

# Application Note



## Design Time and Run Time Resources for Zynq Ultrascale+ TE0808-04-15EG-1EE with SDSoc 2018.2 Support

Jiří Kadlec, Zdeněk Pohl, Lukáš Kohout  
[kadlec@utia.cas.cz](mailto:kadlec@utia.cas.cz) [xpohl@utia.cas.cz](mailto:xpohl@utia.cas.cz) [kohoutl@utia.cas.c](mailto:kohoutl@utia.cas.c)

### Revision history

Rev.	Date	Author	Description
0	11.04.2019	J. Kadlec	Initial draft
1			
2			

## Table of Contents

1	Introduction.....	1
2	Create SDSoC platform for Zynq Ultrascale+ board.....	2
3	Configuration of the PetaLinux 2018.2.....	6
4	Configuration of the Debian 9.8.....	8
5	Create the final SDSoC 2018.2 platform package.....	10
6	Compile HW accelerator by the SDSoC 2018.2 compiler.....	10
7	Video processing demo with Full HD HDMI Video In/Out.....	16
8	Inter-cloud connectivity based on the Arrowhead framework.....	19
9	Installation of Arrowhead Framework Services on RPi3.....	20
10	Install Arrowhead-f support on Zynq Ultrascale+ module.....	20
11	Install Arrowhead-f C++ Provider on Zynq Ultrascale+ module.....	20
12	Install Arrowhead-f C++ Consumer on Zynq Ultrascale+ module.....	20
13	Test the Zynq Ultrascale+ Consumer and Producer.....	21
14	Producer with real temperature measurement on Zynq Ultrascale+ module.....	21
15	Package content.....	26
	References.....	27
	Disclaimer.....	27

## Table of Figures

<i>Figure 1:</i>	TE0808-03-15EG-1EE on TEBF0808-04 carrier with Imageon HDMI I/O FMC.....	1
<i>Figure 2:</i>	The Zynq Ultrascale+ TE0808-03-15EG-1EE module and RaspberryPi 3B.....	2
<i>Figure 3:</i>	The initial Vivado design. It defines the SDSoC 2018.2 platform.....	4
<i>Figure 4:</i>	hdmi_in serves for input of Full HD HDMI from camera via Imageon FMC.....	5
<i>Figure 5:</i>	hdmi_out serves for output of Full HD HDMI to display via Imageon FMC.....	5
<i>Figure 6:</i>	vdma serves for video dma in/out to/from 8 Full HD video frame buffers in DDR4..	5
<i>Figure 7:</i>	RGPIP serves measurement of externally generated clock frequency.....	6
<i>Figure 8:</i>	The SW source code.....	14
<i>Figure 9:</i>	HW generated by the SDSoC 2018.2 compiler for matrix mult and add example ..	15
<i>Figure 10:</i>	HW Accelerated matrix multiplication and add.....	16
<i>Figure 11:</i>	LK DOF in HW with standard DMAs. Full HD HDMI video I/O.....	17
<i>Figure 12:</i>	LK Dense Optical Flow input movie Full HD HDMI video 60fps.....	18
<i>Figure 13:</i>	HW accelerated LK DOF input/output Full HD HDMI 60 fps.....	18
<i>Figure 14:</i>	Provider of the temperature of the Zynq Ultrascale+chip, response to requests..	25
<i>Figure 15:</i>	Consumer output of the temperature of the Zynq Ultrascale+ chip.....	25

## Table of Tables

<i>Table 1:</i>	tools with a corresponding package name.....	8
<i>Table 2:</i>	Performance of HW accelerated LK FOF and load Arm A53 processors.....	19

## Acknowledgement

This work has been partially supported from project FitOptiVis, project number ECSEL 783162 and the corresponding Czech NFA (MSMT) institutional support project 8A18013.

## 1 Introduction

This application note describes FitOptiVis design time and run time resources supporting the Zynq Ultrascale+ board and Xilinx SDSoC 2018.2 system level compiler.

The concrete board is Zynq Ultrascale+ TE0808-03-15EG-1EE [1]. It works with large Xilinx XCZU15EG-1FFVC900E device with the quad core Arm A53 64 bit, dual Arm Cortex R5 and programmable logic area on single 16nm chip. See *Figure 1*.



*Figure 1:* TE0808-03-15EG-1EE on TEBF0808-04 carrier with Imageon HDMI I/O FMC

The Zynq Ultrascale+ module has the 52 x 76 mm form factor. The Zynq Ultrascale+ board is designed and manufactured by company Trenz Electronic [1].



Figure 2: The Zynq Ultrascale+ TE0808-03-15EG-1EE module and RaspberryPi 3B

## 2 Create SDSoC platform for Zynq Ultrascale+ board

The Xilinx SDSoC 2018.2 compiler requires preparation of SDSoC platform. It is specific Vivado 2018.2 design with metadata, enabling to the SDSoC 2018.2 LLVM system level compiler to add additional HW accelerator blocks and data movers on top of the initial Vivado design. See *Figure 3*. The additional HW accelerator blocks are defined as C/C++ user defined functions. These functions can be compiled, debugged and executed in Petalinux user space on ARM A53. But in addition, the selected C/C++ functions can be compiled also to form of Vivado HLS HW accelerators. Blocks are compiled by the Vivado HLS compiler and automatically interfaced with dedicated data movers like DMA or SG DMA. See *Figure 9*.

The resulting compiled system remains compatible with related FitOptiVis run time resources, specifically the 64bit Debian OS and the local cloud Ethernet communication of C++ clients via the Arrowhead framework (result of ECSEL Productive 4.0 project) [2].

Creation of the board support package requires installation of the Xilinx SDSoC 2018.2 tool on your PC. Use the SDSoC 2018.2 web installer for Win7 or Win 10 (64bit) from: <https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/sdx-development-environments/2018-2.html>

The full SDSoC 2018.2 license is required for this large device. The Vivado 2018.2 Web pack license is not sufficient. Contact Xilinx to purchase the SDSoC license.

We will use the FitOptiVis WP3 Design time resource – **the Zynq Ultrascale+ board support package generation project** included in the evaluation package accompanying this application note. The board support package generation project serves for generation of the **board support package** for the TE0808-03-15EG-1EE module on TEBF0808-04 HW carrier with Video I/O. The board support package provides all necessary files needed for the Xilinx SDSoC 2018.2 compiler. The compiler needs this board support package to be able to compile selected C/C++ Arm A53 functions into HW accelerators and the corresponding bit-stream for the programmable part of the design. The board support package includes all necessary information for preparation of the low level SW support for the preconfigured and precompiled Petalinux 2018.2 kernel and for the precompiled Debian 9.8 “Stretch” image for the for the TE0808-03-15EG-1EE module on TEBF0808-04 HW carrier with Video I/O.

Image files included in this evaluation package can be used for quick first evaluation of the development flow of the SDSoC 2018.2 platform. Configurations and compilations of the Petalinux 2018.2 kernel and the Debian 9.8 “Stretch” image are described in Chapters 3 and 4.

To prepare the Zynq Ultrascale+ SDSoC board support package for the TE0808-03-15EG-1EE module on TEBF0808-04 HW carrier with Video I/O follow these steps:

1. Unpack the enclosed evaluation package

*TE0808\_SDSoC\_HIO2.zip*

to Win 7 or Win10 directory of your choice. We will use:

```
c:\TS82\TE0808_SDSoC_IMAGEON_FMC_HDMI_701HDMI\
```

It will create *zusys* folder.

2. On Win 7 or Win10, open dos terminal window, change directory to the *zusys* folder and create an initial setup:

```
cd c:\TS82\TE0808_SDSoC_HIO2\zusys
_create_win_setup.cmd
```

Select option (1) to create maximum setup of CMD-Files and to exit.

Set of scripts is created in the *zusys* folder.

To overcome limitations of Win 7 and Win10 related to the need of short directory paths, use the script *\_use\_virtual\_drive.cmd* to create a virtual short path to your directory drive X:\zusys Type:

```
_use_virtual_drive.cmd
```

Select X as name of the virtual drive and select (0) to create the virtual drive.

Go to the created virtual short-path directory by:

```
X:
cd zusys
```

3. Use text editor of your choice and open and modify script *design\_basic\_settings.sh* Select correct path to SDSoC 2018.2 tool installed on your Win7 or Win10. Line 38:

```
@set XILDIR=C:/Xilinx
```

Select proper Xilinx device. Line 48:

```
@set PARTNUMBER=15
```

The selected number corresponds to the number defined in file

X:\zusys\board\_files\TE0808\_board\_files.csv

Verify, if line 78 sets the SDSoC flow support by: *ENABLE\_SDSOC=1*

```
@set ENABLE_SDSOC=1
```

4. Start the Xilinx Vivado 2018.2 and create the design by executing of the script:

```
X:\zusys\vivado_create_project_gui mode.cmd
```

Figure 3 shows block design of the created system. It includes 4 HW reset IPs for future HW accelerators with system clocks 25 MHz, 100 MHz, 150 MHz and 287.5 MHz.

The DDR4 interface and the connections to the USB ports for keyboard, mouse and 1Gbit Ethernet are all pre-configured inside of the Vivado Zynq Ultrascale+ block `zynq_ultra_ps_e_0`.

5. To build the Vivado 2018.2 design, use the TCL script provided within the board support package. From the Vivado TCL console execute command:

```
TE::hw_build_design -export_prebuilt
```

After the compilation, new hardware description file `zusys.hdf` is generated in folder:

```
X:\zusys\prebuilt\hardware\4ev_1e\zusys.hdf
```

Copy the three precompiled files from the enclosed evaluation package to:

```
X:\zusys\prebuilt\os\petalinux\default\image.ub
```

```
X:\zusys\prebuilt\os\petalinux\default\u-boot.elf
```

```
X:\zusys\prebuilt\os\petalinux\default\b131.elf
```

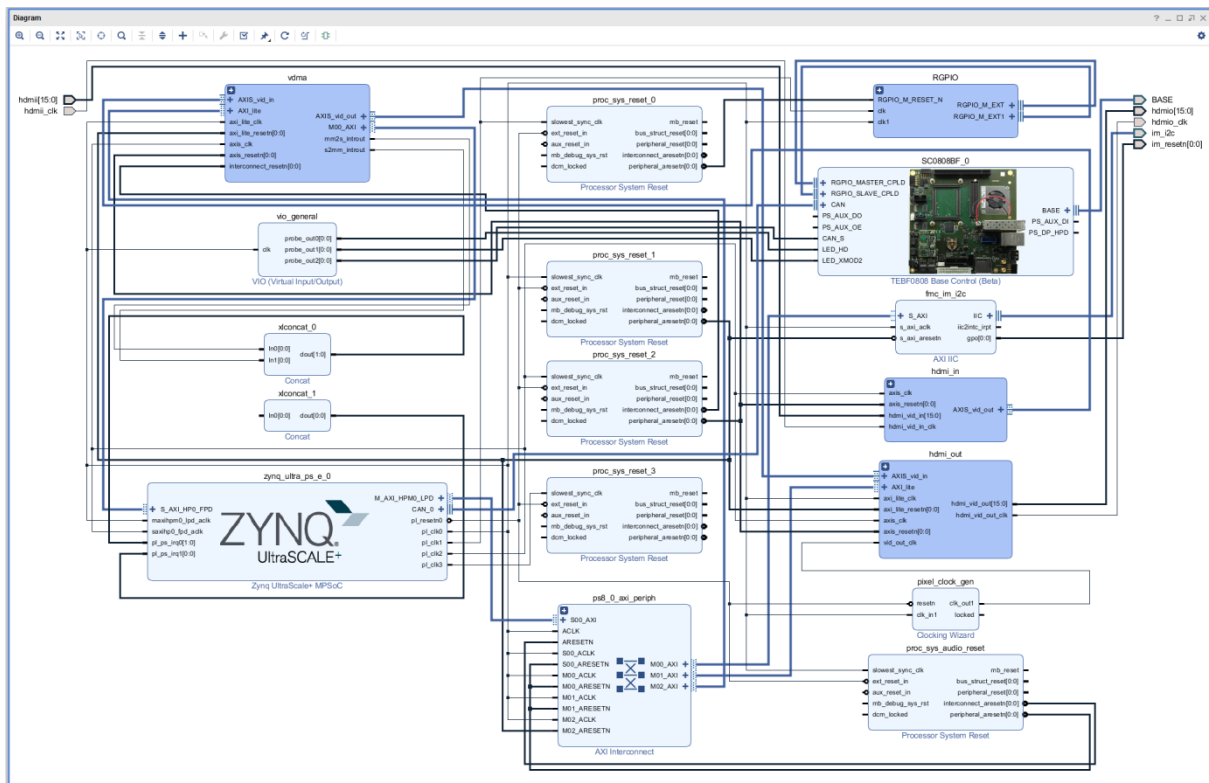


Figure 3: The initial Vivado design. It defines the SDSoc 2018.2 platform.

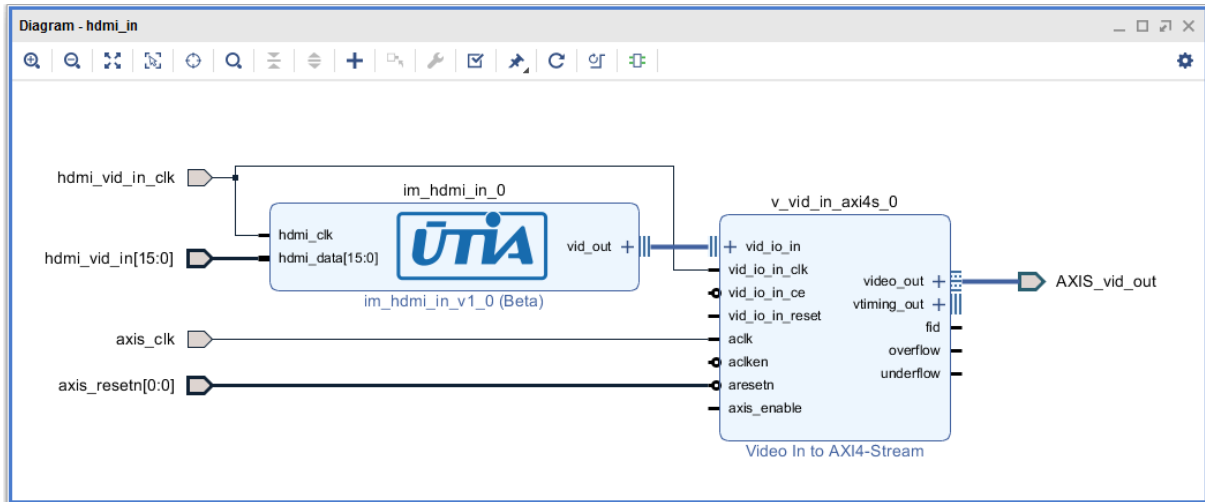


Figure 4: hdmi\_in serves for input of Full HD HDMI from camera via Imageon FMC

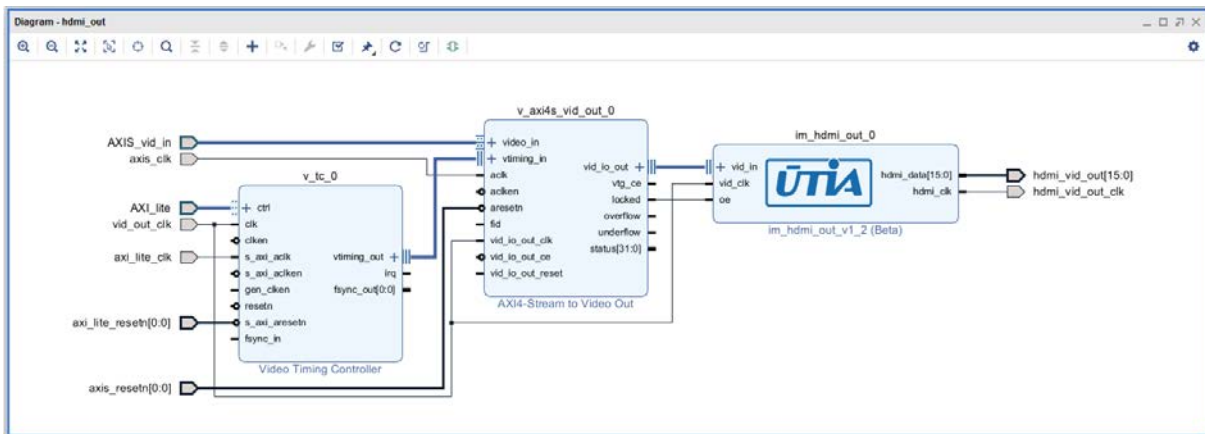


Figure 5: hdmi\_out serves for output of Full HD HDMI to display via Imageon FMC

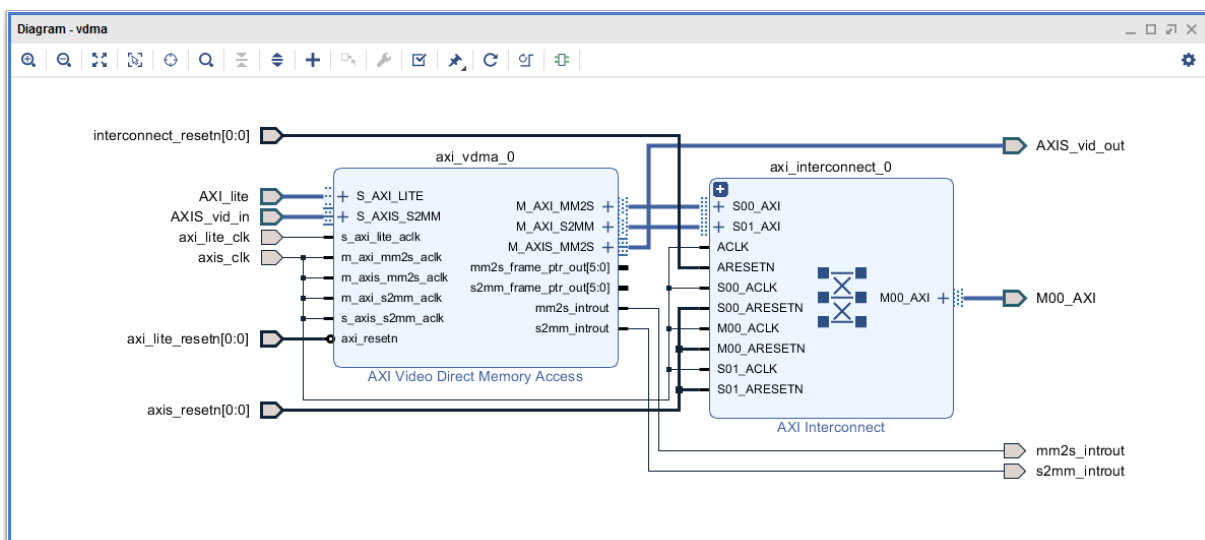


Figure 6: vdma serves for video dma in/out to/from 8 Full HD video frame buffers in DDR4

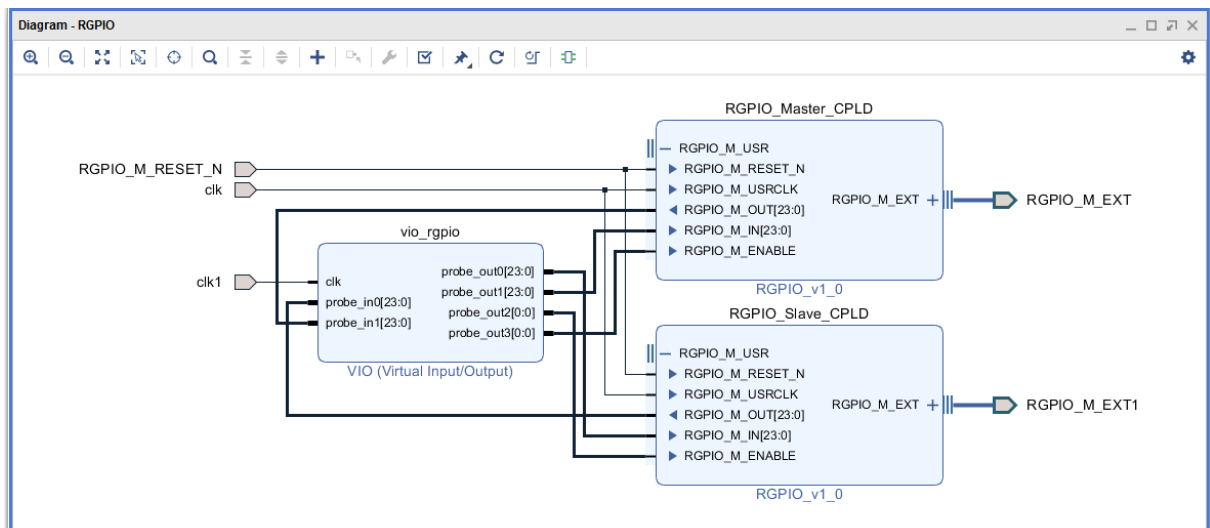


Figure 7: RGPIO serves measurement of externally generated clock frequency.

The hierarchical blocks of Figure 3 described in Figure 4 - Figure 7 form the Full HD video in/out support of the platform.

Platform has one full HD HDMI video input via the Imageon FMC. It serves for video input for the HW accelerated video processing algorithms working on 8 Full HF video frame buffers reserved in the DDR4.

Platform has one Full HD HDMI video output via the Imageon FMC. It serves for video output for the HW accelerated video processing algorithms working on 8 Full HD video frame buffers reserved in the DDR4.

Platform has second Full HD HDMI video output via the HDMI connector on the TE0701 carrier board. It serves for Debian video output from single separate Full HD video frame buffer reserved in the DDR4.

All these subsystems will be present in each demo compiled by the created SDSoc 2018.2 platform. The VDMA subsystems can be controlled by user from the user-space SW running on top of the appropriately configured *Petalinux 2018.2* kernel and *Debian 9.8 "Stretch"* operating system. These configurations/compilations are described in next two sections.

### 3 Configuration of the PetaLinux 2018.2

The configuration and compilation of the *Petalinux 2018.2* kernel and *Debian 9.8 Stretch* image as the FitOptiVis run time resource for the Zynq Ultrascale+ module TE0808-03-15EG-1EE is described now. The configuration is performed on the Ubuntu 16.04 LTS.

We used the *VMware Workstation 14 Player* on Win7 or Win10 PC with Intel i7 CPU (8 processors, 16 GB RAM). We use configuration of the VM machine with allocated 6 processors and 8 GB of RAM for the Ubuntu 16.04 LTS. It results in fast compilation of the PetaLinux 2018.2 kernel.

The PetaLinux 2018.2 distribution can be downloaded to the Ubuntu 16.04 LTS from <https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools/2018-2.html>



and installed to the default Ubuntu directory:

```
/opt/petalinux/petalinux-v2018.2-final
```

The standard PetaLinux 2018.2 distribution requires few modifications.

1. Copy to the Ubuntu OS all content of these to Win7 or Win 10 directories:

```
X:\zusys\prebuilt
```

```
X:\zusys\os
```

to Ubuntu directories:

```
/home/devel/work/TS82/TE0808/zusys/os
```

```
/home/devel/work/TS82/TE0808/zusys/prebuilt
```

2. In Ubuntu, open linux terminal window and set path to the PetaLinux 2018.2:

```
source /opt/petalinux/petalinux-v2018.2-final/settings.sh
```

3. Go to the directory copied from the evaluation package with pre-defined configuration for the Zynq Ultrascale+ module TE0808-03-15EG-1EE:

```
cd /home/devel/work/TS82/TE0808/zusys/os/petalinux
```

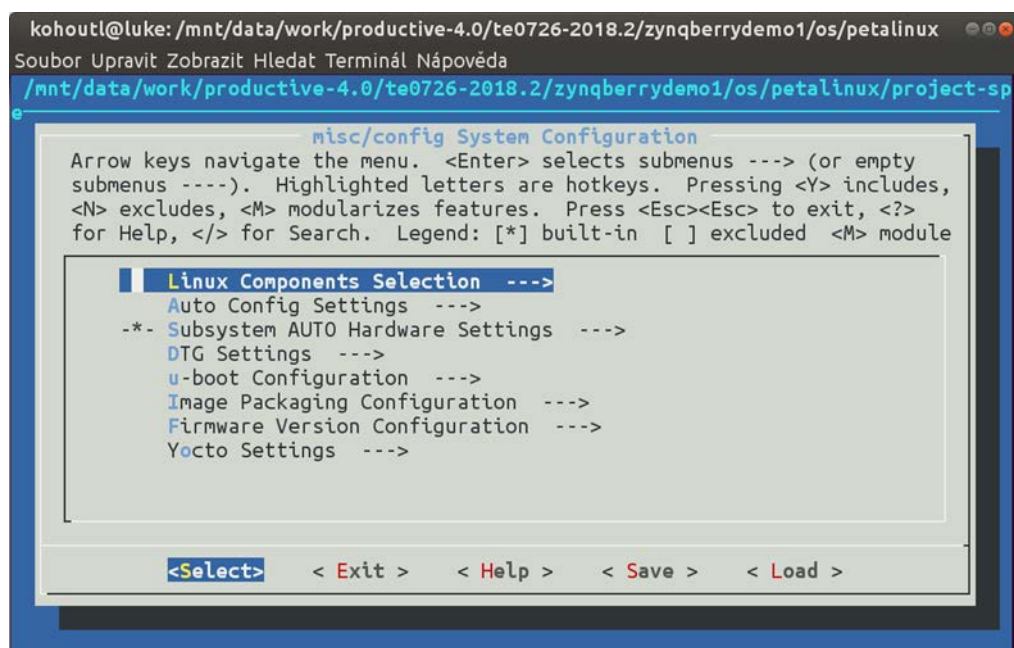
It contains a predefined configuration according to Zynq Ultrascale+ board requirements.

4. The HDF file created (see chapter 3) in Win7 or Win 10 in Vivado 2018.2 tool is present in the Ubuntu folder:

```
/home/devel/work/TS82/TE0808/zusys/prebuilt/hardware/4ev_1e/zusys.hdf
```

5. Load the HDF to current configuration by PetaLinux command (on single line):

```
petalinux-config --get-hw-description=/home/devel/work/TS82/TE0808/zusys/prebuilt/hardware/4ev_1e
```



6. Test if the PetaLinux filesystem location is changed from the ramdisk to the extra partition on the SD card, select:

```
Image Packaging Configuration --->
```

```
Root filesystem type (SD card) --->
```

7. Test if option to generate boot args automatically is disabled and if user defined arguments are set to:

```
earlycon clk_ignore_unused root=/dev/mmcblk1p2 rootfstype=ext4 rw
rootwait quiet
```

Leave the configuration, 3x *Exit* and *Yes*.

8. To build PetaLinux, from the bash terminal execute PetaLinux command:

```
petalinux-build
```

9. Files *image.ub*, *u-boot.elf* and *bl31.elf* are created in:

```
/home/devel/work/TS82/TE0808/zusys/os/petalinux/images/linux/image.ub
/home/devel/work/TS82/TE0808/zusys/os/petalinux/images/linux/u-boot.elf
/home/devel/work/TS82/TE0808/zusys/os/petalinux/images/linux/bl31.elf
```

## 4 Configuration of the Debian 9.8

The file system is based on the latest stable version of Debian 9.8 Stretch distribution (03.25.2019). Follow the steps below.

1. Copy the *mkdebian.sh* file from this evaluation package distribution to the PetaLinux folder.

```
/home/devel/work/TS82/TE0808/zusys/os/petalinux/mkdebian.sh
```

2. Go to the folder with PetaLinux:

```
cd /home/devel/work/TS82/TE0808/zusys/os/petalinux
```

3. The 64bit Debian image will be created by execution of the *mkdebian.sh* script. The script checks all the tools that are needed to create the image, most of them are a standard part of the Ubuntu 16.04 LTS distribution.

When some of them are missing, install them by:

```
sudo apt install Package
```

Table 1: tools with a corresponding package name.

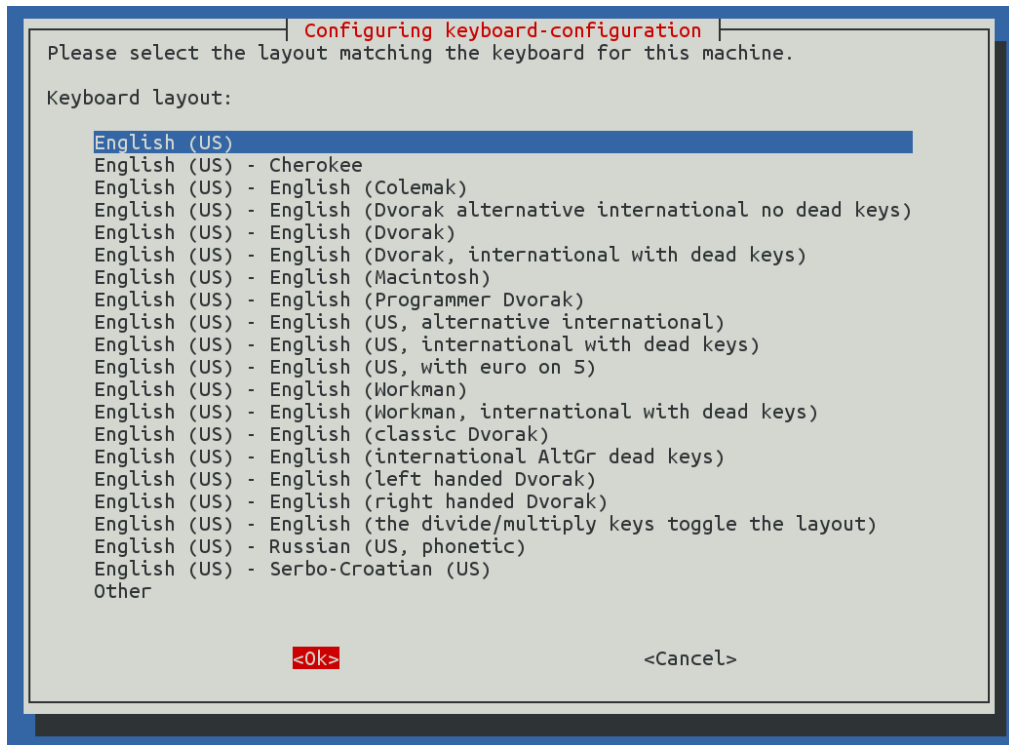
Tool	Package
dd	coreutils
losetup	mount
parted	parted
lsblk	util-linux
mkfs.vfat	dosfstools
mkfs.ext4	e2fsprogs
debootstrap	debootstrap
gzip	gzip
cpio	cpio
chroot	coreutils
apt-get	apt
dpkg-reconfigure	debconf
sed	sed
locale-gen	locales
update-locale	locales
qemu-arm-static	qemu-user-static

4. Create the Debian image. It will consist of two partitions.

The file system of the first one will be FAT32. This partition is dedicated for image of the PetaLinux kernel. The second partition will contain the Debian using EXT4 file system. Create the Debian image from the external Ethernet repositories by this command:

```
chmod ugo+x mkdebian.sh
sudo ./mkdebian.sh
```

During the creation procedure, you will be asked to set language. Choose *English (US)*. The resultant image file will be called *TE0808-debian.img*, its size will be 7 GB.



This step can take some time. It depends on the host machine speed and speed of the internet connection.

5. Compress the created image to file *TE0808-debian.zip*:

```
zip TE0808-debian TE0808-debian.img
```

6. Copy compressed image file from Ubuntu

```
/home/devel/work/TS82/TE0808/zusys/os/petalinux/TE0808-debian.zip
```

to Win7 or Win 10 file:

```
X:\zusys\prebuilt\os\petalinux\default\TE0808-debian.zip
```

7. Copy these files from Ubuntu

```
/home/devel/work/TS82/TE0808/zusys/os/petalinux/images/linux/image.ub
/home/devel/work/TS82/TE0808/zusys/os/petalinux/images/linux/u-boot.elf
/home/devel/work/TS82/TE0808/zusys/os/petalinux/images/linux/bl31.elf
```

to Win7 or Win 10 files:

```
X:\zusys\prebuilt\os\petalinux\default\image.ub
```

```
X:\zusys\prebuilt\os\petalinux\default\u-boot.elf
```

```
X:\zusys\prebuilt\os\petalinux\default\bl31.elf
```

#### 8. In Ubuntu, clean Petalinux project files

```
petalinux-build -x mrproper
```

#### 9. In Ubuntu, delete files

```
/home/devel/work/TS82/TE0808/zusys/os/petalinux/TE0808-debian.zip
```

```
/home/devel/work/TS82/TE0808/zusys/os/petalinux/TE0808-debian.img
```

#### 10. In Ubuntu, close all applications and shut down.

#### 11. In Win7 or Win 10, close the VMware Workstation Player 14.

You can continue with preparation of the Zynq Ultrascale+ board with created files:

- Petalinux kernel image *image.ub*
- Compressed Debian image *te0808-debian.zip*
- U-boot program *u-boot.elf*
- Support firmware *bl31.elf*

This ends configuration and compilation step for the Petalinux and Debian.

## 5 Create the final SDSoC 2018.2 platform package

1. In the open Vivado 2018.2 console, create and compile the initial *BOOT.bin* file and the initial SW modules by execution of the command:

```
TE::sw_run_hsi
```

The resulting *BOOT.bin* file will be located in the folder

```
X:\zusys\prebuilt\boot_images\15eg_1eb_sk\u-boot\BOOT.bin
```

2. In Vivado 2018.2 console, create the SDSoC platform by execution of the command:

```
TE::ADV::beta_util_sdsoc_project
```

The SDSoC 2018.2 platform will be generated in the directory

```
X:\SDSoC_PFM\TE0808-04\15EG-1EE
```

The platform it is also packed into a ZIP file in the directory

```
X:\SDSoC_PFM\TE0808-04\
```

This ends the configuration and compilation steps needed for the initial generation of the SDSoC 2018.2 platform for the TE0808-03-15EG-1EEA module on TEBF0808-04 carrier.

Platform created in chapters 1 – 5 is used in all demos described in next sections of this application note.

## 6 Compile HW accelerator by the SDSoC 2018.2 compiler

Simple matrix multiplication-and-addition application is coded in C and compiled by the SDSoC 2018.2 compiler into HW accelerator for the platform defined in Chapters 5.

1. On Win 7 or Win10, cancel the current virtual drive X: by executing from the currently open command line:

```
_use_virtual_drive.cmd
```

and type response:

```
1
```

2. Change directory to

```
C:\TS82\TE0808\TE0808_SDSoc_HIO2\SDSoC_PFM\TE0808-04\15EG-1EE
```

3. On Win 7 or Win10, open dos terminal window and execute copy of the script `_use_virtual_drive.cmd` to create a new virtual short path to get short SDSoc directory `X:\15EG-1EE`

```
_use_virtual_drive.cmd
```

Select X as name of the virtual drive and type

```
0
```

to create the virtual drive.

Change directory to the created `X:\15EG-1EE` directory:

```
X:
```

```
cd 15EG-1EE
```

4. Open SDSoc 2018 tool in the directory

```
X:\15EG-1EE
```

5. Create new linux target project named

```
te30_1
```

6. Select platform:

```
X:\15EG-1EE\zusys
```

7. Select template project

```
X:\15EG-1EE\zusys\samples\z_is_a_times_b_direct_connect
```

and compile it for the *Release* target with all clocks set to 187,5 MHz.

This example will accelerates *in HW the* int32 matrix operation:

$$D[400,400] = A[400,400] * B[400,400] + C[400,400]$$

in the programmable logic of the Zynq Ultrascale+ device.

8. The SDSoc 2018.2 compiler will create these relevant results in the `sd_card` directory:

```
X:\15EG-1EE\te30_1\Release\sd_card\BOOT.BIN
```

```
X:\15EG-1EE\te30_1\Release\sd_card\te30_1.elf
```

9. Unzip the preconfigured and precompiled Debian image for the Zynq Ultrascale+ board from the evaluation package file: `TE0808-debian.zip` to file `TE0808-debian.img`.

10. Use the *Win32DiskImager* <https://sourceforge.net/projects/win32diskimager/> for creation of the image `TE0808-debian.img` on the SD card. Use 8GB SD card with speed grade 10.

11. Copy to the root of the SD card the HW accelerated matrix multiplication demo executable `te30_1.elf` and the corresponding `BOOT.BIN` file:

```
X:\15EG-1EE\te30_1\Release\sd_card\BOOT.BIN
```

```
X:\15EG-1EE\te30_1\Release\sd_card\te30_1.elf
```

The `BOOT.BIN` file contains the first stage boot loader, the u-boot and the bitstream with the platform design extended by the HW accelerator for matrix multiplication and addition. Application `te30_1.elf` requires that the Zynq Ultrascale+ device is booted with the corresponding `BOOT.bin` file. That is why you have to copy both related files.

12. Remove DS card from PC and insert it to the Zynq Ultrascale+ board.

13. Connect the Zynq Ultrascale+ board to the Ethernet cable.

14. Connect Full HD HDMI video source to the video input HDMI connector of the FMC Imageon card.

15. Connect Full HD HDMI display to the video output HDMI connector of the FMC Imageon card.

16. Connect 4K display to the video output DisplayPort connector of the TEBF0808 carrier board. It will provide the Debian desktop GUI.
17. Connect mouse and keyboard to USB connectors of the TEBF0808 carrier board. It will serve for the Debian desktop GUI input.
18. On PC, you can use the *putty* terminal (download from: <https://www.putty.org/>).
19. Connect the Zynq Ultrascale+ board with your PC via mini USB cable. The mini USB cable supports two connections, the programming JTAG interface and the console. Use *putty* or similar terminal client with *speed (baud) 115200 bps, data bits 8, stop bits 1, parity none and flow control none*. The actual COM port number associated with your connection can be found by the Win7 or Win10 *Device manager* utility.
20. Connect the 12V power supply. The TEBF0808 carrier board is in a stand-by mode. The blue led in the power-on button is blinking and the fan is not running.
21. Press the power-on button to switch the power-on. The fan is running.
22. Press the reset button on TEBF0808 carrier board.
23. The Zynq Ultrascale+ board will start booting process from the SD card. The first stage boot loader (fsbl) program is executed first. It starts the u-boot program. The u-boot program configures the Arm Cortex A9 processing system and boots the preconfigured and precompiled Petalinux *image.ub* image from the SD card with text output to the serial line terminal. The Debian file system is present on the separate partition of the SD card.
24. Login as user:

```
root
```

Password:

```
root
```

25. Find and write down the Ethernet IP address for IP V4 and IP V6 address assigned by the DHCP server by typing command on the console:

```
ifconfig
```

26. The Full HD screen is opened text console on the 4K monitor connected to the video output DisplayPort connector of the TEBF0808 carrier board. Use the USB keyboard and login as:

```
root
```

Password:

```
root
```

Type:

```
startx&
```

The graphical Debian desk-top GUI will open automatically on the 4K monitor with Full HD resolution (1920x1080p60). The USB keyboard and the USB mouse can be used to control the Debian desk-top.

27. The HW accelerated matrix multiplication demo can be executed on the Zynq Ultrascale+ module from the automatically mounted SD card by executing this command.

```
/boot/te06_1.elf
```

28. The HW acceleration measured by the number of Arm A9 clock cycles. See *Figure 10*.

29. To shut down properly the Debian type from the console terminal:

```
halt
```

The Debian OS is properly shut down and all possibly open R/W to the SD card are closed. Press the button with blue led to switch-off power.

The SDSoC 2018.2 compiler have created and compiled new HW accelerator to the programmable logic part of the device from the C++ source code *mmult.cpp*. It accelerates int32 matrix operation:  $D[400,400] = A[400,400] * B[400,400] + C[400,400]$ .

See the listing of *mmult.cpp*:

```
#include "mmult.h"
// Computes matrix addition
// Out = (out + in3) , where a direct connection establishes between the
// HLS kernels for the access of "out"(A X B)
void madd_accel(
    const int *mmult_in,    // Read-Only Matrix
    const int *in3,        // Read-Only Matrix 3
    int *out,              // Output matrix
    int dim                // Size of one dimension of the matrices
)
{
    // Performs matrix addition over output of (A x B) and In3 and
    // writes the result to output
    write_out: for(int j = 0; j < dim * dim; j++) {
        #pragma HLS PIPELINE
        #pragma HLS LOOP_TRIPCOUNT min=1 max=160000
        out[j] = mmult_in[j] + in3[j];
    }
}
// Computes matrix multiplication
// out = (A x B) , where A, B are square matrices of dimension (dim x dim)
void mmult_accel(
    const int *in1,        // Read-Only Matrix 1
    const int *in2,        // Read-Only Matrix 2
    int *out,              // Output Result
    int dim                // Size of one dimension of the matrices
)
{
    // Local memory to store input and output matrices
    // Local memory is implemented as BRAM memory blocks
    int A[MAX_SIZE][MAX_SIZE];
    int B[MAX_SIZE][MAX_SIZE];
    #pragma HLS ARRAY_PARTITION variable=A dim=2 complete
    #pragma HLS ARRAY_PARTITION variable=B dim=1 complete
    // Burst reads on input matrices from DDR memory
    // Burst read for matrix A, B and C
    read_data: for(int itr = 0 , i = 0 , j =0; itr < dim * dim; itr++, j++){
        #pragma HLS PIPELINE
        #pragma HLS LOOP_TRIPCOUNT min=160000 max=160000
        if(j == dim) { j = 0 ; i++; }
        A[i][j] = in1[itr];
        B[i][j] = in2[itr];
    }
}
```

```

}
// Performs matrix multiply over matrices A and B and stores the result
// in "out". All the matrices are square matrices of the form (size x size)
// Typical Matrix multiplication Algorithm is as below
mmult1: for (int i = 0; i < dim ; i++) {
#pragma HLS LOOP_TRIPCOUNT min=1 max=400
    mmult2: for (int j = 0; j < dim ; j++) {
#pragma HLS PIPELINE
#pragma HLS LOOP_TRIPCOUNT min=1 max=400
        int result = 0;
        mmult3: for (int k = 0; k < DATA_SIZE; k++) {
#pragma HLS LOOP_TRIPCOUNT min=1 max=400
            result += A[i][k] * B[k][j];
        }
        out[i * dim + j] = result;
    }
}
}
}

```

Figure 8: The SW source code

The generated HW design is interfaced to the modified user C++ source code. SW is compiled into *te30\_1.elf* file to run as process in user space of the Debian OS with the Petalinux 2018.2 kernel on the Zynq Ultrascale+ board.

The design includes the two Vivado HLS HW accelerators for matrix (400x400 int32) multiplication and for matrix (400x400 int32) addition. Both accelerators operate at 187.5 MHz system clock. Both accelerators are directly connected in HW and complemented with automatically instantiated DMA data-movers.

The corresponding bitstream has been compiled to the *BOOT.BIN* file and the modified SW for the application *te30\_1.elf* file. The generated HW respects the initial board support package constrains and fits to the Zynq Ultrascale+ TE0808-03-15EG-1EE module.



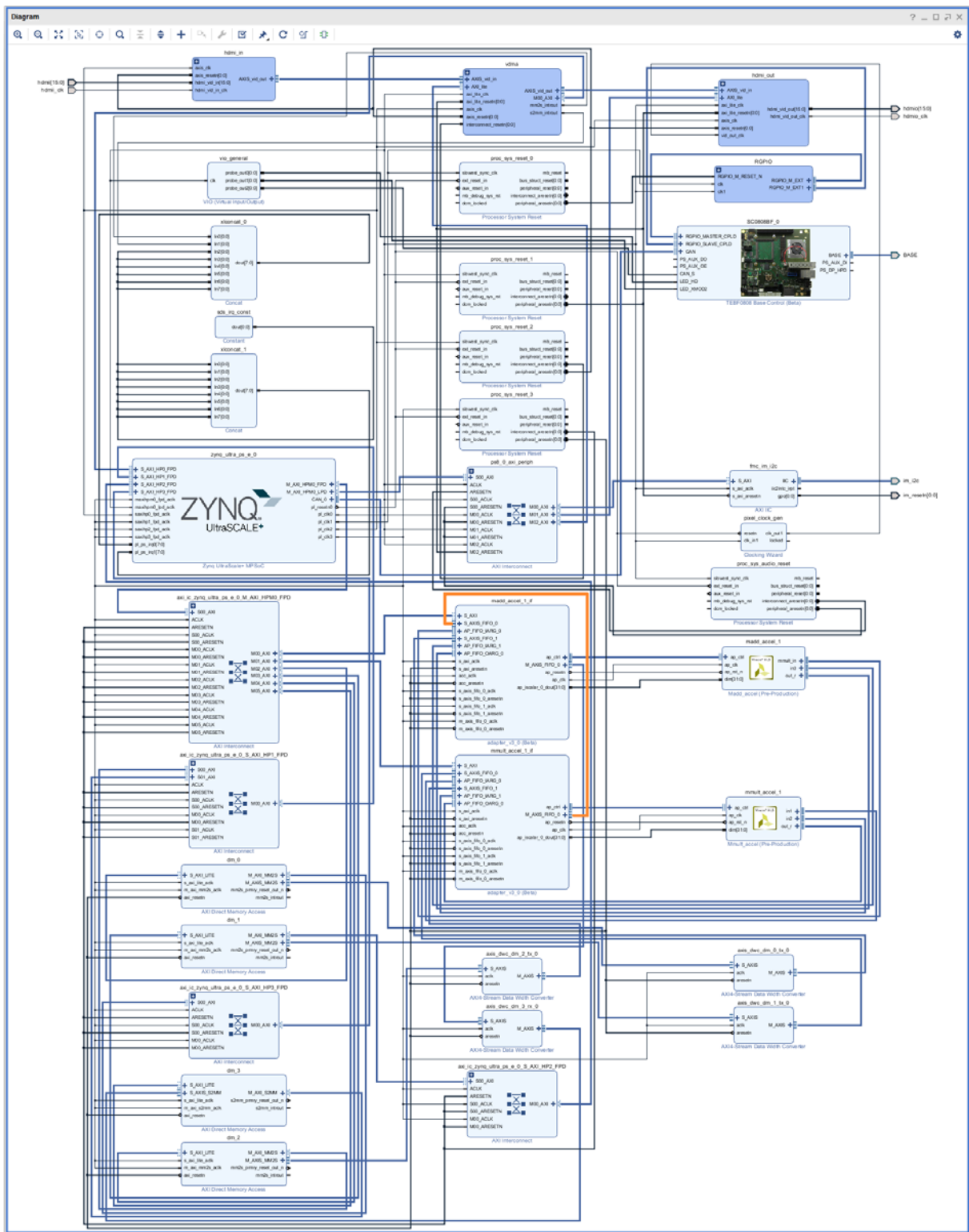


Figure 9: HW generated by the SDSoC 2018.2 compiler for matrix mult and add example

```
root@zynqmp: /boot
Speed up: 282.854
Note: Speed up is meaningful for real hardware execution only, not for emulation
.
TEST PASSED
root@zynqmp:/boot# ./te30_1.elf
Number of average CPU cycles running application in software: 560794051
Number of average CPU cycles running application in hardware: 1983230
Speed up: 282.768
Note: Speed up is meaningful for real hardware execution only, not for emulation
.
TEST PASSED
root@zynqmp:/boot# ./te30_1.elf
Number of average CPU cycles running application in software: 561277416
Number of average CPU cycles running application in hardware: 1985341
Speed up: 282.711
Note: Speed up is meaningful for real hardware execution only, not for emulation
.
TEST PASSED
root@zynqmp:/boot# ./te30_1.elf
Number of average CPU cycles running application in software: 561119853
Number of average CPU cycles running application in hardware: 1985940
Speed up: 282.546
Note: Speed up is meaningful for real hardware execution only, not for emulation
.
TEST PASSED
root@zynqmp:/boot# █
```

Figure 10: HW accelerated matrix multiplication and add

The measured HW acceleration is **282x** in comparison to the optimized SW computation on the 1.05 GHz Arm A53 processor. See Figure 10.

## 7 Video processing demo with Full HD HDMI Video In/Out

The complete demo performing video processing with HW acceleration is described in this section. We demonstrate the LK Dense Optical Flow (LK DOF) algorithm with Full HD HDMI video input and video output.

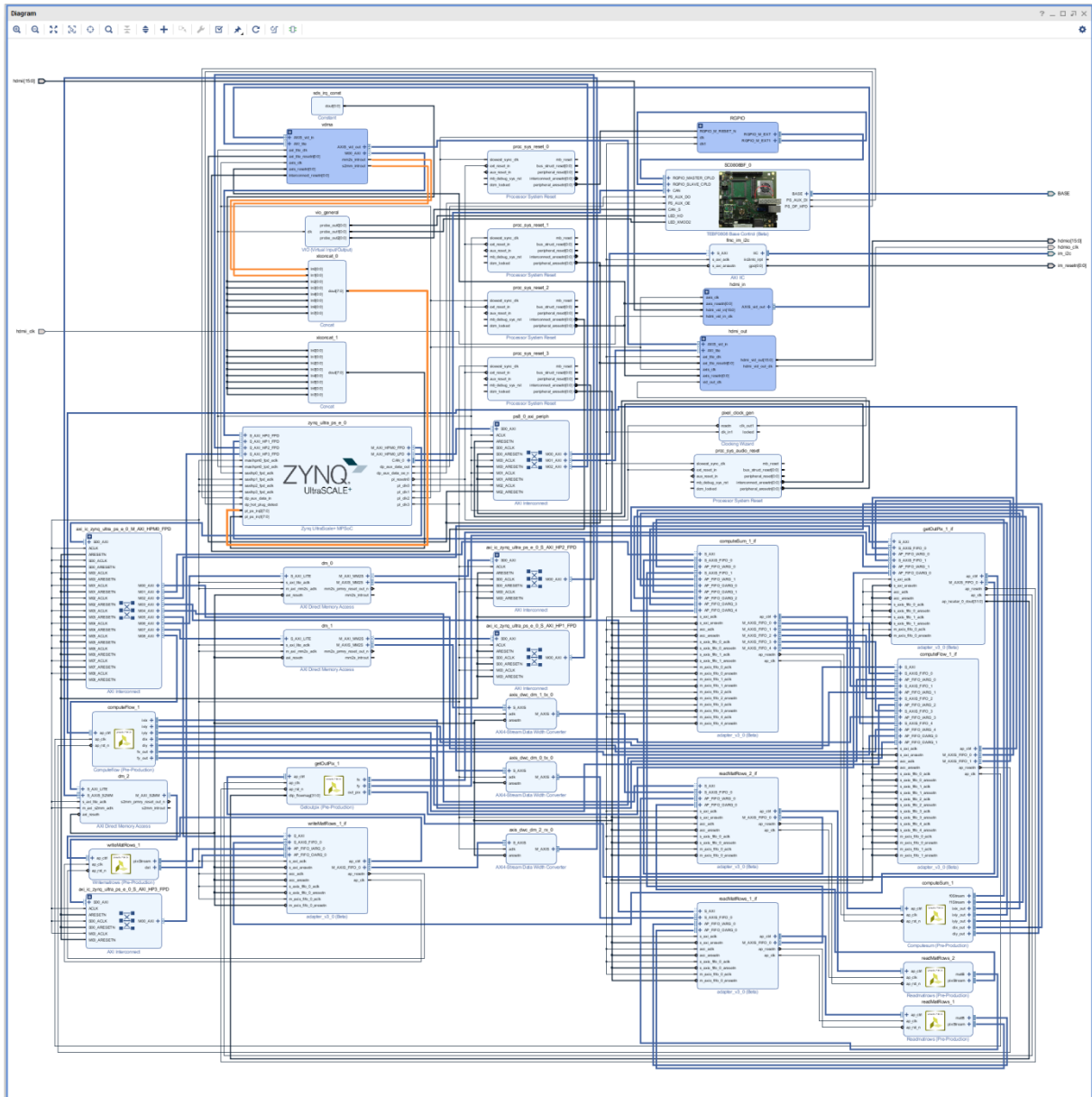
The algorithm works with two subsequent Full HD frames. It computes for each pixel of the frame vector characterizing the direction and the speed of movement of a given pixel relative to its background.

The LK Dense Optical Flow algorithm involves massive fixed point computation and also floating point matrix inversion computed for each pixel of the frame.

The fixed point moving sum of the pixel background is computed for a square area 53x53 pixel. Computation is performed for each pixel of each video frame.

Figure 11 presents HW implementation generated by the SDSoC 2018.2 compiler from C++ algorithm definition SW with standard DMA engines for video In/Out data transfer. Two DMA engines serve for parallel read of two subsequent video frames from the DDR4 video frame buffers.

The third DMA serves for writing of resulting frames with movement vectors to the DDR4 video frame buffer for the display of results. All three DMA engines use pooling and therefore one of A53 is busy. Alternative SG DMA design works with interrupt based drivers and therefore the used Arm 53 (one of 4 cores) is not 100% busy by the pooling. See *Figure 11*. The highlighted interrupt lines are connected to the Video DMA engines. The VDMA is part of the initial platform and serves for the Full HD HDMI 60fps video in and the Full HD HDMI 60fps video out via the Imageon FMC card.



*Figure 11: LK DOF in HW with standard DMAs. Full HD HDMI video I/O*

The Display Port HW support is instantiated in the ZYNQ Ultrascale+ module. It is used for the Full HD 60 fps Debian desk-top.



Figure 12: LK Dense Optical Flow input movie Full HD HDMI video 60fps



Figure 13: HW accelerated LK DOF input/output Full HD HDMI 60 fps

Figure 12 and Figure 13 present set-up for computation of the LK Dense Optical Flow input movie with Full HD HDMI input 60 FPS from the PC and output in Full HD HDMI to the HDMI monitor. See Table 2 summarizing the performance of HW accelerated implementation and also the load of two most utilized Arm A53 processors.

Table 2: Performance of HW accelerated LK FOF and load Arm A53 processors.

LK DOF algorithm with per pixel integral tile size [53x53]	Frames per second	A53 SDSoC CPU load	A53 DeskTop CPU load	Acceleration of LK DOF
In SW	0.0907	100%	3%	<b>1x</b>
In HW DMA	60	100%	80%	<b>661x</b>
In HW SG DMA	60	30%	80%	<b>661x</b>

The C++ source code of the used LK Dense Optical Flow algorithm SW is in these folders:

```
X:\4EV-1EA\zusys\samples\optical_flow_dma\  
X:\4EV-1EA\zusys\samples\optical_flow_sgdma\  
X:\4EV-1EA\zusys\samples\optical_flow_sw\  

```

This ends short presentation of the HW acceleration of relatively complex video processing algorithm with HW acceleration **661x** over the same algorithm implemented on 1.05 GHz Arm A53 processor. This acceleration is reached with design using only:

- 15% of BRAM (block rams) of the PL logic and no ULTRA RAM)
- 14% of CLB (logic block tiles)
- 2% of DSP resources

This indicates the potential of the large TE0808-03-15EG-1EE module in the area of video processing algorithm acceleration.

## 8 Inter-cloud connectivity based on the Arrowhead framework

The FitOptiVis (WP4) run-time resources are supported for the Zynq Ultrascale+ module TE0808-03-15EG-1EE by SW implementation of the Arrowhead framework compatible clients on the 64 bit Arm Cortex A53 processor. The Arrowhead framework [3] has been developed within ECSEL Arrowhead project and Productive4.0 projects <https://productive40.eu/>.

In FitOptiVis WP4, we support as an SW design time resource the Arrowhead framework for board to board Ethernet communication in the local cloud.

The Arrowhead framework works on one RaspberryPi 3B (RPi3) board. The RPi3 implements the Arrowhead framework as set of Java services. See documentation in [3]. The Zynq Ultrascale+ module TE0808-03-15EG-1EE hosts C++ provider capable to measure the actual temperature of the Xilinx XCZU4EV-1SFVC784E device. The Zynq Ultrascale+ in module can also hosts C++ Consumer application capable to ask the Arrowhead framework about the temperature provided as service by the producer service running as separate process on the Zynq Ultrascale+ module.

## 9 Installation of Arrowhead Framework Services on RPi3

The Arrowhead client SW acts as the *Producer* providing a service or as a *Consumer* requesting the service via the Arrowhead framework. The base hardware platform for the Zynq Ultrascale+ module is compiled as described in Chapter 2 - 6.

Testing and running of the Arrowhead C++ clients on Zynq Ultrascale+ boards requires Ethernet access to the Arrowhead framework services. It is recommended to use the precompiled image for the RPi3 board. It includes already installed and configured Arrowhead framework G4.0 lightweight implementation. The image is available as one of results of the work package WP1 of the running ECSEL JU project Productive4.0 <https://productive40.eu/>. It is accessible for all Productive4.0 consortium project partners. Please contact coordinator of the consortium for further information about the access to the Arrowhead-framework G4.0 light-weight installation running on the RPi3 board. After receiving the access to the download, unzip the three downloaded files *Arrowhead-40-raspi.z01*, *Arrowhead-40-raspi.z02* and *Arrowhead-40-raspi.zip* into the final image file *image\_180626.img* (size 3.711.959.040 Bytes).

Copy the RPi3 image *image\_180626.img* to (at least) 4GB SD card (speed grade 10). You can use the *Win32DiskImager* utility from: <https://sourceforge.net/projects/win32diskimager/>.

Connect the RPi3 to USB keyboard, HDMI monitor with inserted SD card. Connect it to Ethernet with the DHCP server. Power ON the board by connecting the 5V power supply via micro USB cable. Power can be provided from the PC via the USB port or, preferably, from the dedicated 5V power supply. Details of the installation and use are described in Chapter 8 of App. note [6].

## 10 Install Arrowhead-f support on Zynq Ultrascale+ module

At this stage, the Debian OS configured for the Zynq Ultrascale+ module TE0808-03-15EG-1EE can be upgraded to become compatible with the Arrowhead framework G4.0 client and provider C++ demo applications. Details of the installation and use are described in Chapter 9 of App. note [6].

## 11 Install Arrowhead-f C++ Provider on Zynq Ultrascale+ module

The Arrowhead *ProviderExample* can be compiled and tested on the same Zynq Ultrascale+ module. Details of the installation and use are described in Chapter 10 of App. note [6]. Start the compiled *ProviderExample*:

```
./ProviderExample
```

The *ProvidedExample* registers itself in the Arrowhead framework database running on the RPi3 board. On *Consumer* request, it returns an artificial temperature, fixed to value 26 degrees Celsius, at this first installation stage.

## 12 Install Arrowhead-f C++ Consumer on Zynq Ultrascale+ module

The Arrowhead *ConsumerExample* can be compiled and tested on the same Zynq Ultrascale+ module. Details of the installation and use are described in Chapter 11 of App. note [6]. Run the compiled *ConsumerExample*:

```
./ConsumerExample
```

The program should show the following response from the *ProviderExample*:

Provider Response:

```
{ "e": [ { "n": "this_is_the_sensor_id", "v": 26.0, "t": "1553675692" } ], "bn": "this_is_the_sensor_id", "bu": "Celsius" }
```

The *ConsumerExample* might fail in the very first instance of the Database use. The database of the Arrowhead-f running on the RPi3 has to be configured. The *ProviderExample* and the *ConsumerExample* have to be connected by the operator of the Database. Modification of the Arrowhead Database.

The Arrowhead framework running on the RPi3 board provides *phpMyAdmin* interface to control the Database. To allow the *ConsumerExample* to get the *ProducerExample* service response, follow steps described in Chapter 12 of App. note [6].

The *ConsumerExample* should get the proper response from the *ProviderExample*, now.

### 13 Test the Zynq Ultrascale+ Consumer and Producer

The *ProducerExample* server is running on the “Producer” Zynq Ultrascale+ module.

Execute the *ConsumerExample* client example on the “Consumer” Zynq Ultrascale+ module.

```
./ConsumerExample
```

The *ConsumerExample* client example program should show the modelled constant temperature response (26.0) from the *ProviderExample* and exit.

Provider Response:

```
{ "e": [ { "n": "this_is_the_sensor_id", "v": 26.0, "t": "1553675692" } ], "bn": "this_is_the_sensor_id", "bu": "Celsius" }
```

### 14 Producer with real temperature measurement on Zynq Ultrascale+ module

Real temperature of the Xilinx chip of the “producer” Zynq Ultrascale+ module can be measured by modified *ProviderExample.cpp* code.

This is modified source code of the *ProviderExample.cpp* code. It measures and provides the temperature of the Zynq Ultrascale+ chip to the Arrowhead framework:

```
#pragma warning(disable:4996)
#include "SensorHandler.h"
#include <sstream>
#include <string>
#include <stdio.h>
#include <thread>
#include <list>
#include <time.h>
#include <iomanip>
#ifdef __linux__
    #include <unistd.h>
```

```

#elif _WIN32
    #include <windows.h>
#endif

#define TEMP_RAW_FILE
"/sys/bus/iio/devices/iio:device0/in_temp0_ps_temp_raw"
#define TEMP_OFFSET_FILE
"/sys/bus/iio/devices/iio:device0/in_temp0_ps_temp_offset"
#define TEMP_SCALE_FILE
"/sys/bus/iio/devices/iio:device0/in_temp0_ps_temp_scale"
bool bSecureProviderInterface = false; //Enables HTTPS interface on the
application service (with token enabled)
bool bSecureArrowheadInterface = false; //Enables HTTPS interface towards
ServiceRegistry AH module
inline void parseArguments(int argc, char* argv[]){
    for(int i=1; i<argc; ++i){
        if(strstr("--secureArrowheadInterface", argv[i]))
            bSecureArrowheadInterface = true;
        else if(strstr("--secureProviderInterface", argv[i]))
            bSecureProviderInterface = true;
    }
}
int main(int argc, char* argv[]){

    printf("\n=====\nProvider Example -
v%s\n=====\n", version.c_str());
    parseArguments(argc, argv);
    SensorHandler oSensorHandler;
    std::string measuredValue; //JSON - SENML format
    time_t linuxEpochTime = std::time(0);
    std::string sLinuxEpoch = std::to_string((uint64_t)linuxEpochTime);
    FILE *f_t_raw, *f_t_off, *f_t_scale;
    if ( (f_t_raw = fopen(TEMP_RAW_FILE, "r")) == NULL ) {
        printf("Cannot open file %s \n", TEMP_RAW_FILE);
        return -1;
    }
    if ( (f_t_off = fopen(TEMP_OFFSET_FILE, "r")) == NULL ) {
        printf("Cannot open file %s \n", TEMP_OFFSET_FILE);
        return -1;
    }
    if ( (f_t_scale = fopen(TEMP_SCALE_FILE, "r")) == NULL ) {
        printf("Cannot open file %s \n", TEMP_SCALE_FILE);
        return -1;
    }
    printf("OK\n");
    int t_raw;
    int t_off;
    float t_scale;
    fscanf(f_t_raw, "%d", &t_raw);

```



```

fscanf(f_t_off, "%d", &t_off);
fscanf(f_t_scale, "%f", &t_scale);
if ( fclose(f_t_raw) == EOF ) {
    printf("Cannot close file %s \n", TEMP_RAW_FILE);
    return -1;
}
printf("OK\n");
if ( fclose(f_t_off) == EOF ) {
    printf("Cannot close file %s \n", TEMP_OFFSET_FILE);
    return -1;
}
if ( fclose(f_t_scale) == EOF ) {
    printf("Cannot close file %s \n", TEMP_SCALE_FILE);
    return -1;
}
float value = ((float)(t_raw + t_off) * t_scale) / 1000.00f;
std::ostringstream streamObj;
streamObj << std::fixed;
streamObj << std::setprecision(1);
streamObj << value;
std::string sValue = streamObj.str();
measuredValue =
    "{"
        "\ne\":[{"
            "\n": \"this_is_the_sensor_id\",
            "\nv\":" + sValue + ",
            "\nt\":" + sLinuxEpoch + "\"
            "}]",
        "\nb\":" + \"this_is_the_sensor_id\",
        "\nbu\":" + \"Celsius\"
    "};
oSensorHandler.processProvider(
    measuredValue, bSecureProviderInterface, bSecureArrowheadInterface);
while (true) {
    linuxEpochTime = std::time(0);
    sLinuxEpoch = std::to_string((uint64_t)linuxEpochTime);
    if ( (f_t_raw = fopen(TEMP_RAW_FILE, "r")) == NULL ) {
        printf("Cannot open file %s \n", TEMP_RAW_FILE);
        return -1;
    }
    fscanf(f_t_raw, "%d", &t_raw);
    if ( fclose(f_t_raw) == EOF ) {
        printf("Cannot close file %s \n", TEMP_RAW_FILE);
        return -1;
    }
    value = ((float)(t_raw + t_off) * t_scale) / 1000.00f;
    printf("Zynq Temp : %f °C\n", value);
    streamObj.clear();
}

```

```

    streamObj.str("");
    streamObj << std::fixed;
    streamObj << std::setprecision(1);
    streamObj << value;
    sValue = streamObj.str();
    measuredValue =
        "{"
        "\ne\":[{"
        "\n\": \"this_is_the_sensor_id\","
        "\nv\":" + sValue + ","
        "\nt\": \"\" + sLinuxEpoch + "\""
        "}],{"
        "\bn\": \"this_is_the_sensor_id\","
        "\bu\": \"Celsius\""
        "}";
    oSensorHandler.processProvider(
        measuredValue, bSecureProviderInterface, bSecureArrowheadInterface);
#ifdef __linux__
    sleep(1);
#elif _WIN32
    Sleep(1000);
#endif
}
printf("Close file %s ... ", TEMP_RAW_FILE);
if ( fclose(f_t_raw) == EOF ) {
    printf("FAILED\n");
    return -1;
}
printf("OK\n");
return 0;
}

```

All other files remain identical. Recompile the *ProviderExample* project by *make*.

Test the real temperature measurement compatible with the Arrowhead framework on the Zynq Ultrascale+ module. Consumer can run on the same module as separate Debian application or it can run on a ZynqBerry board connected to the local cloud as described in the App. note [6].

```

root@zynqmp: ~/arrowheadclient/ArrowheadCpp/ProviderExample
LastValue updated.
Zynq Temp : 40.101017 Å°C
New measurement received from: this_is_the_sensor_id
LastValue updated.
MHD_Callback
MHD_Callback

HTTP GET request received
Received URL: /this_is_the_custom_url
Response:
{"e":[{"n": "this_is_the_sensor_id","v":40.1,"t": "1555067891"}],"bn": "this_is_the_sensor_id","bu": "Celsius"}

Zynq Temp : 40.847080 Å°C
New measurement received from: this_is_the_sensor_id
LastValue updated.
Zynq Temp : 40.722740 Å°C
New measurement received from: this_is_the_sensor_id
LastValue updated.
Zynq Temp : 40.380795 Å°C
New measurement received from: this_is_the_sensor_id
LastValue updated.
^C
root@zynqmp:~/arrowheadclient/ArrowheadCpp/ProviderExample# ^C

```

Figure 14: Provider of the temperature of the Zynq Ultrascale+chip, response to requests.

```

root@zynqmp: ~/arrowheadclient/ArrowheadCpp/ConsumerExample
ConsumedServiceTable
-----
TestconsumerID : {"orchestrationFlags":{"externalServiceRequest":false,"matchmaking":true,"metadataSearch":false,"onlyPreferred":true,"overrideStore":true,"pingProviders":false},"preferredProviders":[{"providerSystem":{"address":"192.168.13.232","port":"8000","systemName":"SecureTemperatureSensor"}}], "requestedService":{"interfaces":["REST-JSON-SENML"],"serviceDefinition":"IndoorTemperature_ProviderExample","serviceMetadata":{"security":""},"requesterSystem":{"address":"dont care","authenticationInfo":"null","port":8002,"systemName":"client1"}}

Failed to bind to port 8002: Address already in use
Error: Unable to start HTTP Server (192.168.13.232:8002)!
Error: Unable to start Orchestrator Interface!
consumerID: TestconsumerID
Sending Orchestration Request: (Insecure Arrowhead Interface)

sendHttpRequestToProvider

Provider Response:
{"e":[{"n": "this_is_the_sensor_id","v":40.1,"t": "1555067891"}],"bn": "this_is_the_sensor_id","bu": "Celsius"}

Done.

```

Figure 15: Consumer output of the temperature of the Zynq Ultrascale+ chip

## 15 Package content

```
├── debian
│   ├── mkdebian.sh
│   ├── image.ub
│   ├── u-boot.elf
│   └── bl31.elf
└── zynq
    ├── TE0808_SDSoc_IMAGEON_FMC_HDMI_701HDMI.zip
    └── install-arrohead-cli-dep.sh
```

## References

- [1] Trenz Electronic, "UltraSOM+ MPSoC Module with Zynq UltraScale+ XCZU15EG-1FFVC900E, 4 GB DDR4", [Online].  
<https://shop.trenz-electronic.de/en/TE0808-04-15EG-1EE-UltraSOM-MPSoC-Module-with-Zynq-UltraScale-XCZU15EG-1FFVC900E-4-GB-DDR4?c=450>
- [2] Trenz Electronic, "TE0726 TRM," [Online].  
<https://shop.trenz-electronic.de/en/27229-Bundle-ZynqBerry-512-MByte-DDR3L-and-SDSoC-Voucher?c=350>
- [3] Documents for Arrowhead Framework  
Available:[https://forge.soa4d.org/docman/?group\\_id=58](https://forge.soa4d.org/docman/?group_id=58)
- [4] Jiří Kadlec, Zdeněk Pohl, Lukáš Kohout: Design Time and Run Time Resources for the ZynqBerry Board TE0726-03M with SDSoC 2018.2 Support. UTIA application note. [Online]. <http://sp.utia.cz/index.php?ids=projects/fitoptivis>
- [5] UltraTX+ Baseboard for Trenz Electronic TE080X UltraSOM+ [Online].  
<https://shop.trenz-electronic.de/en/TEBF0808-04-UltraTX-Baseboard-for-Trenz-Electronic-TE080X-UltraSOM?c=261>

## 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.