurs. This error happens at the end of the project and with all the previous and current steps well configured and performed. This limitation is not advised and will represent to repeat the previous steps regarding the neuronal network in the host and the DPU IP configuration and requisites. This is an amount of hours that can not be taken up by the time slot of an academic work. However, the last steps of the project will be explained as they need to be implemented because they were in the scope of the project and a lot of research has been done to understand them.

Regarding the compilation step, Xilinx provides another script called *dnnc.sh* with some parameters to adjust. This step will convert the *deploy_model.pb* obtained in the previous step to a *comp_model.elf* file for the DPU kernel. The content of the file is the following:

```
dnnc-dpu1.4.0.1 \
      --parser=tensorflow \
      --frozen_pb=./quantize_results/deploy_model.pb \
      --dpu=2304FA \
      --cpu_arch=arm64 \
      --mode=normal \
      --net_name=SegNet \
      --output_dir=./compile_results
```

- The parser needs to be set to tensorflow and the net_name to SegNet.
- In frozen_pb needs to be specified the path to the *deploy_model.pb* file and in the output_dir parameter, the output directory to store the generated files.
- The DPU architecture configured in the IP must be specified in dpu parameter, the CPU architecture in the cpu_arch parameter and the mode is leaved as “normal”.

Once the compilation is finished, the .elf file to be included in the XSDK is generated.

3.5.4. XSDK Bare metal application

Once the neural network model is compiled into the *comp_model.elf* file to be stored in the DPU, it is also needed to define a bare metal application running in the CPU to call the DPU and perform if needed a pre or post processing of the images. This CPU code is in charge also to handle the different kernels needed depending on the type of operations in the model. For example, in the project use case there is a kernel for all the net layers but softmax and then another kernel specific for softmax operation. If there were any operation not supported by DPU (which is not the case in this report), it may be implemented by the CPU.

So, last step is to compile in XSDK the neural network .elf file among with the C++ bare metal application and the sysroot resulting in compiling PetaLinux. When this compilation is finished, an *image_clas.elf* file is ready to be placed in the boot SD as a description of the application. This is called hybrid compilation.

Before launching XSDK and compiling the application, the rootfs of the project must be specified in an environment variable to easily indicate the software where it is located. This can be done with the export command:

```
$: export SYSROOT=<path to PetaLinux project>/sdk/sysroots/aarch64-xilinx-linux
```

Then, the *comp_model.elf* file may be converted to a Linux shared library to be included in XSDK. To do so, the gcc linker needs to be used by the following command:

```
$: aarch64-linux-gnu-gcc -fPIC -shared <path to .elf file >/comp_model.elf -o libdpumodel.so
```

Once these preparations are made, the XSDK software can be sourced and launched.

In the software, a new application project may be imported with the following main settings:

- OS platform: linux
- Processor: psu_cortexa53
- Language: C++

A new empty application is imported, and the code can be written.

But before that, there are some configurations that need to be modified to success in working with an opencv code, using the proper root file system and with all the required libraries. These configurations are:

- C++ Build Settings/ ARMv8 Linux g++ compiler/ Miscellaneous
  - ➔ append the parameter `--sysroot=${SYSROOT}` in Other Flags
- C++ Build Settings/ ARMv8 Linux g++ linker/ Miscellaneous
  - ➔ append the parameter `--sysroot=${SYSROOT}` in Linker Flags
- C++ Build Settings/ ARMv8 Linux g++ linker/ Libraries
  - ➔ Add lib n2cube
  - ➔ Add lib dputils
  - ➔ Add lib opencv_core
  - ➔ Add lib opencv_imgcodecs
  - ➔ Add lib opencv_highgui
  - ➔ Add lib opencv_imgproc
  - ➔ Add lib oipencv_videoio
  - ➔ Add lib pthread

Finally, once all are well configured, the code can be written.

The C++ code is composed by three main parts: The pre-processing of the acquired images, the handling of the DPU using the proper DNNDK API functions [46] and the post-processing of the images.

The main function is in charge of configuring all the required elements, launch the main task calling the Model and close all.

First, a DPUKernel and DPUTask variables must be initialized as:

```
DPUKernel *kernelSegNet;
DPUTask *taskSegNet;
```

Then, the DPU driver needs to be opened and the kernel and task properly created using DNNDK API commands:

```
dpuOpen();
kernelSegNet = dpuLoadKernel(comp_model.elf);
taskSegNet = dpuCreateTask(kernelSegNet, 0);
```

The model task can be now called with the command:

```
runSegNet(taskSegNet);
```

This function is the one in charge to do the pre-processing, call the DPU and realize the post-processing. It consists in a capture thread, a worker thread and a display thread.

First, the capture thread can work in two modes. It captures an image from camera and puts it to input queue or it captures n frames and sends it to the VCU to be encoded and stored as a video file, which is put into the input queue.

Then, the worker thread is called and performs three operations: gets an image from input queue and realizes the pre-processing, processes it with the model and performs the post-processing before putting the image back to the display queue. The performance of this thread is equivalent to the SegNet project code used in the host PC.

For the first operation, the `dpuInput()` function is called. It gets the image from the input queue and resizes it to the already commented expected shape by the SegNet function. If the input is a video file, it splits the video into frames and reshapes each frame. Then, prepares a set of images.

When the input is prepared, the task calling the DPU is called with the command:

```
dpuRunTask(taskSegNet);
```

When the video file mode is used, this set of commands are called recursively.

This task manages the DPU to process the input images. To get the output from the DPU in DPU INT8 format, and the data pointer from this output the following commands are used:

```
DPUTensor* dpuOutTensorInt8 = dpuGetOutputTensorInHWCInt8(taskSegNet,
OUTPUT_NODE);
DPUResult = dpuGetTensorAddress(dpuOutTensorInt8);
```

This is needed because the Softmax operation needs a separate call, so the final layer is applied using the command:

```
dpuRunSoftmax(DPUResult, softmax, channel, 1, scale);
```

After that, the last operation of the worker thread is performed by the CPU with the function `dpuOutput()`. This function draws the input image as the drawing_objects.py function and puts the processed image to the display queue. In case the video file option is used, it picks up all the frames, process them as described and then mounts again the video file.

Finally, the display thread gets the output image from the display queue and sends it to be displayed by the external monitor.

Once the runSegNet function ends, the main function destroys the kernel and the task and closes the DPU driver with the following lines:

```
dpuDestroyTask(taskSegNet);
dpuDestroyKernel(kernelSegNet);
dpuClose();
```

Lastly, when the code is finished, the project can be built and the *image_clas.elf* file is generated to be used in the system.

3.5.5. Boot the Application

At this point, all the required files are generated and prepared to be placed in the SD card.

First, in the *boot* partition, the *BOOT.bin* and *image.ub* files need to be placed as in previous configurations.

Then, after deploying the root file system in the *root* partition, the generated library file and the application file for our model must be included in the suitable directory. The previously generated library file *libdpumodel.so* must be placed under /usr/lib/ directory and the application file *image_clas.elf* may be placed in the working directory to be called by the user when the application in the PetaLinux OS is launched.

By doing these modifications, the SD card is properly configured and can be placed into the development board and power on the system.

After logging in the board, before calling the application, the following command need to be written to properly configure the dpu to use the shared library stored in /usr/lib/:

```
$: export DPU_COMPILATIONMODE=1
```

Finally, the system is ready to run the application by writing:

```
$: <path_to_.elf_file>/image_clas.elf
```

4. Results

4.1. Results Embedded PetaLinux

Figures 4-1, 4-2, 4-3 and *4-4* show the success in the different steps of the generating and booting a PetaLinux embedded OS image.

1. PetaLinux build

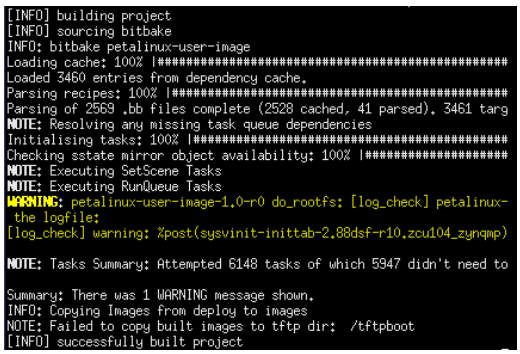


Figure 4-1: Terminal capture showing success in build PetaLinux project.

In this image, there is a warning that is not relevant. Xilinx contact confirms that it can be ignored, and the project is working well. Also, there is a Fail to copy built images to tftp. This is also correct, because in this project tftp is not used.

2. Image files generated

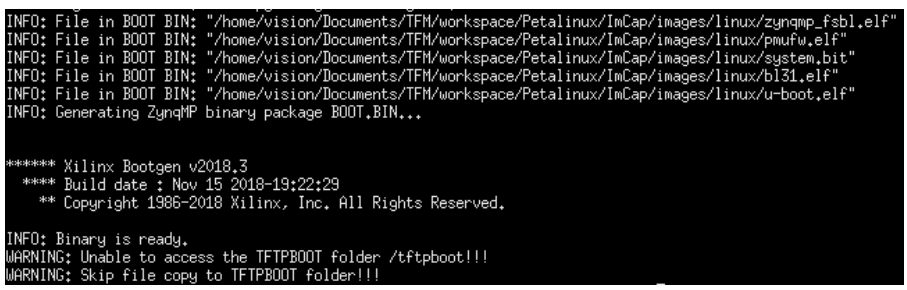


Figure 4-2: Terminal capture showing success in generating PetaLinux image files

There is again a warning referred to the tftp usage.

3. SD card with partitions

In *Figure 4-3* there are two windows related to the two partitions of the SD card. In one it can be seen the boot files *BOOT.BIN* and *Image.ub* and in the other the root file system.

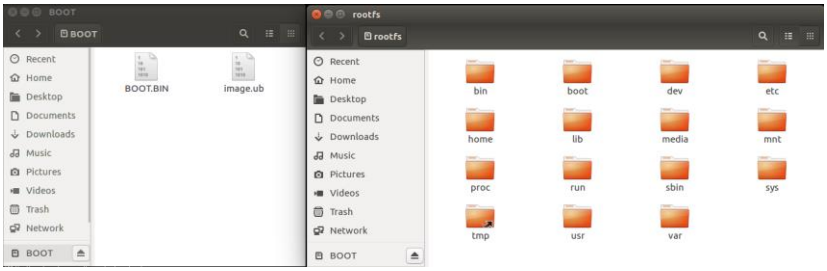


Figure 4-3: BOOT and rootfs partitions of the boot SD card.

4. Boot in zcu104 Board

Using Minicom via JTAG, the host PC can access to the development board and log in to the PetaLinux embedded OS. *Figure 4-4* show the log in and navigation through the root file system mounted.

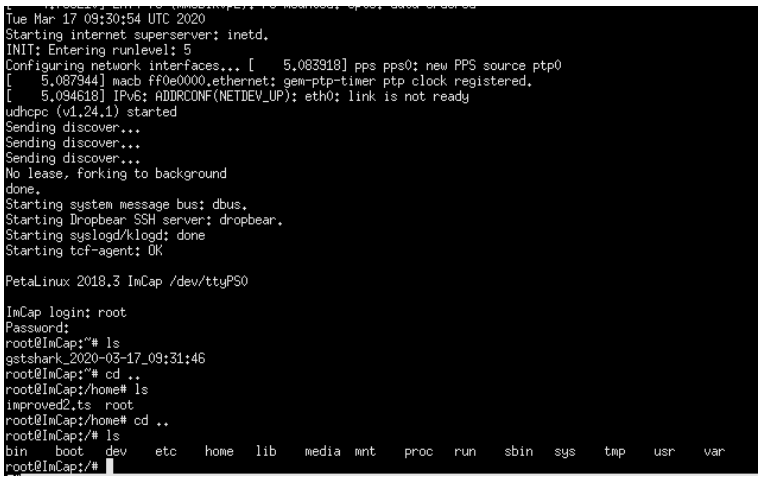


Figure 4-4: Terminal capture when using minicom to log in the embedded PetaLinux in the zcu104 Board

4.2. Image Capture

4.2.1. Image Displayed in Screen

Figure 4-5 shows the zcu104 powered and booted, connected to the host PC via JTAG, and with the USB camera and display port monitor connected. It is displaying a video stream using the GStreamer function explained in section *3.3. Capture, display and store video image from USB camera*.

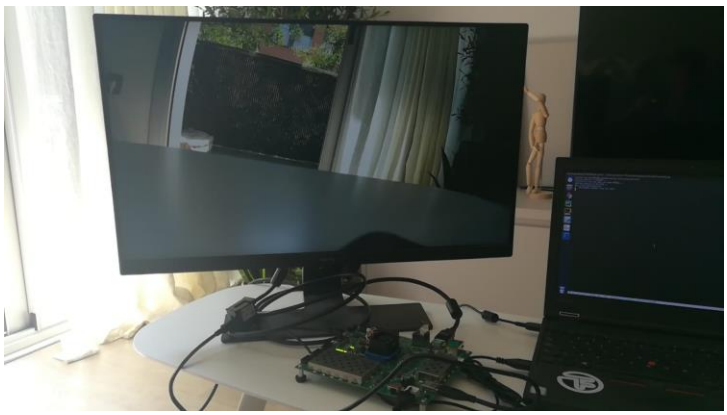


Figure 4-5: foto taken when the USB camera is capturing image of the balcony and displaying in the monitor.

4.2.2. Captured Video

Figure 4-6 shows a collage mounted with frames of the captured video using the GStreamer pipeline explained in section *3.3. Capture, display and store video image from USB camera*.

Figure 4-6: Frames from the captured video.

4.3. CNN capture in host PC

4.3.1. Single frame

Figure 4-7 shows an input image to the CNN and its corresponding output to verify the right performance of the SegNet in the host.

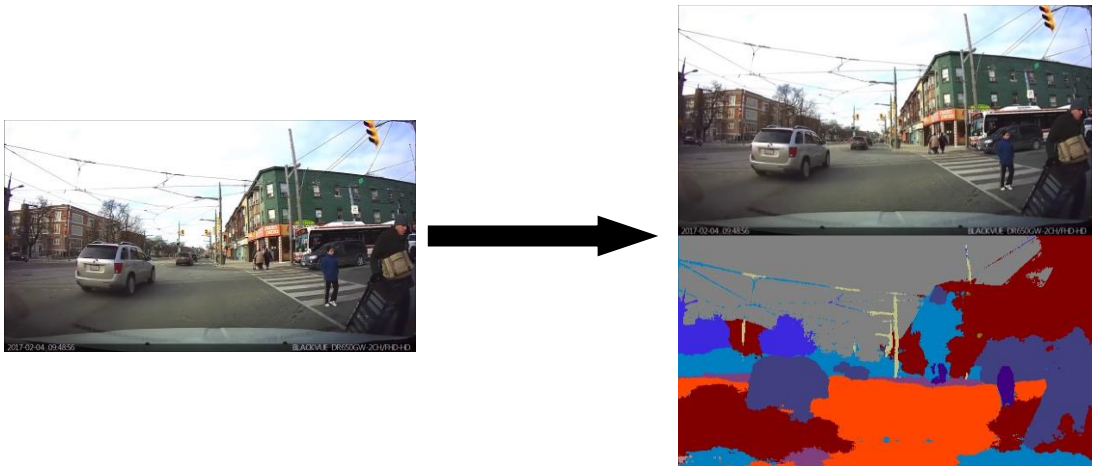


Figure 4-7: Input-output of a single image to the SegNet.

4.3.2. Sample Video

In *Figure 4-8* there is a sequence of frames from input and output sample video of the SegNet in host PC.

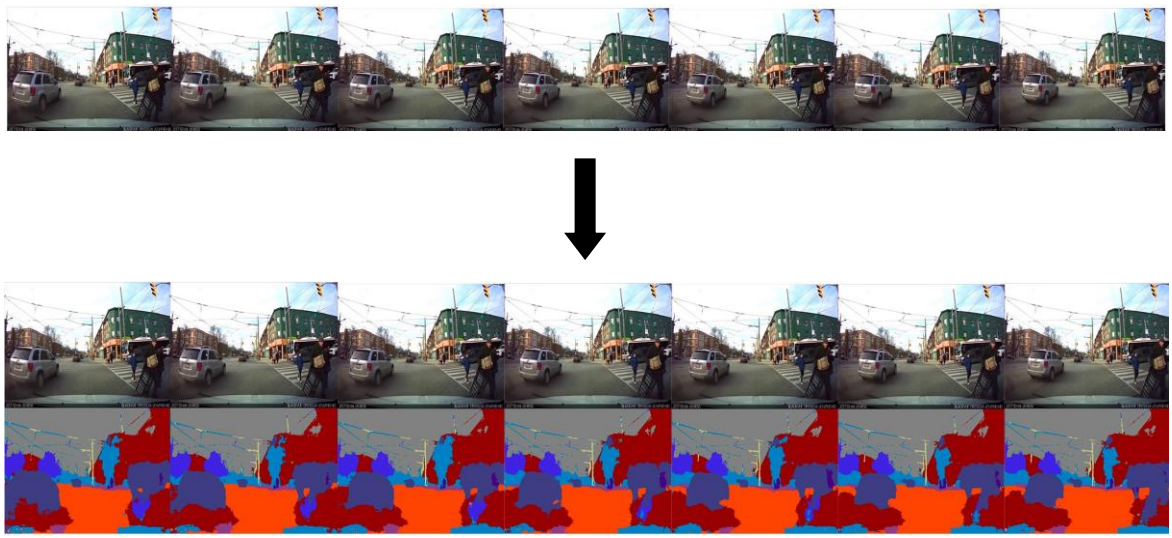


Figure 4-8: Input and output video of SegNet in host PC. Sample Video.

4.3.3. Captured Video

In *Figure 4-9* there is a sequence of frames from input and output captured video of the SegNet in host PC.

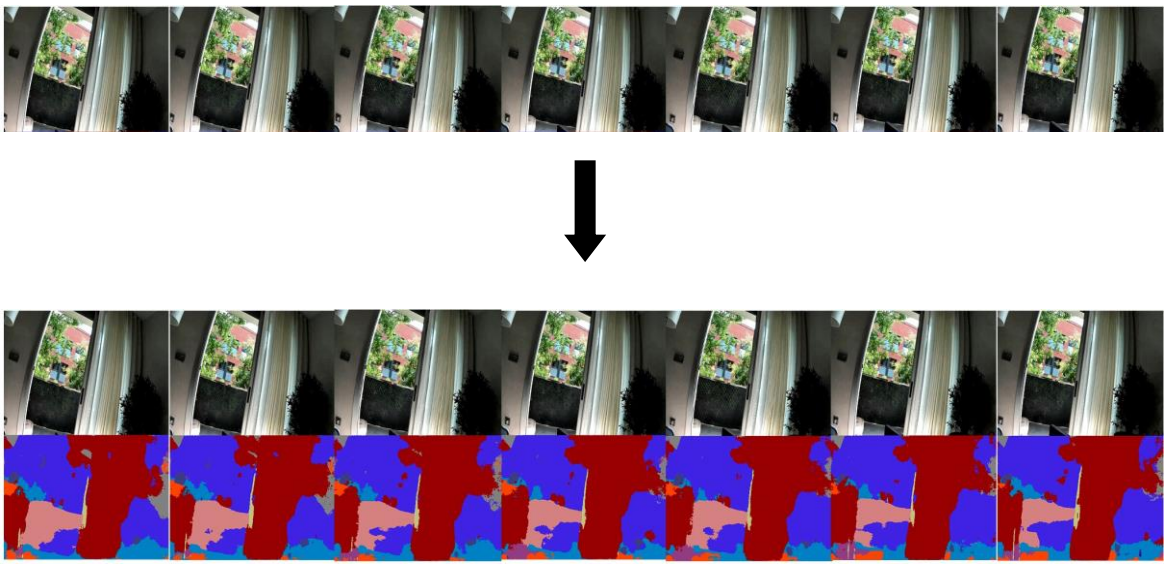


Figure 4-9: Input and output video of SegNet in host PC. Captured Video.

4.4. Final PL Vivado Block Design reports

4.4.1. Utilization Report

Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.

| Tool Version: Vivado v.2018.3.1 (lin64) Build 2489853 Tue Mar 26 04:18:30 MDT 2019
| Date: Sun Oct 3 15:13:07 2021
| Host: vision-ThinkPad-L560 running 64-bit Ubuntu 16.04.7 LTS
| Command: report_utilization -file
/home/vision/Documents/TFM/workspace/Hardware/ERROR/xilinx-zcu104-
2018.3_Hardware/xilinx-zcu104-2018.3.runs/impl_1/utilization.txt -name utilization_1
| Design: design_1_wrapper
| Device: xczu7evffvc1156-2
| Design State: Fully Placed

Utilization Design Information

Table of Contents
---
1. CLB Logic
1.1 Summary of Registers by Type
2. CLB Logic Distribution

3. BLOCKRAM
4. ARITHMETIC
5. I/O
6. CLOCK
7. ADVANCED
8. CONFIGURATION
9. Primitives
10. Black Boxes
11. Instantiated Netlists

1. CLB Logic

Site Type	Used	Fixed	Available	Util%
CLB LUTs	81860	0	230400	35.53
LUT as Logic	75352	0	230400	32.70
LUT as Memory	6508	0	101760	6.40
LUT as Distributed RAM	3130	0		
LUT as Shift Register	3378	0		
CLB Registers	136459	0	460800	29.61
Register as Flip Flop	136458	0	460800	29.61
Register as Latch	0	0	460800	0.00
Register as AND/OR	1	0	460800	<0.01
CARRY8	1759	0	28800	6.11
F7 Muxes	3107	0	115200	2.70
F8 Muxes	531	0	57600	0.92
F9 Muxes	0	0	28800	0.00

1.1 Summary of Registers by Type

Total	Clock Enable	Synchronous	Asynchronous
1	_	-	-
0	_	-	Set
0	_	-	Reset
0	_	Set	-
0	_	Reset	-
0	Yes	-	-
625	Yes	-	Set
890	Yes	-	Reset
2437	Yes	Set	-
132506	Yes	Reset	-

2. CLB Logic Distribution

Site Type	Used	Fixed	Available	Util%
CLB	18418	0	28800	63.95
CLBL	9747	0		
CLBM	8671	0		
LUT as Logic	75352	0	230400	32.70
using O5 output only	1404			
using O6 output only	54140			
using O5 and O6	19808			

LUT as Memory	6508	0	101760	6.40
LUT as Distributed RAM	3130	0		
using O5 output only	0			
using O6 output only	498			
using O5 and O6	2632			
LUT as Shift Register	3378	0		
using O5 output only	2			
using O6 output only	3101			
using O5 and O6	275			
CLB Registers	136459	0	460800	29.61
Register driven from within the CLB	65689			
Register driven from outside the CLB	70770			
LUT in front of the register is unused	54349			
LUT in front of the register is used	16421			
Unique Control Sets	4340		57600	7.53

* Note: Available Control Sets calculated as CLB Registers / 8, Review the Control Sets Report for more information regarding control sets.

3. BLOCKRAM

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	302	0	312	96.79
RAMB36/FIFO*	279	0	312	89.42
FIFO36E2 only	5			
RAMB36E2 only	274			
RAMB18	46	0	624	7.37
FIFO18E2 only	5			
RAMB18E2 only	41			
URAM	50	0	96	52.08

* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E2 or one FIFO18E2. However, if a FIFO18E2 occupies a Block RAM Tile, that tile can still accommodate a RAMB18E2

4. ARITHMETIC

Site Type	Used	Fixed	Available	Util%
DSPs	426	0	1728	24.65
DSP48E2 only	426			

5. I/O

Site Type	Used	Fixed	Available	Util%
Bonded IOB	118	118	360	32.78
HPIOB_M	63	63	144	43.75
INPUT	1			
OUTPUT	14			
BIDIR	48			
HPIOB_S	54	54	144	37.50
INPUT	1			
OUTPUT	13			

BIDIR	40			
HDIOB_M	0	0	24	0.00
HDIOB_S	0	0	24	0.00
HPIOB_SNGL	1	1	24	4.17
INPUT	0			
OUTPUT	1			
BIDIR	0			
HPIOBDIFFINBUF	9	9	192	4.69
DIFFINBUF	9	9		
HPIOBDIFFOUTBUF	0	0	192	0.00
HDIOBDIFFINBUF	0	0	48	0.00
BITSLICE_CONTROL	22	0	64	34.38
BITSLICE_RX_TX	106	106	416	25.48
RXTX_BITSLICE	106	106		
BITSLICE_TX	22	0	64	34.38
RIU_OR	11	0	32	34.38

6. CLOCK

Site Type	Used	Fixed	Available	Util%
GLOBAL CLOCK BUFFERs	16	0	544	2.94
BUFGCE	15	0	208	7.21
BUFGCE_DIV	0	0	32	0.00
BUFG_GT	0	0	144	0.00
BUFG_PS	1	0	96	1.04
BUFGCTRL*	0	0	64	0.00
PLL	3	0	16	18.75
MMCM	3	1	8	37.50

* Note: Each used BUFGCTRL counts as two global buffer resources. This table does not include global clocking resources, only buffer cell usage. See the Clock Utilization Report (report_clock_utilization) for detailed accounting of global clocking resource availability.

7. ADVANCED

Site Type	Used	Fixed	Available	Util%
GTHE4_CHANNEL	0	0	20	0.00
GTHE4_COMMON	0	0	5	0.00
OBUFDS_GTE4	0	0	10	0.00
OBUFDS_GTE4_ADV	0	0	10	0.00
PCIE40E4	0	0	2	0.00
PS8	1	0	1	100.00
SYSMONE4	0	0	1	0.00
VCU	1	0	1	100.00

8. CONFIGURATION

Site Type	Used	Fixed	Available	Util%
BSCANE2	1	0	4	25.00
DNA_PORTE2	0	0	1	0.00
EFUSE_USR	0	0	1	0.00

FRAME_ECCE4	0	0	1	0.00
ICAPE3	0	0	2	0.00
MASTER_JTAG	0	0	1	0.00
STARTUPE3	0	0	1	0.00

9. Primitives

Ref Name	Used	Functional Category
FDRE	132506	Register
LUT6	29735	CLB
LUT3	28359	CLB
LUT5	14033	CLB
LUT4	11656	CLB
LUT2	9415	CLB
RAMD32	4562	CLB
MUXF7	3107	CLB
SRL16E	3077	CLB
FDSE	2437	Register
LUT1	1962	CLB
CARRY8	1759	CLB
FDCE	890	Register
RAMS32	704	CLB
FDPE	625	Register
SRLC32E	576	CLB
MUXF8	531	CLB
RAMD64E	496	CLB
DSP48E2	426	Arithmetic
RAMB36E2	274	Block Ram
RXTX_BITSLICE	106	I/O
IBUFCTRL	81	Others
OBUFT_DCIEN	72	I/O
INBUF	72	I/O
URAM288	50	Block Ram
RAMB18E2	41	Block Ram
OBUF	28	I/O
TX_BITSLICE_TRI	22	I/O
BITSLICE_CONTROL	22	I/O
OBUFT	16	I/O
BUFGCE	15	Clock
RIU_OR	11	I/O
INV	9	CLB
DIFFINBUF	9	I/O
HPIO_VREF	8	I/O
FIFO36E2	5	Block Ram
FIFO18E2	5	Block Ram
PLLE4_ADV	3	Clock
MMCME4_ADV	3	Clock
VCU	1	Advanced
PS8	1	Advanced
BUFG_PS	1	Clock
BSCANE2	1	Configuration
AND2B1L	1	Others

10. Black Boxes

Ref Name	Used
design_1_vcu_ddr4_controller_0_0__RateMeasurement__parameterized0__7	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement__parameterized0__6	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement__parameterized0__5	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement__parameterized0__4	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement__parameterized0__3	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement__parameterized0__2	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement__parameterized0__1	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement__parameterized0	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement__55	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement__54	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement__53	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement__52	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement__51	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement__50	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement__49	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_482	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_481	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_480	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_470	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_469	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_468	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_461	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_460	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_459	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_449	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_448	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_447	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_336	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_335	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_334	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_324	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_323	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_322	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_315	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_314	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_313	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_303	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_302	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_301	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2190	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2189	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2188	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2178	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2177	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2176	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2169	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2168	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2167	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2157	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2156	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2155	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2060	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2059	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2058	1

design_1_vcu_ddr4_controller_0_0__RateMeasurement_2048	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2047	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2046	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2039	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2038	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2037	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2027	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2026	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement_2025	1
design_1_vcu_ddr4_controller_0_0__RateMeasurement	1

11. Instantiated Netlists

Ref Name	Used
vcu_ddr4_controller_v1_0_0_ddr4_0_phy	1
design_1_zynq_ultra_ps_e_0_0	1
design_1_xbar_5	1
design_1_xbar_4	1
design_1_xbar_3	1
design_1_xbar_2	1
design_1_xbar_1	1
design_1_xbar_0	1
design_1_vcu_ddr4_controller_0_0	1
design_1_vcu_0_0	1
design_1_v_frmbuf_wr_0_0	1
design_1_v_frmbuf_rd_0_0	1
design_1_s01_regslice_0	1
design_1_s01_mmu_0	1
design_1_s00_regslice_7	1
design_1_s00_regslice_6	1
design_1_s00_regslice_5	1
design_1_s00_regslice_4	1
design_1_s00_regslice_3	1
design_1_s00_regslice_2	1
design_1_s00_regslice_1	1
design_1_s00_regslice_0	1
design_1_s00_mmu_4	1
design_1_s00_mmu_3	1
design_1_s00_mmu_2	1
design_1_s00_mmu_1	1
design_1_s00_mmu_0	1
design_1_proc_sys_reset_4_0	1
design_1_proc_sys_reset_3_0	1
design_1_proc_sys_reset_2_0	1
design_1_proc_sys_reset_1_0	1
design_1_proc_sys_reset_0_0	1
design_1_m01_regslice_2	1
design_1_m01_regslice_1	1
design_1_m01_regslice_0	1
design_1_m00_regslice_3	1
design_1_m00_regslice_2	1
design_1_m00_regslice_1	1
design_1_m00_regslice_0	1
design_1_dpu_eu_0_0	1
design_1_clk_wiz_1_0	1
design_1_clk_wiz_0_0	1

design_1_auto_us_0	1
design_1_auto_rs_w_0	1
design_1_auto_pc_0	1
design_1_auto_ds_0	1
design_1_auto_cc_6	1
design_1_auto_cc_5	1
design_1_auto_cc_4	1
design_1_auto_cc_3	1
design_1_auto_cc_2	1
design_1_auto_cc_1	1
design_1_auto_cc_0	1
dbg_hub	1

4.4.2. Timing Report

Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.

| Tool Version: Vivado v.2018.3.1 (lin64) Build 2489853 Tue Mar 26 04:18:30 MDT 2019
| Date: Wed Sep 29 19:56:27 2021
| Host: vision-ThinkPad-L560 running 64-bit unknown
|Command: report_timing_summary -max_paths 10 -file design_1_wrapper_timing_summary_routed.rpt -pb design_1_wrapper_timing_summary_routed.pb -rpx design_1_wrapper_timing_summary_routed.rpx -warn_on_violation
| Design: design_1_wrapper
| Device: xczu7ev-ffvc1156
| Speed File: -2 PRODUCTION 1.23 10-29-2018
| Temperature Grade: E

Timing Summary Report

| Timer Settings

Enable Multi Corner Analysis:	Yes
Enable Pessimism Removal:	Yes
Pessimism Removal Resolution:	Nearest Common Node
Enable Input Delay Default Clock:	No
Enable Preset / Clear Arcs:	No
Disable Flight Delays:	No
Ignore I/O Paths:	No
Timing Early Launch at Borrowing Latches:	false

Corner Name	Analyze Max Paths	Analyze Min Paths
Slow	Yes	Yes
Fast	Yes	Yes

check_timing report

Table of Contents

1. checking no_clock

2. checking constant_clock
3. checking pulse_width_clock
4. checking unconstrained_internal_endpoints
5. checking no_input_delay
6. checking no_output_delay
7. checking multiple_clock
8. checking generated_clocks
9. checking loops
10. checking partial_input_delay
11. checking partial_output_delay
12. checking latch_loops

1. checking no_clock

There are 0 register/latch pins with no clock.

2. checking constant_clock

There are 0 register/latch pins with constant_clock.

3. checking pulse_width_clock

There are 0 register/latch pins which need pulse_width check

4. checking unconstrained_internal_endpoints

There are 0 pins that are not constrained for maximum delay.
There are 0 pins that are not constrained for maximum delay due to constant clock.

5. checking no_input_delay

There are 0 input ports with no input delay specified.
There are 0 input ports with no input delay but user has a false path constraint.

6. checking no_output_delay

There is 1 port with no output delay specified. (HIGH)
There are 0 ports with no output delay but user has a false path constraint
There are 0 ports with no output delay but with a timing clock defined on it or propagating through it

7. checking multiple_clock

There are 0 register/latch pins with multiple clocks.

8. checking generated_clocks

There are 0 generated clocks that are not connected to a clock source.

9. checking loops

There are 0 combinational loops in the design.

10. checking partial_input_delay

There are 0 input ports with partial input delay specified.

11. checking partial_output_delay

There are 0 ports with partial output delay specified.

12. checking latch_loops

There are 0 combinational latch loops in the design through latch input

---
| Design Timing Summary
| ---

WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints	WHS(ns)	THS(ns)
0.079	0.000	0	395384	0.010	0.000

THS Failing Endpoints	THS Total Endpoints	WPWS(ns)	TPWS(ns)	TPWS Failing Endpoints
0	394310	0.011	0.000	0

TPWS Total Endpoints
147605

All user specified timing constraints are met.

| Clock Summary
| ---

Clock	Waveform(ns)	Period(ns)	Frequency(MHz)
clk_pl_0	{0.000 5.000}	10.000	100.000
dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/INTERNAL_TCK	{0.000 25.000}	50.000	20.000
design_1_i/clk_wiz_0/inst/clk_in1	{0.000 5.000}	10.000	100.000
clk_out1_design_1_clk_wiz_0_0	{0.000 15.000}	30.000	33.333
clk_out2_design_1_clk_wiz_0_0	{0.000 1.513}	3.025	330.556
clk_out3_design_1_clk_wiz_0_0	{0.000 5.042}	10.084	99.167
design_1_i/clk_wiz_1/inst/clk_in1	{0.000 5.000}	10.000	100.000
axi_aclk_design_1_clk_wiz_1_0_1	{0.000 5.000}	10.000	100.000
axi_dpu_aclk_design_1_clk_wiz_1_0_1	{0.000 2.500}	5.000	200.000
dpu_2x_clk_design_1_clk_wiz_1_0_1	{0.000 1.250}	2.500	400.000
mig_sys_clk_p[0]	{0.000 1.668}	3.335	299.850
mmcm_clkout0	{0.000 1.876}	3.752	266.533
pll_clk[0]	{0.000 0.234}	0.469	2132.267
pll_clk[0]_DIV	{0.000 1.876}	3.752	266.533
pll_clk[1]	{0.000 0.234}	0.469	2132.267
pll_clk[1]_DIV	{0.000 1.876}	3.752	266.533
pll_clk[2]	{0.000 0.234}	0.469	2132.267
pll_clk[2]_DIV	{0.000 1.876}	3.752	266.533
mmcm_clkout2	{0.000 1.876}	3.752	266.533
mmcm_clkout5	{0.000 7.504}	15.008	66.633
mmcm_clkout6	{0.000 3.752}	7.504	133.267

4.5. Full system Implementation

4.5.1. PetaLinux built

Figure 4-10 shows a capture of the terminal with the successful build of the full PetaLinux project. As in the previous results, there is a failure referred to TFTP because it is not used. Also, there are two warnings that can be ignored.

```
[INFO] building project
[INFO] sourcing bitbake
INFO: bitbake petalinux-user-image
Loading cache: 100% |################################################################################################################
Loaded 3462 entries from dependency cache.
Parsing recipes: 100% |##############################################################################################################
Parsing of 2571 .bb files complete (2530 cached, 41 parsed). 3463 targets, 137 skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100% |###########################################################################################################
Checking sstate mirror object availability: 100% |###################################################################################
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
WARNING: dnndk-1.0-r0 do_package_qa: QA Issue: dnndk: found library in wrong location: /usr/local/lib/libn2cube.so [libdir]
WARNING: petalinux-user-image-1.0-r0 do_rootfs: [log_check] petalinux-user-image: found 1 warning message in the logfile:
[log_check] warning: %post(sysvinit-inittab-2.88dsf-r10.zcu104_zynqmp) scriptlet failed, exit status 1

NOTE: Tasks Summary: Attempted 6558 tasks of which 6191 didn't need to be rerun and all succeeded.

Summary: There were 2 WARNING messages shown.
INFO: Copying Images from deploy to images
NOTE: Failed to copy built images to tftp dir:  /tftpboot
[INFO] successfully built project
```

Figure 4-10: Screen captured showing success in building the project.

Then, in *Figure 4-11* there is a screen capture with the messages generated at rebuilding the project with the sdk option.

```
(venv) vision@vision-ThinkPad-L560:~/Documents/TFM/workspace/Petalinux/Training$ petalinux-build --sdk
[INFO] building project
[INFO] sourcing bitbake
INFO: bitbake petalinux-user-image -c do_populate_sdk
Loading cache: 100% |################################################################################################################
Loaded 3462 entries from dependency cache.
Parsing recipes: 100% |##############################################################################################################
Parsing of 2571 .bb files complete (2537 cached, 34 parsed). 3463 targets, 137 skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100% |###########################################################################################################
Checking sstate mirror object availability: 100% |###################################################################################
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
NOTE: Tasks Summary: Attempted 5459 tasks of which 5458 didn't need to be rerun and all succeeded.
[INFO] Copying SDK Installer...
[INFO] successfully built project
```

Figure 4-11: Screen captured showing success in building the project with sdk.

Finally, *Figure 4-12* shows the success in packaging the roofs to be opened with XSDK.

```
PetaLinux SDK installer version 2018.3
======================================
You are about to install the SDK to "/home/vision/Documents/TFM/workspace/Petalinux/sdk". Proceed[Y/n]? Y
Extracting SDK.....................................................................................................done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the environment setup script e.g.
 $ . /home/vision/Documents/TFM/workspace/Petalinux/sdk/environment-setup-aarch64-xilinx-linux
```

Figure 4-12: Screen captured showing success in packaging the rootfs for XSDK.

4.5.2. DPU utilities available

Figure 4-13 shows that the required utilities to run the model in the DPU IP are properly placed in their corresponding directories.

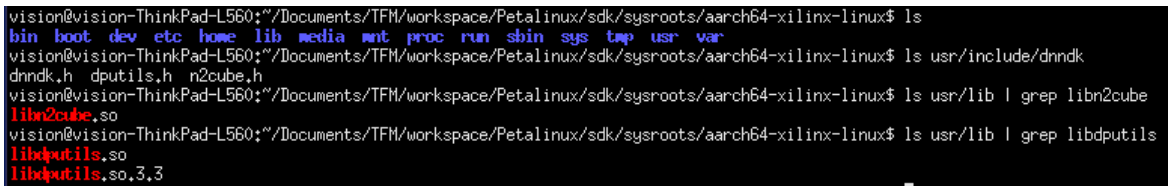


Figure 4-13: Screen capture showing the DPU utilities in their folder.

4.5.3. DNNDK SegNet error

When trying to quantize the SegNet model, an error shown in *Figure 4-14* occurs.

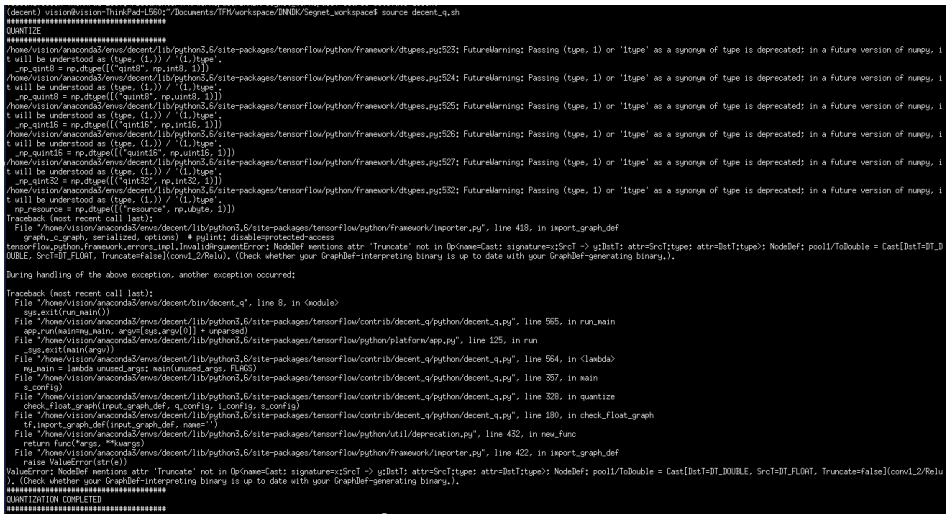


Figure 4-14: Screen captured when truing to quantize the model with DNNDK.

At first, this error seems to refer to some data type misunderstanding. After consulting with the Xilinx developer, navigating through lots of forums and trying many workarounds, I conclude that the error root case is due to a mismatch in the TensorFlow version supported by the DNNDK and the one used to train the model. SegNet is working with version 1.11.0 while DNNDK only works with version 1.9. It is a hard issue because changing the DNNDK version may lead to a set of incompatibilities between all the Xillinx tools, libraries versions, Ubuntu version and maybe can force to use VITIS. Otherwise, it could imply to train the net with the right TensorFlow version or changing the net. When asking the Xilinx developer, it tells that all the steps and configurations are correctly

followed but they do not guarantee that these tools may work with any other Net but the ones in their examples. In addition, the advertising, they make about their tools is that they are robust tools able to implement any net trained in caffe, tensorflow, pytorch… but the reality is that a small deviation from its example project with its recipe will result in errors that they are not able to solve or understand due to the complexity of the operations that they have tried to simplify in a short sequence of instructions.

5. Budget

Hardware			
Concept	**Units**	**Unit price [€]**	**Total price [€]**
ZCU104 evaluation board + e-con_See3CAM_CU30	1	1.586,81	1.586,81
Display Port monitor	1	130	130
PC (amortization)	1	50	50
			1.766,81

Work hours		
Units [h]	**Unit price [€]**	**Total price [€]**
750	45	33.750
		33.750

Energy consumption		
Units [h]	**Unit price [€/kWh]**	**Total price [€]**
750	0,14	105
		105

Total cost of the project	**35.621,81 €**

6. Conclusions and future development

As mentioned in the introduction, the development of this thesis is divided in two implementations. The first implementation has been successfully performed and the video images have been processed with the SegNet in the host PC. Unfortunately, the second implementation fails when trying to quantize and compile the Neural Network to introduce it in the DPU IP.

This thesis has completed successfully the steps regarding the first implementation and some of the second implementation:

1. Configuring and packaging an embedded operating system including customizing the kernel, root file system and device tree and add applications, libraries and modules to the system.
2. Customizing and building a block design using the needed IP to capture video, encode it with the VCU, store a video file and use the DPU IP along with display the results in a display port external monitor.
3. Use video capture libraries such GStreamer to build pipelines from the terminal and configure each element to get a video stream from USB camera to external monitor and to directly store a video file from the USB camera.
4. Understand and execute the CNN SegNet to process the captured video in the host PC adapting the original code to the desired features.

Also, even if the implementation fails at trying to compile the neural network into the DPU IP, the research done regarding this and the next steps has been realized and they may work if the incompatibility error does not appear. The DECENT quantization script and the DNNC compilation script are made based on the trainings and the example codes provided by Xilinx and they have been checked by the Xilinx developer and they should be correct. After that, the configuration of the XSDK is also checked by the developer and the later steps are just compile, boot and execute. Thus, even though the practical implementation has been failed, the scope of the project regarding the evaluation of the complexity to implement the NN in an FPGA instead of GPU is completed and useful conclusions can be extracted. Also, the first implementation is completed so the objective of processing images captured with an embedded system in a neural network is achieved.

As a conclusion to the availability to implement a Neural Network in an FPGA instead of a GPU, even if the only drawback for the FPGA is the complexity of the configuration process, nowadays it is still better to use GPU. In this project only Xilinx alternative has been explored. In that specific case, they say that this implementation works with most of the frameworks, with most of models and customizing the applications. In reality, the single issue to discriminate between tools and versions of each tool is a hard task and requires a significant research time. In addition to that, it is not trivial which model can be used in which version. When asking Xilinx, they only provide tutorials and project examples, but not a clear explanation of what is happening and how everything is working. This may make easier to work without the need of understand everything, but it is only useful if the specifications of your project are the same than in the training. If not, unexpected errors may occur and there is not a clear explanation of how to solve it because it seems that there is not a clear understanding of how it is working. In the order

hand, there are a lot of development boards that can be used as an alternative to easily adapt the HW to the needs of your model using a GPU with a good performance.

Regarding future implementations, there is a new software called VITIS that could be a better choice for this implementation. It is intended to be a merge of all the tools used in this thesis in a single software and it may provide the possibility to program the application directly in python. This alternative has been discarded in this thesis because it is very recent and the needed workflow and requirements are even more confusing.

Bibliography

Tools installation:

[1] UG973 (v2018.3) “Vivado Design Suite User Guide.” *Release Notes, Installation, and Licening.*

[2] UG1144 (v2018.3) “Petalinux Tools Documentation.” *Reference Guide*.

[3] UG1294 (v2018.3) “SDSoC Development Environment.” *Release Notes, Installation, and Licensing Guide.*

Project Examples:

[4] *xilinx-zcu104-v2018.3-final-v2.bsp*

[5] zcu104-rv-ss-2018-3.zip

[6] zcu102-dpu-trd-2018-2-190531.zip and rdf0428-zcu106-vcu-trd-2018-3.zip

Hardware Support:

[7] UG1267 (v1.1) “ZCU104 Evaluation Board.” *User Guide.*

[8] UG1085 (v2.1) “Zynq UltraScale+ Device.” *Technical Reference Manual.*

[9] e-con_See3CAM_CU30_CHL_TC_BX. *Getting Started Manual.*

[10] e-con_See3CAM_CU30_CHL_TC_BX. *Datasheet.*

[11] e-con_See3CAM_CU30_CHL_TC_BX. *Product Datasheet*

[12] https://www.e-consystems.com/ar0330-lowlight-usb-cameraboard.asp

[13] https://www.xilinx.com/products/intellectual-property/dpu.html

Linux Image Building:

[14] UG1157 (v2018.3) “PetaLinux Tools Documentation.” *PetaLinux Command Line Reference.*

[15] UG1209 (v2018.3) “Zynq UltraScale+ MPSoC: Embedded Design Tutorial.” *A Hands-On Guide to Effective Embedded System Design*

[16] UG1283 (v2018.2) *Bootgen User Guide*

[17] UG1156 (v2017.3) “PetaLinux Tools Documentation.” *Workflow tutorial*

[18] https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841937/Zynq+UltraScale+MPSoC+Ubuntu+part+2+-+Building+and+Running+the+Ubuntu+Desktop+From+Sources

[19] https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841724/PMU+Firmware

[20] https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842107/Arm+Trusted+Firmware

[21] https://www.yoctoproject.org

[22] https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841883/Yocto

[23] https://forums.xilinx.com/t5/Forums/ct-p/XlnxProd

[24] https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842385/How+to+format+SD+card+for+SD+boot

Xilinx Trainings:

[25] https://www.xilinx.com/training/customer-training/developing-xilinx-ai-solutions-edge-applications.html

Image Capture:

[26] PG252 “H.264/H265 Video Codec Unit v1.2.” *LogiCORE IP Product Guide.*

[27] UG1085 “Zynq UltraScale+ Device.” *Technical Reference Manual*

[28] UG1228 (v1.0) *Zynq UltraScale+ MPSoC Embedded Design Methodology Guide*

[29] https://github.com/Xilinx/meta-petalinux/tree/master/recipes-multimedia/gstreamer/gstreamer-vcu-examples

[30] https://www.mankier.com/1/v4l2-ctl

[31] https://gstreamer.freedesktop.org/documentation/tutorials/basic/index.html?gi-language=c

[32]https://developer.ridgerun.com/wiki/index.php?title=Zynq_Ultrascale%2B_Capture_settings_and_Gstreamer_pipelines

NN in host:

[33] https://github.com/PINTO0309/SegNet-TF

[34] https://www.tensorflow.org/install/pip

[35] https://www.ffmpeg.org/

[36] https://www.jetbrains.com/es-es/pycharm/

[37] https://www.learnopencv.com/read-write-and-display-a-video-using-opencv-cpp-python/

[38] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[40] https://paperswithcode.com/paper/segnet-a-deep-convolutional-encoder-decoder

Tensorflow in PS:

[41] https://docs.python.org/3/tutorial/venv.html

[42] https://docs.bazel.build/versions/master/install-ubuntu.html

[43] https://www.tensorflow.org/install/source

DPU + DNNDK:

[44]https://github.com/Xilinx/Embedded-Reference-Platforms-User-Guide/blob/master/README.md

[45] PG338 (v2.0) “DPU for Convolutional Neuronal Network v2.0.” *DPU IP Product Guide*

[46] UG1327 (v1.6) *DNNDK User Guide*

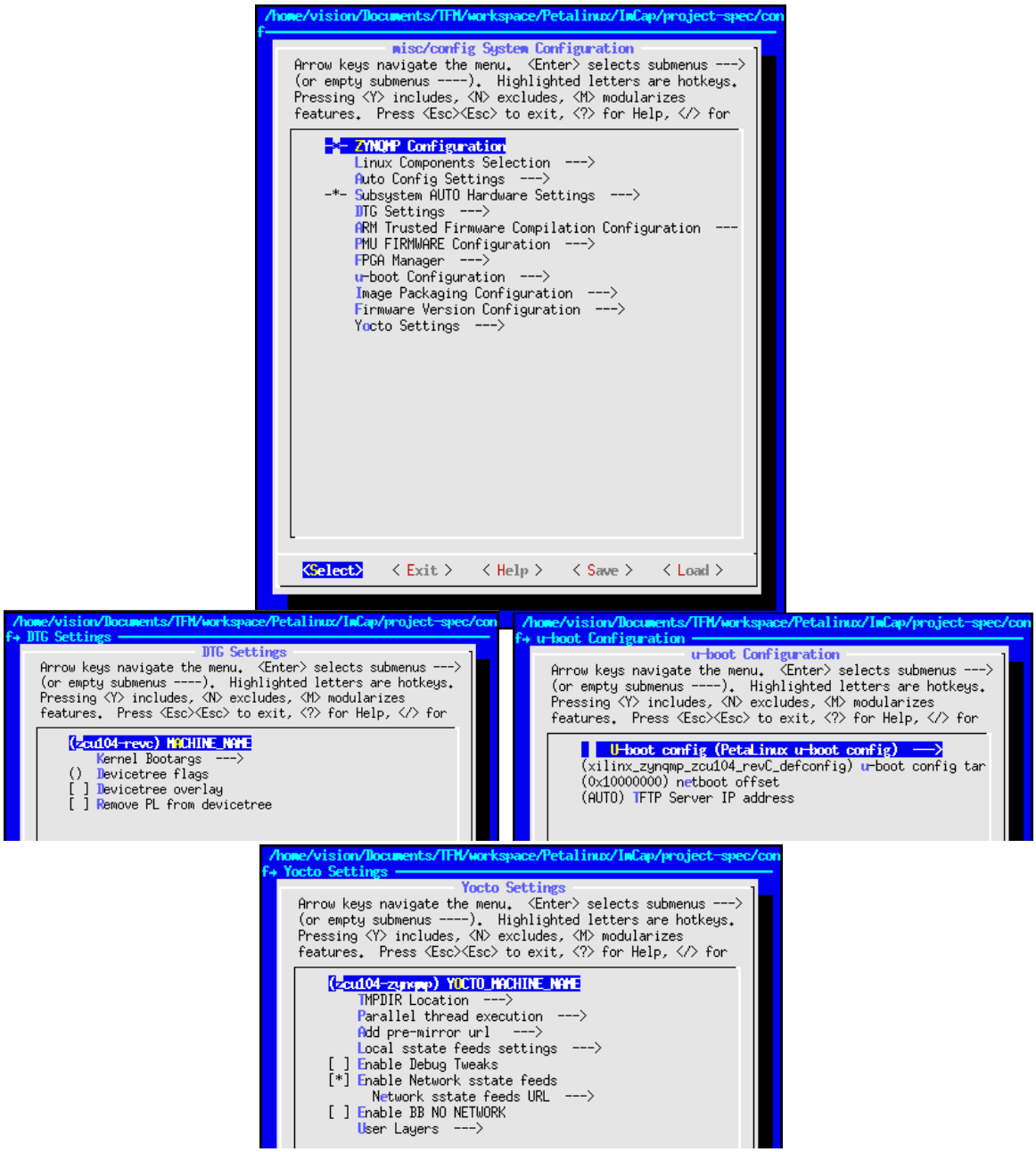
[47] UG1331 (v1.0) *DNNDK User Guide for the SDSoC Developement Environment*

[48] Avnet Technical Training Course. *Introduction to Deep Learning with Xilinx SoCs*

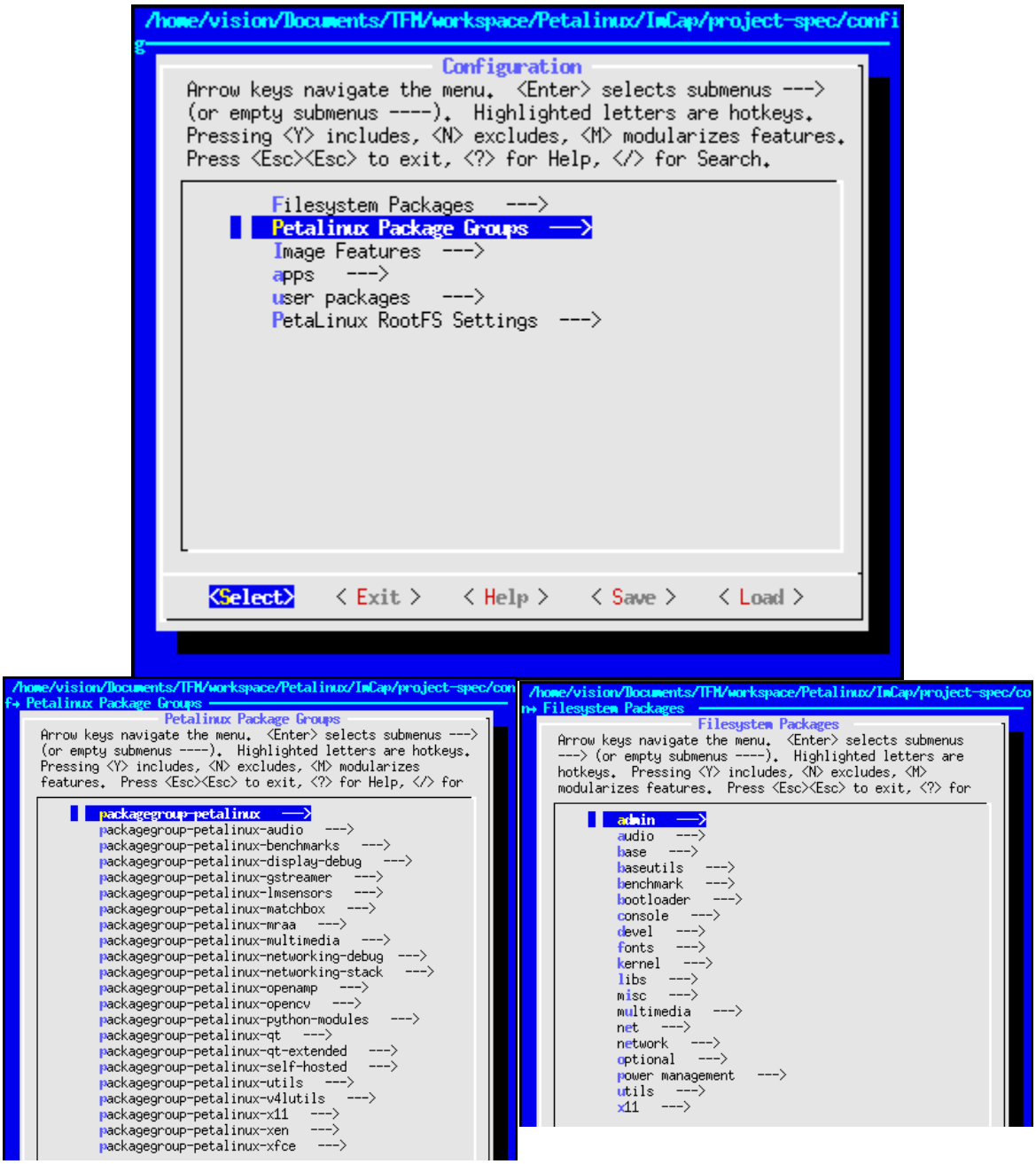
Appendices

APPENDIX A: PetaLinux Configuration

PetaLinux HW configuration window



Rootfs configuration Windows



APPENDIX B: VCU Demo files

vcu-demo-camera-encode-decode-display.sh

```
#!/bin/bash
#
# Get RAW YUV frames from Camera, encode it, decode it and display it
#
# Copyright (C) 2017 Xilinx
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
type vcu-demo-functions.sh > "/dev/null"
if [ $? -ne 0 ]; then
        echo "Copy vcu-demo-functions.sh to /usr/bin/ or append it's path to PATH variable
and re-run the script" && exit -1
fi

source vcu-demo-functions.sh

scriptName=`basename $0`
declare   -a   scriptArgs=("inputPath"   "videoSize"   "codecType"   "sinkName"   "numFrames"
"targetBitrate"     "showFps"     "audioType"     "internalEntropyBuffers"     "v4l2Device"
"displayDevice" "alsaSrc" "pulseSrc" "audioOutput" "alsaSink" "pulseSink" "frameRate")
declare -a checkEmpty=("codecType" "sinkName" "targetBitrate" "v4l2Device" "displayDevice"
"frameRate")


################################################################################
# Name:         usage
# Description: To display script's command line argument help
################################################################################
usage () {
        echo ' Usage : '$scriptName' -i <device_id_string> -v <video_capture_device> -s
<video_size> -c <codec_type> -a <audio_type> -o <sink_name> -n <number_of_frames> -b
<target_bitrate>     -e     <internal_entropy_buffers>     -r     <capture_device_rate>     -d
<display_device> -f --use-alsasrc --use-pulsesrc --audio-output <Audio output device> --
use-pulsesink --use-alsasink'
        DisplayUsage "${scriptArgs[@]}"
        echo '  Example :'
        echo '   '$scriptName''
        echo '   '$scriptName' -a aac'
        echo '   '$scriptName' --use-alsasrc -i "hw:1" -d "fd4a0000.zynqmp-display" -a aac'
        echo '   '$scriptName' -v "/dev/video1"'
        echo '   '$scriptName' -n 500 --use-alsasrc'
        echo '   '$scriptName' --use-alsasrc -i "hw:1" -n 500 --use-alsasrc -b 1200 -a aac'
        echo       '           '$scriptName'      --use-pulsesrc      -i      "alsa_input.usb-
046d_C922_Pro_Stream_Webcam_FCD7727F-02.analog-stereo" -n 500 -b 1200 -a aac'
        echo '   '$scriptName' -f'
        echo '   '$scriptName' -o fakevideosink'
        echo '   '$scriptName' --use-alsasrc -i "hw:1" -s 1920x1080 -c avc -a aac'
```

```
        echo '   '$scriptName' -s 1920x1080 -c avc -e 3'
        echo '   '$scriptName' -s 1280x720 -c avc'
        echo '   '$scriptName' --use-alsasrc -i "hw:1" -s 1280x720 -c avc -a aac'
        echo '   '$scriptName' --use-alsasrc -i "hw:1" -s 1280x720 -c avc -a vorbis'
        echo '   '$scriptName' -s 1280x720'
        echo '   '$scriptName' --use-alsasrc -i "hw:1" -s 1280x720 -c avc -a aac'
        echo '   '$scriptName' --use-alsasrc -i "hw:1" -s 1280x720 -c avc -a aac --audio-
output "hw:0"'
        echo '   '$scriptName' --use-pulsesrc -i "alsa_input.usb-
046d_C922_Pro_Stream_Webcam_FCD7727F-02.analog-stereo" -n 500 -b 1200 -a aac'
        echo '   "NOTE: This script depends on vcu-demo-settings.sh to be present in
/usr/bin or its path set in $PATH"'
        exit
}

##############################################################################
# Name:        CameraToDisplay
# Description: Get RAW data from camera, encode it, decode and display it
##############################################################################
CameraToDisplay() {
        if [ $SHOW_FPS ]; then
                SINK="fpsdisplaysink name=fpssink text-overlay=false video-
sink=\"$SINK_NAME\" sync=true -v"
        else
                SINK="$SINK_NAME"
        fi

         if [ $NUM_FRAMES ]; then
                 V4L2SRC="$V4L2SRC num-buffers=$NUM_FRAMES"
                 AUDIO_BUFFERS=$(($NUM_FRAMES*100/$FRAME_RATE))
         fi

        AUDIO_SRC_BASE="$AUDIO_SRC"
        AUDIO_SINK_BASE="$AUDIO_SINK"

        case $AUDIODEC_TYPE in
        "aac")
                AUDIODEC="faad"
                AUDIOENC="faac";;
        "vorbis")
                AUDIODEC="vorbisdec"
                AUDIOENC="vorbisenc";;
        *)
                if ! [ -z $AUDIODEC_TYPE ]; then
                        ErrorMsg "Invalid audio codec type specified, please specify either
vorbis or aac"
                fi
        esac

        IFS='x' read WIDTH HEIGHT <<< "$VIDEO_SIZE"
        CAMERA_CAPS="video/x-raw,width=$WIDTH,height=$HEIGHT,framerate=$FRAME_RATE/1"
        VIDEOCONVERT="videoconvert"
        VIDEOCONVERT_CAPS="video/x-raw, format=\(string\)NV12"
        if [ -z $SET_ENTROPY_BUF ]; then
                INTERNAL_ENTROPY_BUFFERS="6"
        fi

        OMXH264ENC="omxh264enc num-slices=8 control-rate="low-latency" target-
bitrate=$BIT_RATE prefetch-buffer=true"
        OMXH265ENC="omxh265enc num-slices=8 control-rate="low-latency" target-
bitrate=$BIT_RATE prefetch-buffer=true"
        OMXH264DEC="$OMXH264DEC internal-entropy-buffers=$INTERNAL_ENTROPY_BUFFERS latency-
mode="reduced-latency""
        OMXH265DEC="$OMXH265DEC internal-entropy-buffers=$INTERNAL_ENTROPY_BUFFERS latency-
mode="reduced-latency""

         case $CODEC_TYPE in
         "avc")
```

```
                PARSER=$H264PARSE
                ENCODER=$OMXH264ENC
                DECODER=$OMXH264DEC
                CAMERA_CAPS_ENC="video/x-
h264,width=$WIDTH,height=$HEIGHT,framerate=$FRAME_RATE/1";;
        "hevc")
                PARSER=$H265PARSE
                ENCODER=$OMXH265ENC
                DECODER=$OMXH265DEC
                CAMERA_CAPS_ENC="video/x-
h265,width=$WIDTH,height=$HEIGHT,framerate=$FRAME_RATE/1";;
        esac
        restartPulseAudio
        setAudioSrcProps

        if ! [ -z $AUDIO_OUTPUT ] && [ $AUDIO_SINK_BASE != "autoaudiosink" ]; then
                AUDIO_SINK="$AUDIO_SINK device=\"$AUDIO_OUTPUT\""
        fi

        if [ -z $AUDIODEC_TYPE ]; then
                        pipeline="$GST_LAUNCH $V4L2SRC ! $CAMERA_CAPS ! $VIDEOCONVERT !
$VIDEOCONVERT_CAPS ! $ENCODER ! $QUEUE ! $DECODER ! $QUEUE max-size-bytes=0 ! $SINK"
        else
                if [ "$AUDIO_SRC_BASE" == "pulsesrc" ] && [ "$AUDIO_SINK_BASE" ==
"pulsesink" ]; then
                        pipeline="$GST_LAUNCH $V4L2SRC ! $CAMERA_CAPS ! $VIDEOCONVERT !
$VIDEOCONVERT_CAPS ! $ENCODER ! $QUEUE ! $DECODER ! $QUEUE max-size-bytes=0 ! $SINK
$AUDIO_SRC ! $QUEUE ! $AUDIOENC ! $AUDIODEC ! $AUDIO_SINK"
                else
                        pipeline="$GST_LAUNCH $V4L2SRC ! $CAMERA_CAPS ! $VIDEOCONVERT !
$VIDEOCONVERT_CAPS ! $ENCODER ! $QUEUE ! $DECODER ! $QUEUE max-size-bytes=0 ! $SINK
$AUDIO_SRC ! $QUEUE ! $AUDIOCONVERT ! $AUDIOENC ! $QUEUE ! $AUDIODEC ! $AUDIOCONVERT !
$AUDIORESAMPLE ! $AUDIO_CAPS ! $AUDIO_SINK"
                fi
        fi

        runGstPipeline "$pipeline"
}

# Command Line Argument Parsing
args=$(getopt -o "i:v:d:s:c:o:a:b:n:e:r:fh" --long "input-path:,video-capture-
device:,display-device:,video-size:,audio-type:,codec-type:,sink-name:,num-frames:,bit-
rate:,internal-entropy-buffers:,audio-output:,frame-rate:,show-fps,help,use-alsasrc,use-
pulsesrc,use-alsasink,use-pulsesink" -- "$@")

[ $? -ne 0 ] && usage && exit -1

trap catchCTRL_C SIGINT
parseCommandLineArgs
checkforEmptyVar "${checkEmpty[@]}"
if [ -z $VIDEO_SIZE ]; then
        VIDEO_SIZE="640x480"
        echo "Video Size is not specified in args hence using 640x480 as default value"
fi

if [ -z $BIT_RATE ];then
        BIT_RATE=1000
fi
if ! [ -z $AUDIODEC_TYPE ]; then
         audioSetting
fi

RegSetting
DisableDPMS
CameraToDisplay
restoreContext
```

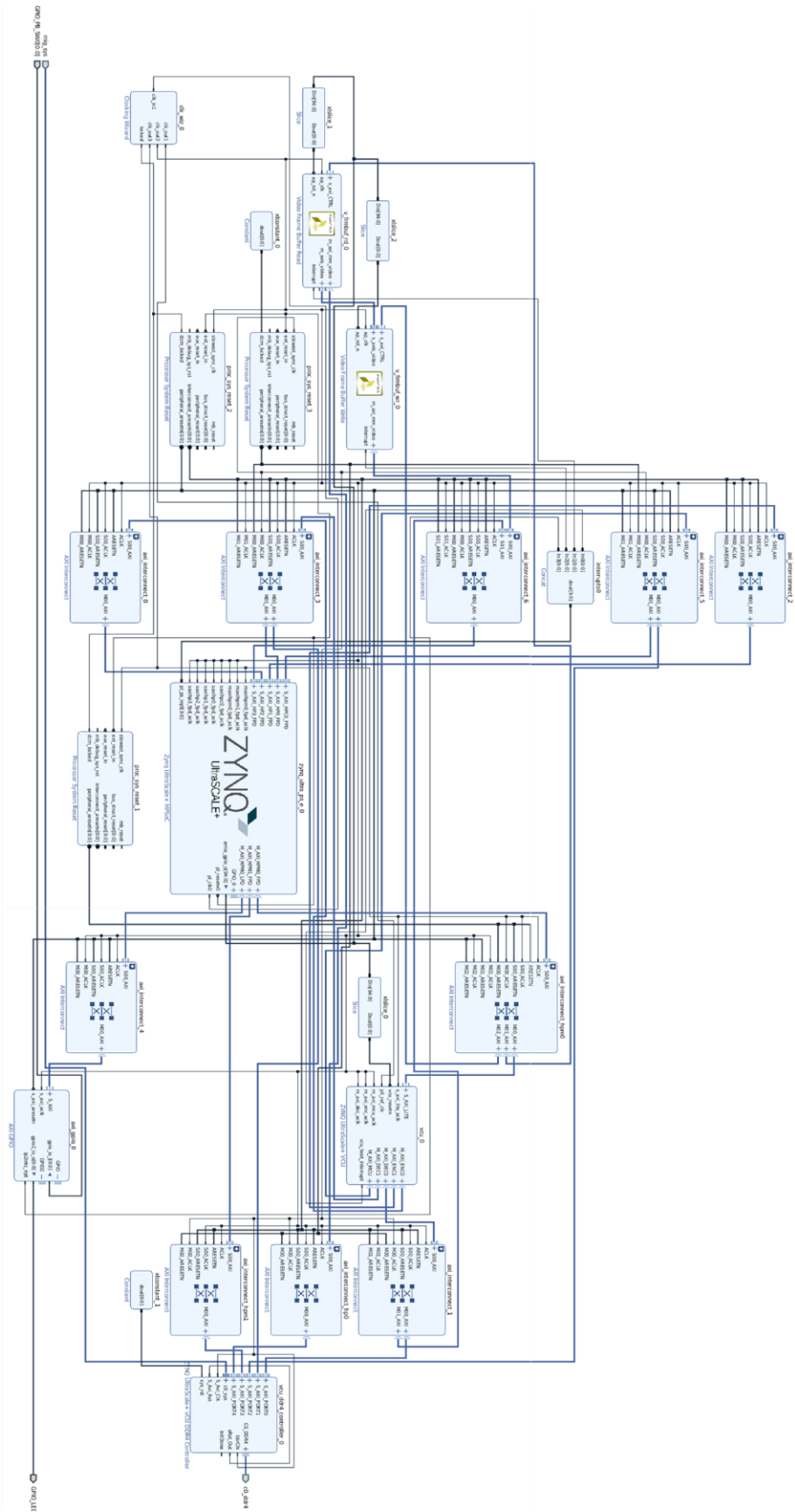
vcu-demo-camera-encode-file.sh

```
#!/bin/bash
#
#Get RAW YUV frames from Camera, encode it
#
# Copyright (C) 2017 Xilinx
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
type vcu-demo-functions.sh > "/dev/null"
if [ $? -ne 0 ]; then
        echo "Copy vcu-demo-functions.sh to /usr/bin/ or append it's path to PATH variable
and re-run the script" && exit -1
fi

source vcu-demo-functions.sh

scriptName=`basename $0`
declare -a scriptArgs=("inputPath" "v4l2Device" "videoSize" "codecType" "outputPath"
"numFrames" "targetBitrate" "audioType" "showFps" "compressedMode" "alsaSrc"
"pulseAudiosrc" "frameRate" "gopLength" "periodicityIdr")
declare -a checkEmpty=("v4l2Device" "codecType" "targetBitrate" "sinkName" "frameRate"
"gopLength" "periodicityIdr")


########################################################################
# Name:         usage
# Description: To display script's command line argument help
########################################################################
usage () {
        echo ' Usage : '$scriptName' -i <device_id_string> -v <video_capture_device> -s
<video_size> -c <codec_type> -o <output_path> -n <number_of_frames> -b <target_bitrate> -a
<audio_type>   -r   <capture_device_rate>   -f   --compressed-mode   --use-alsasrc   --use-
pulseaudiosrc --gop-length <gop_length> --periodicity-idr <periodicity_idr>'
        DisplayUsage "${scriptArgs[@]}"
        echo '  Example :'
        echo '  '$scriptName''
        echo '  '$scriptName' -o /mnt/sata/op.ts'
        echo '  '$scriptName' -v "/dev/video1"'
        echo '  '$scriptName' -v "/dev/video1" --gop-length 45'
        echo '  '$scriptName' -v "/dev/video1" --gop-length 45 --periodicity-idr 45'
        echo '  '$scriptName' --use-alsasrc -i "hw:1" -n 500 -a aac'
        echo '  '$scriptName' --use-alsasrc -i "hw:1" -n 500 -b 1200 -a aac'
        echo '  '$scriptName' --use-alsasrc -i "hw:1" -f -a aac'
        echo '  '$scriptName' --use-pulseaudiosrc -i "alsa_input.usb-
046d_C922_Pro_Stream_Webcam_FCD7727F-02.analog-stereo" -f -a aac'
        echo '  '$scriptName' -f -o fakevideosink'
        echo '  '$scriptName' -s 1920x1080 -c avc'
        echo '  '$scriptName' --use-alsasrc -i "hw:1" -s 1280x720 -c avc -a aac'
        echo '  '$scriptName' --use-alsasrc -i "hw:1" -a aac'
        echo '  '$scriptName' --use-alsasrc -i "hw:1" --compressed-mode -a vorbis'
```

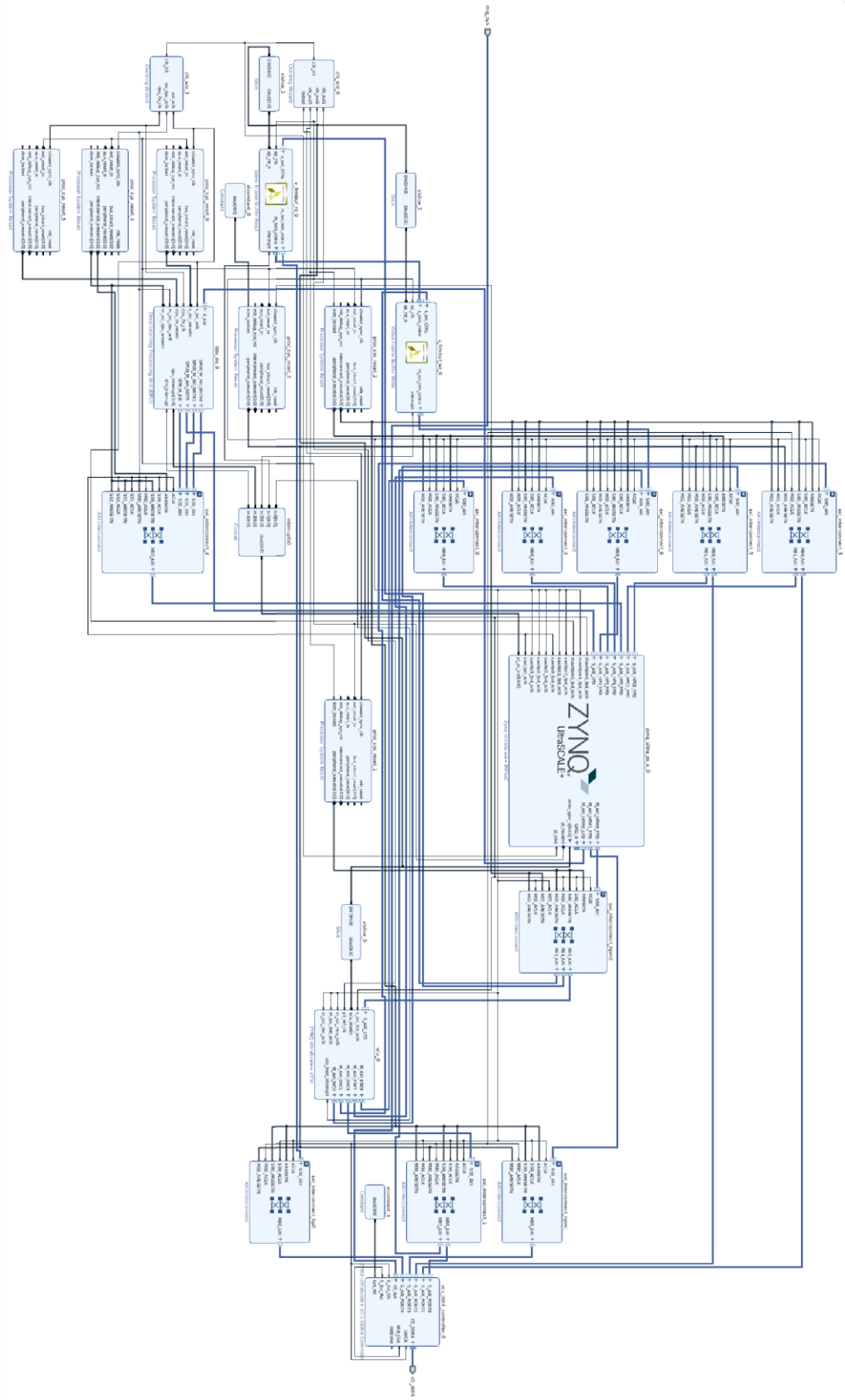
```
        echo     '          '$scriptName'     --use-pulseaudiosrc     -i     "alsa_input.usb-
046d_C922_Pro_Stream_Webcam_FCD7727F-02.analog-stereo" --compressed-mode -a vorbis'
        echo '   "NOTE: This script depends on vcu-demo-settings.sh to be present in
/usr/bin or its path set in $PATH"'
        exit
}

################################################################################
# Name:        CameraToFile
# Description: Get RAW data from camera, encode it
################################################################################
CameraToFile() {
        case $AUDIODEC_TYPE in
        "aac")
                AUDIODEC="faad"
                AUDIOENC="faac";;
        "vorbis")
                AUDIODEC="vorbisdec"
                AUDIOENC="vorbisenc";;
        *)
                if ! [ -z $AUDIODEC_TYPE ]; then
                        ErrorMsg "Invalid audio codec type specified, please specify either
vorbis or aac"
                fi
        esac

        OMXH264ENC="$OMXH264ENC  control-rate=constant  b-frames=2  gop-length=$GOP_LENGTH
periodicity-idr=$PERIODICITY_IDR prefetch-buffer=true target-bitrate=$BIT_RATE  ! video/x-
h264, profile=high"
        OMXH265ENC="$OMXH265ENC  control-rate=constant  b-frames=2  gop-length=$GOP_LENGTH
periodicity-idr=$PERIODICITY_IDR  prefetch-buffer=true  target-bitrate=$BIT_RATE  !  video/x-
h265, profile=main,level=\(string\)6.2,tier=main"
        IFS='x' read WIDTH HEIGHT <<< "$VIDEO_SIZE"

        case $CODEC_TYPE in
        "avc")
                ENC_PARSER=$H264PARSE
                DEC_PARSER=$H264PARSE
                ENCODER=$OMXH264ENC
                CAMERA_CAPS_ENC="video/x-
h264,width=$WIDTH,height=$HEIGHT,framerate=$FRAME_RATE/1";;
        "hevc")
                ENC_PARSER=$H265PARSE
                DEC_PARSER=$H265PARSE
                ENCODER=$OMXH265ENC
                CAMERA_CAPS_ENC="video/x-
h265,width=$WIDTH,height=$HEIGHT,framerate=$FRAME_RATE/1";;
        esac

        if [ -z $OUTPUT_PATH ]; then
                DIR_NAME=$(pwd)
                OUTPUT_PATH="$DIR_NAME/camera_output.ts"
        fi

        SINK="$FILESINK=$OUTPUT_PATH"
        OUTPUT_FILE_NAME=$(basename "$OUTPUT_PATH")
        OUTPUT_EXT_TYPE="${OUTPUT_FILE_NAME##*.}"
        if [ $SHOW_FPS ]; then
                SINK="fpsdisplaysink  name=fpssink  text-overlay=false  video-sink=fakesink
sync=true -v"
        fi

        MUX="mpegtsmux name=mux"

         if [ $NUM_FRAMES ]; then
                 V4L2SRC="$V4L2SRC num-buffers=$NUM_FRAMES"
                 AUDIO_BUFFERS=$(($NUM_FRAMES*100/$FRAME_RATE))
         fi
```

```
        setAudioSrcProps
        CAMERA_CAPS="video/x-raw,width=$WIDTH,height=$HEIGHT,framerate=$FRAME_RATE/1"
        VIDEOCONVERT="videoconvert"
        VIDEOCONVERT_CAPS="video/x-raw, format=\(string\)NV12"

        restartPulseAudio
        GST_LAUNCH="$GST_LAUNCH -e"

        if [ -z $AUDIODEC_TYPE ]; then
                if [ $COMPRESSED_MODE -eq 1 ]; then
                        pipeline="$GST_LAUNCH $V4L2SRC ! $CAMERA_CAPS_ENC ! $DEC_PARSER ! $QUEUE ! $MUX mux. ! $SINK"
                else
                        pipeline="$GST_LAUNCH $V4L2SRC ! $CAMERA_CAPS ! $VIDEOCONVERT ! $VIDEOCONVERT_CAPS ! $ENCODER ! $ENC_PARSER ! $QUEUE ! $MUX mux. ! $SINK"
                fi
        else
                if [ $COMPRESSED_MODE -eq 1 ]; then
                        pipeline="$GST_LAUNCH $V4L2SRC ! $CAMERA_CAPS_ENC ! $DEC_PARSER ! $QUEUE ! mux. $AUDIO_SRC ! $AUDIOCONVERT ! $AUDIOENC ! $QUEUE ! $MUX mux. ! $SINK"
                else
                        pipeline="$GST_LAUNCH $V4L2SRC ! $CAMERA_CAPS ! $VIDEOCONVERT ! $VIDEOCONVERT_CAPS ! $ENCODER ! $ENC_PARSER ! $QUEUE ! mux. $AUDIO_SRC ! $AUDIOCONVERT ! $AUDIOENC ! $QUEUE ! $MUX mux. ! $SINK"
                fi
        fi

        runGstPipeline "$pipeline"
}

# Command Line Argument Parsing
args=$(getopt -o "i:v:s:c:o:n:r:b:a:fh" --long "input-path:,video-capture-device:,video-size:,codec-type:,output-path:,num-frames:,bit-rate:,audio-type:,frame-rate:,gop-length:,show-fps,help,compressed-mode,use-alsasrc,use-pulseaudiosrc" -- "$@")

[ $? -ne 0 ] && usage && exit -1

trap catchCTRL_C SIGINT
parseCommandLineArgs
checkforEmptyVar "${checkEmpty[@]}"
if [ -z $VIDEO_SIZE ]; then
        VIDEO_SIZE="640x480"
        echo "Video Size is not specified in args hence using 640x480 as default value"
fi

if [ -z $BIT_RATE ];then
        BIT_RATE=1000
fi

RegSetting
CameraToFile
```

APPENDIX C: Image Capture Vivado Block Design



APPENDIX D: Full System Vivado Block Design



Glossary and Acronyms

Acronyms / Terms	Description
API	Application Programming Interface
APU	Application Processing Unit
ARM	Acorn RISC (Reduced instruction set computer) Machines
ATF	Arm Trusted Firmware
AVC	Advanced Video Coding
AXI	Advanced eXtensible Interface
BGR	Pixel colour space composed by three chroma coordinates (Blue, Green, Red)
CLI	Command-Line Interface
CMOS	Complementary Metal Oxide Semiconductor
CNN	Convolutional Neuronal Network
CPU	Central Processing Unit
DDR4	Double Data Rate Fourth Generation
DECENT	Deep Compression Tool
Device tree	Device Tree is a data structure that describes the HW components of the board and it is used by the kernel to handle them.
DIP	Dual In-line Package
DMA	Direct Memory Address
DNNC	Deep Neural Network Compiler
DNNDK	Deep Neuronal Network Development Kit
DP	Display Port
DPDMA	Display Port Direct Memory Address
DPU	Deep Learning Processor Unit
DRD	Dual-Role Device
FHD	Full High Definition
FIFO	First Input First Output
FPGA	Field-Programmable Gate Array
FPS	Frames Per Second
FSBL	First Stage Boot Loader

GDR	Gradual Decoder Refresh
GIC	Generic Interrupt Controller
GOP	Group Of Pictures
GPIO	General Purpose Input Output
GPU	Graphics Processing Unit
GStreamer	GStreamer is a pipeline-based multimedia framework that links together a wide variety of media processing systems to complete complex workflows.
GUI	Graphical User Interface
H.264	Video compression standard also known as Advanced Video Coding.
H.265	Video compression standard also known as High Efficiency Video Coding
HD	High Definition
HDL	Hardware Definition Language
HEVC	High-Efficiency Video Coding
HW	Hardware
IP	Intellectual Property
IR	Intermediate Representation
IRQ	Interrupt Request
ISP	Image Signal Processor
JTAG	Joint Test Action Group
Kernel	Kernel is a part of a Linux Operating System that manages the HW resources used by the Software.
LTS	Long Term Support
LVDS	Low-Voltage Differential Signal
MB	Macro Blocks
MCU	Machine Control Unit
MJPEG	Motion-JPEG (Joint Photographic Experts Group)
ML	Machine Learning
Mp	Mega pixels
MPSoC	Multi-Processor System on Chip
NN	Neural Network
OS	Operative System
PHY	Physical layer
PL	Programable Logic. Description of the HW implementation in the FPGA.

PMU	Platform Management Unit
PS	Processing System
RAM	Random Access Memory
RGB	Pixel colour space composed by three chroma coordinates (Red, Green, Blue)
ROM	Read-Only Memory
rootfs	Root File System. It is the top of the hierarchical file tree. It contains the files and directories critical for system operation, including the device directory and programs for booting the system.
SD	Secure Digital
SDIO	Secure Digital Input Output
SDK	Software Development Kit
SFM	Softmax
sysroots	Sysroots are the folder structures of the embedded image that contains the essentials for a system to run.
TCM	Tightly Coupled Memory
TFTP	Trivial File Transfer Protocol
UHD	Ultra-High Definition (4K, 8K, 16K)
ULPI	Utmi+ Low Pin Interface
UVC	USB Video Class
UYVY	Variant of YUV pixel colour space. It is a 4:2:2 subsampling with luma coordinate (Y) in each pixel and only one of the chroma coordinates (U or V) each pixel.
V4L2	Video for Linux 2
VCU	Video Codec Unit
VESA	Video Electronics Standards Association
VGG16	VGG16 is a convolutional network for classification and Detection. Its structure is included in the more complex SegNet architecture.
XSCT	Xilinx Software Command-line Tool
XSDK	Xilinx Software Development Kit
YCbCr	Pixel Colour space composed by one luma coordinate (Y) and two chroma coordinates (Cb, Cr)
Yocto	The Yocto Project (YP) is an open-source collaboration project that helps developers create custom Linux-based systems regardless of the hardware architecture.
YUV	Pixel Colour space composed by one luma coordinate (Y) and two chroma coordinates (U, V)