ALMARVI Python Camera Platform

Zdenek Pohl, Lukas Kohout, Jiri Kadlec
[zdenek.pohl, kohoutl, kadlec]@utia.cas.cz

Revision history

Rev.	Date	Author	Description
0	22.6.2016	Z.P.	Description of platform and files
1	28.6.2016	Z.P.	Added compilation from sources
2	29.6.2016	Z.P.	Added board modification note
3	1.7.2016	Z.P.	Improved description in compile from sources section
4	5.7.2016	Z.P.	sdcard format reqirements Trenz board package for Vivado Licensing and disclaimer added

Contents

1 Introduction 1
2 Platform Structure Description 1
2.1 FPGA 2
2.2 IP Licensing 3
2.3 Linux 4
2.4 Sobel Example 4
3 Used tools and resources 5
4 How to Run from Precompiled Bitstreams 5
5 How to Compile from Sources 6
6 Used Terminology 7
7 Known problems and solutions 7
8 Package contents 7
9 Licensing 8
10 Disclaimer 8
11 References 9

Acknowledgement

This work has been supported from project ALMARVI, project number ARTEMIS JU 621439 and MSMT 7H14004.

1 Introduction

This document provides description of Petalinux based video processing platform employing Python 1300 camera, Trenz carrier board and Zynq System on Module (SoM) and Avnet FMC IMAGEON extension board. Simple sw sobel filter example running on the platform is also provided. In this document, the platform is denoted as APCP (Almarvi Python Camera Platform).
The APCP package is provided by UTIA in two versions; the evaluation and full version. The evaluation is free and available for download from UTIA ZS web pages as it contains only documentation and precompiled bitstreams. The full version contains also hardware project with all necessary IP cores and support for Petalinux compilation. Thus the full version supports the extension of FPGA design by custom hardware blocks and linux image customization. This document is describing full version of the package with notes where the guides can be applied also to evaluation version.
Full version of the platform is available to ALMARVI project [1] partners as a baseline design to be extended by custom hardware accelerated image/video processing components.

2 Platform Structure Description

APCP consists of following components:

1. Hardware itself: Trenz TE0701-04 carrier board (green PCB), FMC IMAGEON extension card (red PCB), Python 1300 camera module (black in blue holder) and Zynq FPGA SoM module Trenz TE0720-02-2IF (under black passive cooler). All parts can be found in Figure [1].

   IMPORTANT NOTE: Trenz TE0701-04 carrier requires modifications to run the video platform. Boards provided to ALMARVI partners by UTIA have this modifications already done. For those who plan to build their own please contact kadlec@utia.cas.cz for modification details.

2. Xilinx Vivado 2015.4 project for implementation of FPGA bitstream where the Python 1300 camera interface, VDMA and HDMI output are supported. The project also requires camera and FMC IMAGEON extension card specific IP repository.
3. Petalinux 2015.4 sources with board dependent BSP file prepared for compilation of embedded linux.
4. Xilinx SDK workspace with sobel filter application and script for building Zynq bitstream.
5. Precompiled SD card files for platform quick test.

   NOTE: Evaluation version of this package contains only files described in point 5

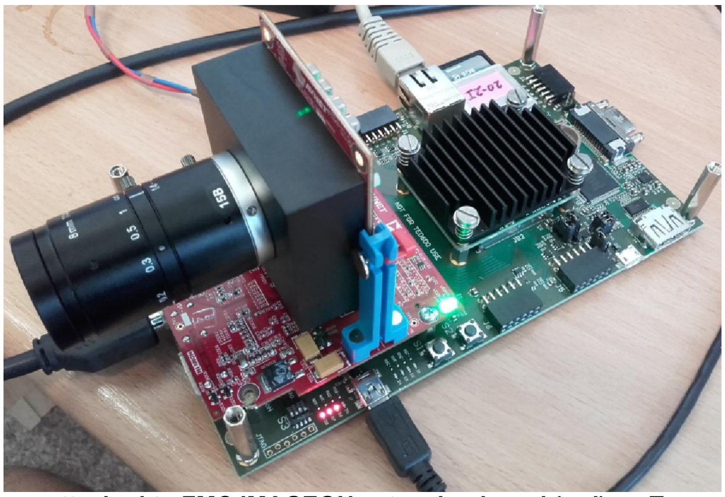


Figure 1: Python camera attached to FMC IMAGEON extension board (red) on Trenz Carrier (green). Trenz SoM with Zynq FPGA is located under passive cooler (black).

2.1 FPGA

FPGA design consists of camera input to stream conversion block ‘python1300_reciever_color’ which receives raw video signal from camera and converts it to video data stream. Pixels in video stream are in YUV422 format, i.e. 16 bit per pixel with subsampled color information. Video stream is then stored frame by frame to frame buffers in DDR memory by VDMA block. VDMA uses 7 frame storages in order to maximize data throughput (image processing algorithm can start anytime and always have data available to process except the case of maximum FPS speed was reached – at that moment synchronization will delay processing, detailed explanation is given in sobel example description). VDMA unit is also used for reading frames and sending them as video stream to ‘imageon_hdmi_out’. In output block, the video stream is converted to video signal and passed to HDMI output. Video frames have fixed resolution 1280x1080. Camera frame rate and HDMI output frame rate is 60 Hz. Clock for video stream processing blocks are set to 150 MHz. Block design of hardware project is shown in Figure [2].

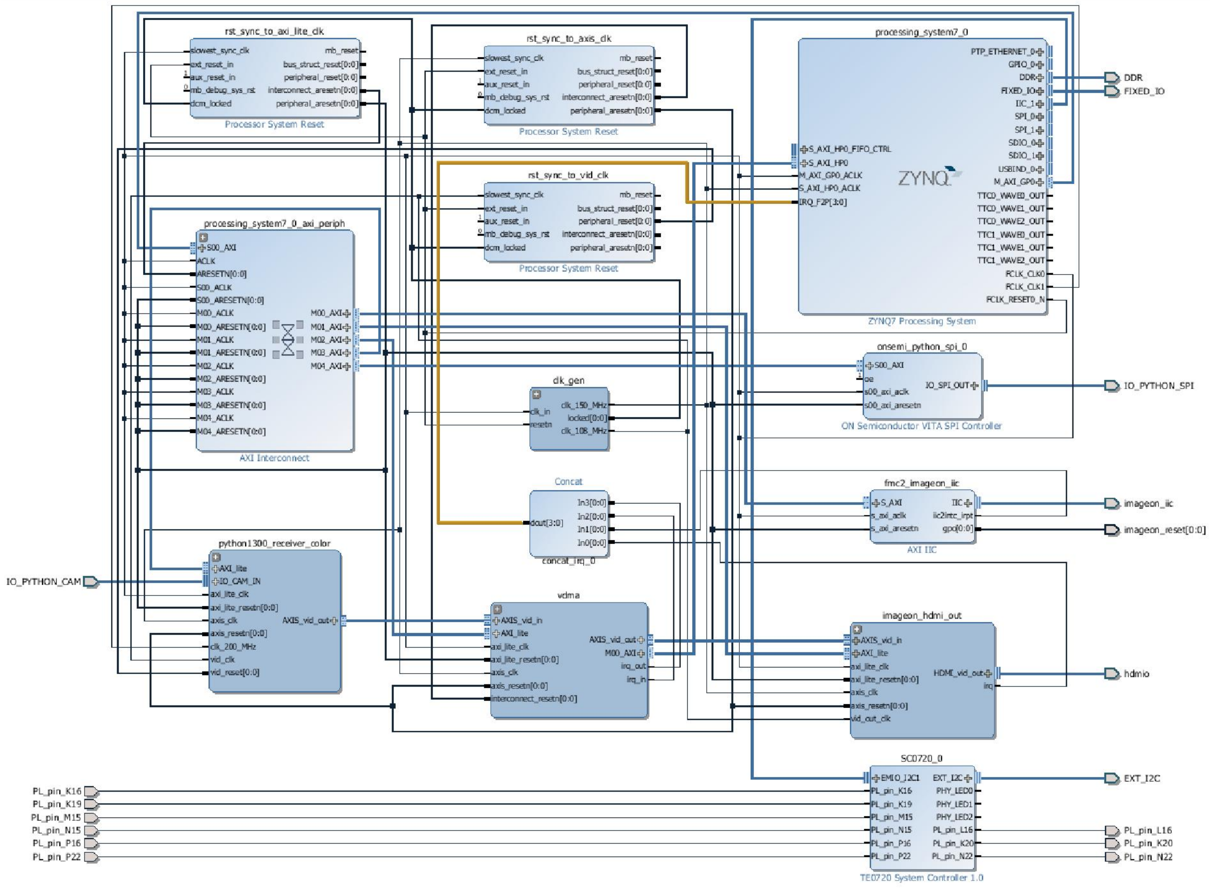


Figure 2: Xilinx Vivado Block design.

2.2 IP Licensing

Valid licenses (evaluation or full) are required in Xilinx Vivado for following video cores:

- CFA 7.0, Color Filter Array Interpolation, v_cfa
- Chroma Resampler 4.0, v_cresample
- RGB to YCrCb Color-Space Converter, v_rgb2ycrcb
- Video Timing Controller, v_tc

NOTE: The evaluation version of this appnote has been compiled with valid full license.

2.3 Linux

The compilation of Linux image from Petalinux 2015.4 sources is supported by providing platform BSP package file ‘te0720-02-2if-plnx.bsp’. This file can be used in standard way described in Xilinx user guides UG1144, UG1156, UG1157 and UG976 to build linux image ‘image.ub’ file. Precompiled version of the linux image is also provided.

2.4 Sobel Example

Sobel filter project requires Xilinx SDK 2015.4 tool and paths set to include headers and two libraries from linux_lib folder. Both libraries must be added for compilation in given order shown in the example project. IO space of all video chain components was mapped into user space to provide control of the chain to the application. The same drivers as in the standalone board support package were used for the linux application. VDMA unit requires also the definition of the framebuffers in main memory. For each framebuffer the physical and virtual address pair must be known. Physical address is used for VDMA configuration and virtual for linux user application. In provided Petalinux image, the memory area was reserved for that purpose from address 0x2E700000 with size big enough to hold 10 frames (10*1280*1024*2 bytes total).

The frame processing in user application is based on parking mode of VDMA unit for frame synchronization. The VDMA unit has two channels: reading and writing. Writing channel reads frames from video stream and writes them to one of frame buffers in DDR. Reading channel reads frame from DDR storage and writes it to video stream. In normal operation, the VDMA controller hardware automatically moves its output frame buffer to another location when one frame is received. It does so for both channels at the same time and it also ensures proper switching without collisions. Minimal number of framebuffers is 3. In the parking mode, the VDMA controller hardware can hold the storage of writing and reading channel at given frame storage index. The user must implement its own synchronization and switching.

In demo application case, the seven framebuffers were used to ensure maximum throughput. We define framebuffer indexes as follows:

- **writingFrame**

  a frame into which VDMA is writing incoming frames from camera (until not changed it is rewritten over and over again as explained above)

- **nextInputFrame**

  a frame which is prepared for immediate processing after currently processed frame will be finished

- **inputFrame**

  a frame currently accessed as input by image processing algorithm

- **nextOutputFrame**

  a frame prepared for immediate switch of image processing output when current frame is finished

- **outputFrame**

  a frame to which is currently being written by image processing algorithm

- **readingFrame**

  a frame which is currently read by VDMA and sent to output video stream (60FPS)

- **isolationFrame (not maintained)**

  a frame located between reading and writing frame. The exact order of frame parking switch cannot be guaranteed in our example of VDMA parking, thus one frame must be inserted between reading and writing to prevent collisions while switching.

The switch of VDMA parking positions can be started at the same time as computation of new frame processing by sobel filter. The software synchronization point is added after sobel filter processing to slow the algorithm down to 60 FPS if necessary.

Frame data are stored in DDR memory in YCrCb 4:2:2 format. Thus each pixel size is 2 bytes representing YCr or YCb pairs interlaced as the Cr Cb components are stored in half resolution compared to Y.

3 Used tools and resources

1. Vivado 2015.4 design suite
2. Petalinux 2015.4 for compilation of linux image (optional, provided precompiled image can be used)
3. SDK 2015.4 for compilation of example sobel application
4. Serial terminal application on PC
5. 1280x1024p60 capable monitor with HDMI input
6. HDMI cable, mini USB cable, Ethernet cable, power source
7. SD or SDHC card with FAT32 filesystem.

4 How to Run from Precompiled Bitstreams

This section is also applicable to evaluation version of the package.

1. Copy contents of the SD card folder to the SD card medium root folder

NOTE: SD card or SDHC may be used. The card must have FAT32 filesystem

2. Insert the SD card to the slot J8 on carrier board
3. Connect HDMI output from FMC IMAGEON to monitor
4. Connect mini USB cable from PC to J7 connector on carrier
5. For debug, the Ethernet cable can be connected to the J14 connector

6. Connect power cable and switch the board on
7. Connect serial terminal to the correct virtual COM port with parameters: 115200 bps, 8 bit, 1 stop, no parity, no flow
8. When linux is up and running type on serial console:
    a. cd /media/card
    b. ./sobel.elf

5 How to Compile from Sources

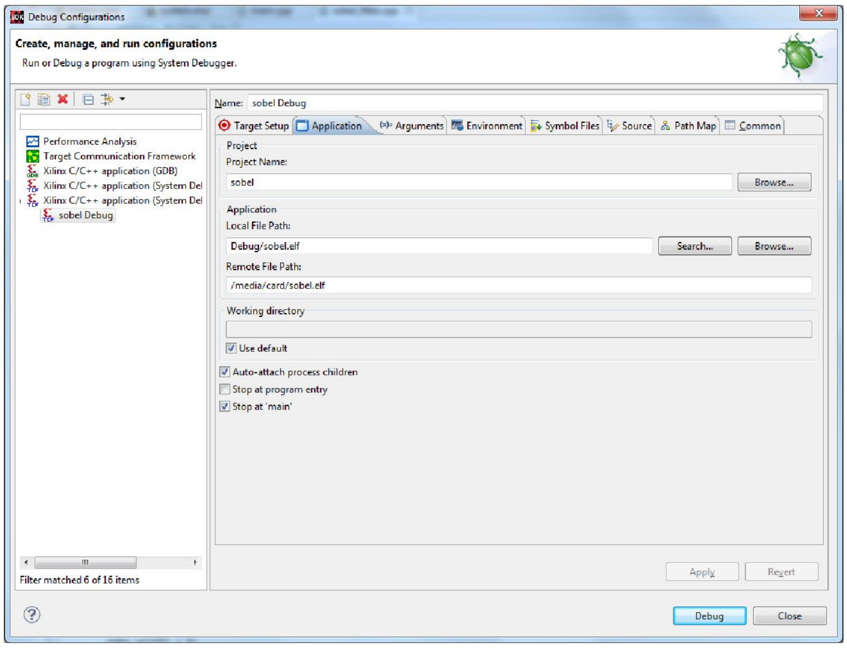
This guide applies only for full version of APCP package (with sources).

1. Install board files for Trenz module as follows:
    a. Unzip archive 'Trenz-BoardFiles-2015-12-09.zip'
    b. Copy subfolder '2015.4' from archive to <vivado_install_path>/Vivado (folder from archive will add to the same folder in Vivado the Trenz board support files)
    c. Check successful installation of files by setting up a new project in Vivado. The Trenz board should appear as a new option between boards in wizard.
2. Open hardware project in 'hw' folder in Xilinx Vivado tool
3. Check/add path to repositories. There must be reference to 'ip' folder
4. Build bitstream
5. Export hardware and bitstream to 'sdk_workspace' folder. The file 'system_wrapper.hdf' will be created in sdk_workspace folder.
6. Open Xilinx SDK and set its workspace to 'sdk_workspace' folder
    a. Delete fsbl, fsbl_bsp, standalone_bsp_0 and system_wrapper_hw_platfrom_0 projects (check delete also from filesystem)
    b. Create new fsbl project with new fsbl_bsp, for new hardware platform created from 'system_wrapper.hdf'. Name of the new hardware platform must be: 'system_wrapper_hw_platform_0'.
    c. Create new 'standalone_bsp_0' and delete 'sleep.h' from its include subfolder
    d. Xilinx tools -> Create Zynq boot image
        i. Choose import existing BIF file and use file 'sdk_workspace/boot/ linux.bif'
        ii. Click generate and confirm overwrite old 'boot.bin' file
    e. (Re)Build 'sobel.elf'. Its include directory list must point to '../../linux_lib/include' and to 'standalone_bsp_0/ps7_cortexa9_0/include'
7. Collect newly generated files and copy them to SD card root:

File	Location
sobel.elf	sdk_workspace/sobel/Debug
image.ub	sdk_workspace/boot or /petalinux (it is precompiled linux image)
boot.bin	sdk_workspace/boot

8. The sobel project may be also debugged using TCF debugger.
    a. Connect Ethernet cable from board to the PC and setup network
    b. In SDK connect Linux TCF agent to the board. Use board IP address and left other settings default.
    c. In debug configurations choose 'System debugger'

d. Fill in application tab settings as shown in image:



Create, manage, and run configurations

Run or Debug a program using System Debugger.

Name: sobel Debug

Project

Project Name: sobel

Application

Local File Path: Debug/sobel.elf

Remote File Path: /media/card/sobel.elf

Working directory

[✓] Use default

[✓] Auto-attach process children

[ ] Stop at program entry

[✓] Stop at 'main'

6 Used Terminology

- **Video signal** – a signal containing pixel data as well as horizontal and vertical frame synchronization (the exact constant timing pattern of synchronization signals and pixel data is used)
- **Video stream** – pixel data stripped off from frame synchronization signals. Pixel data are complemented by additional bits indicating start of frame (SoF) and end of line (EoL) for each frame.
- **AXI Stream** – a bus which is used for propagation of **video stream** through FPGA, implemented as FIFO with side channels tuser and tlast carrying SoF and EoL bits respectively.

7 Known problems and solutions

- sleep.h in standalone_bsp include folder makes conflict with unistd.h sleep functions. Solution is to remove sleep.h from standalone_bsp

8 Package contents

```
Release     - doc           this document
            - hw            Vivado project for Zynq
            - ip            IP repository for Vivado project
            - petalinux     Petalinux installation and BSP
            - sdcard        precompiled linux image and application
            - sdk_workspace workspace for Xilinx SDK
- Trenz-BoardFiles-2015-12-09.zip     archive with Trenz
                                      board support for
                                      Vivado
```

9 Licensing

Evaluation License

The evaluation version of the package can be downloaded from UTIA www pages free of charge for evaluation of the Python 1300 video sensor on TE0720-03-2IF module located on TE0701-04 carrier.

The evaluation package includes only precompiled bitstreams which has been compiled with valid full IP licenses of used IP components.

Full License

To obtain license to full package with sources files which allow custom extension of the platform by hardware accelerators please contact Jiri Kadlec, UTIA (kadlec@utia.cas.cz). The UTIA provides full version of the package to ALMARVI project partners free of charge.

10 Disclaimer

This disclaimer is not a license and does not grant any rights to the materials distributed herewith. Except as otherwise provided in a valid license issued to you by UTIA AV CR v.v.i., and to the maximum extent permitted by applicable law:

(1) THIS APPLICATION NOTE AND RELATED MATERIALS LISTED IN THIS PACKAGE CONTENT ARE MADE AVAILABLE "AS IS" AND WITH ALL FAULTS, AND UTIA AV CR V.V.I. HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and

(2) UTIA AV CR v.v.i. shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under or in connection with these materials, including for any direct, or any indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or UTIA AV CR v.v.i. had been advised of the possibility of the same.
Critical Applications:

UTIA AV CR v.v.i. products are not designed or intended to be fail-safe, or for use in any application requiring fail-safe performance, such as life-support or safety devices or systems, Class III medical devices, nuclear facilities, applications related to the deployment of airbags, or any other applications that could lead to death, personal injury, or severe property or environmental damage (individually and collectively, "Critical Applications"). Customer assumes the sole risk and liability of any use of UTIA AV CR v.v.i. products in Critical

Applications, subject only to applicable laws and regulations governing limitations on product liability.

11 References

[1] ARTEMIS JU project ALMARVI, http://sp.utia.cz/index.php?ids=projects/almarvi