Application Note

Akademie věd České republiky
Ústav teorie informace a automatizace AV ČR, v.v.i.

Design Time and Run Time Resources for Zynq Ultrascale+ TE0820-03-4EV-1E with SDSoC 2018.2 Support

Jiři Kadlec, Zdeněk Pohl, Lukáš Kohout
kadlec@utia.cas.cz xpohl@utia.cas.cz kohoutl@utia.cas.c

Revision history

Rev.	Date	Author	Description
0	10.04.2019	J. Kadlec	Initial draft
1	11.04.2019	J.Kadlec	Improved chapters 9-15 (Provider/Consumer)
2			

Table of Contents

Table of Figures

Table of Tables

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+ TE0820-03-4EV-1E [1]. It works with Xilinx XCZU4EV-1SFVC784E device with the quad core Arm A53 64 bit, dual Arm Cortex R5 and programmable logic area on single 16nm chip. See *Figure 1*.

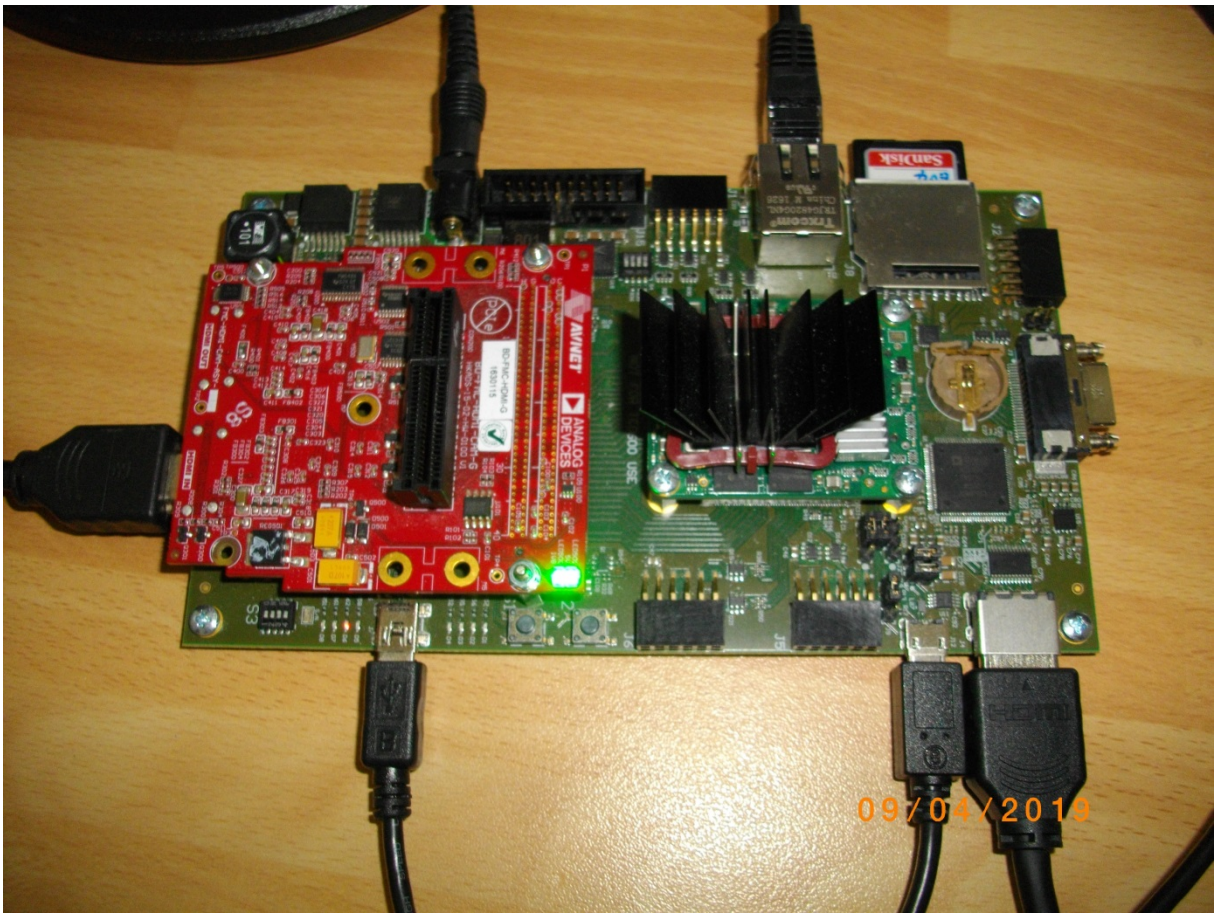


Figure 1: TE0820-03-4EV-1E on TE0701-06 carrier with Imageon HDMI I/O FMC card

The Zynq Ultrascale+ PCB module has the 4x5cm form factor. The Zynq Ultrascale+ board is designed and manufactured by company Trenz Electronic [1].

Figure 2: The Zynq Ultrascale+ TE0820-03-4EV-1E 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. 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 3*.
The resulting compiled system remains compatible with the FitOptiVis run time resources – the 64bit Debian OS and with the local cloud Ethernet communication of C++ clients via the Arrowhead framework (result of ECSEL Productive 4.0 project) [2].

The initial hardware platform is compiled with Xilinx SDSoC 2018.2 tool. The design is based on a board support package provided by Trenz Electronic for the Zynq Ultrascale+ board. You have to have the Xilinx SDSoC 2018.2 installed 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 SDSoC 2018.2 license voucher can be purchased together with TE0726-03M board as bundle: "Zynq Ultrascale+ 512 MByte DDR3L and SDSoC Voucher". See [3]: https://shop.trenz-electronic.de/en/27229-Bundle-ZynqBerry-512-MByte-DDR3L-and-SDSoC-Voucher?c=350 The voucher supports compilation of designs for the Zynq Ultrascale+ TE0820-03-4EV-1E chip.

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 TE0820-03-4EV-1E module on TE0701-06 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 TE0820-03-4EV-1E module on TE0701-06 carrier board 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 TE0820-03-4EV-1E module on TE0701-06 carrier board with Video I/O follow these steps:

1. Unpack the enclosed evaluation package
   TE0820_SDSoC_IMAGEON_FMC_HDMI_701HDMI.zip
   to Win 7 or Win10 directory of your choice. We will use:
   ```
   c:\TS82\TE0820_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\TE0820_SDSoC_IMAGEON_FMC_HDMI_701HDMI\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/TE0820_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_guimode.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 200 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 thre 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\bl31.elf
```

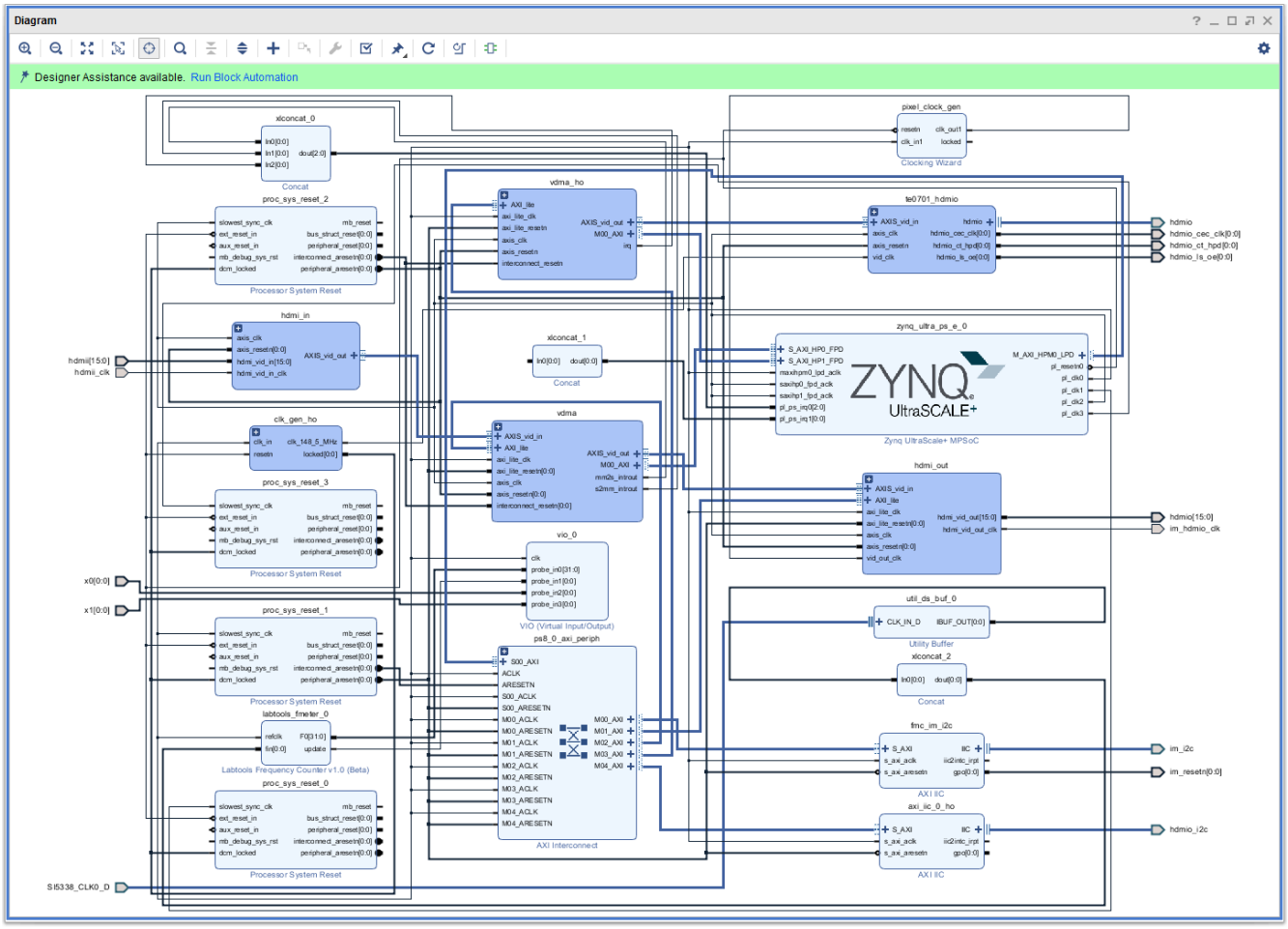


Figure 3: The initial Vivado design. It defines the SDSoC 2018.2 platform.

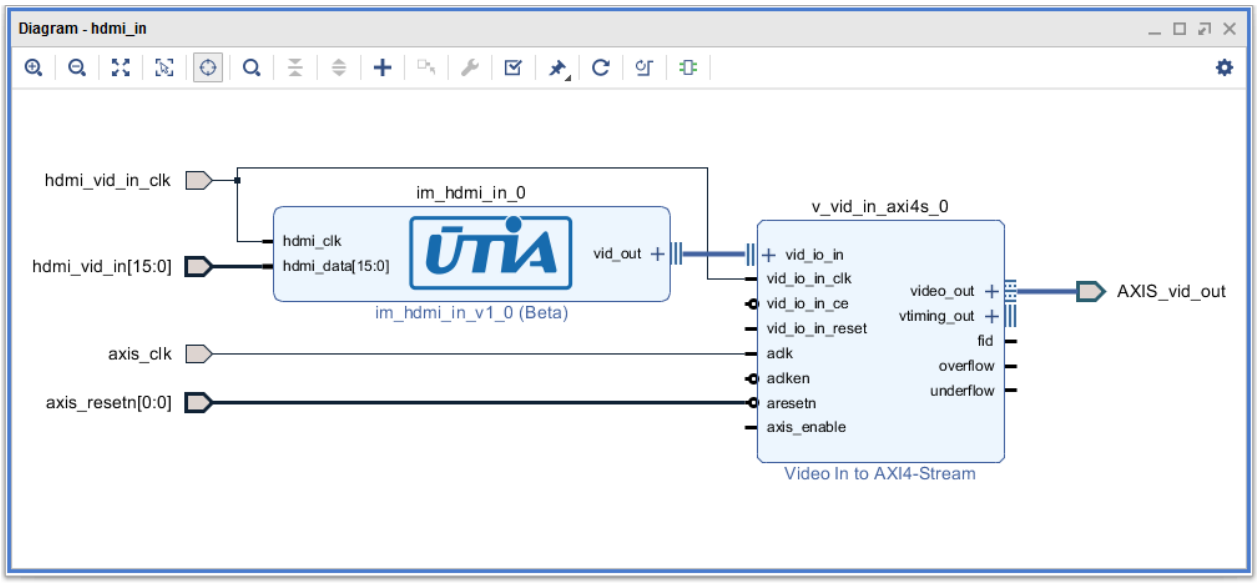


Figure 4: hdmi_in serves for input of Full HD HDMI from camera via Imageon FMC

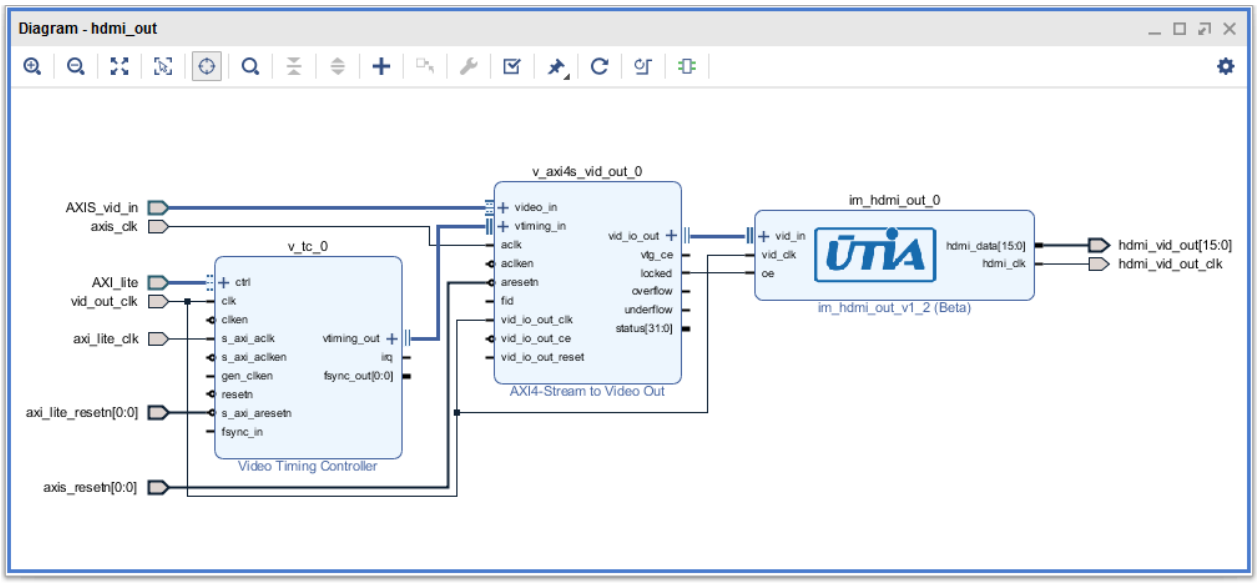


Figure 5: hdmi_out serves for output of Full HD HDMI to display via Imageon FMC

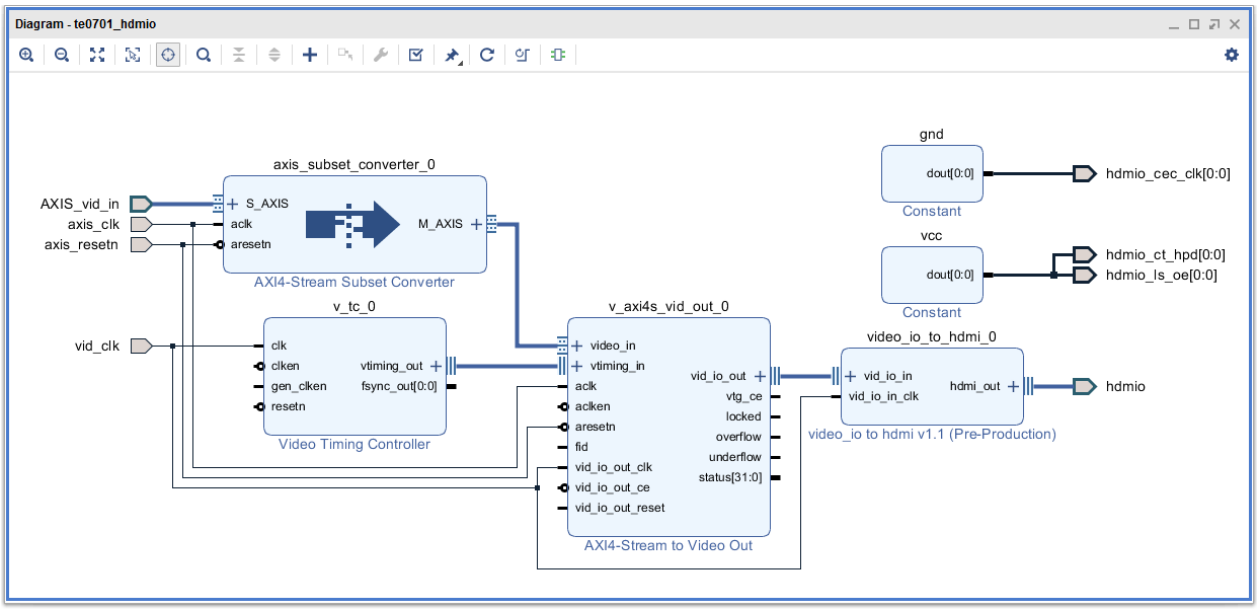


Figure 6: te0701_hdmio serves for output of Full HD HDMI display via TE0701 board

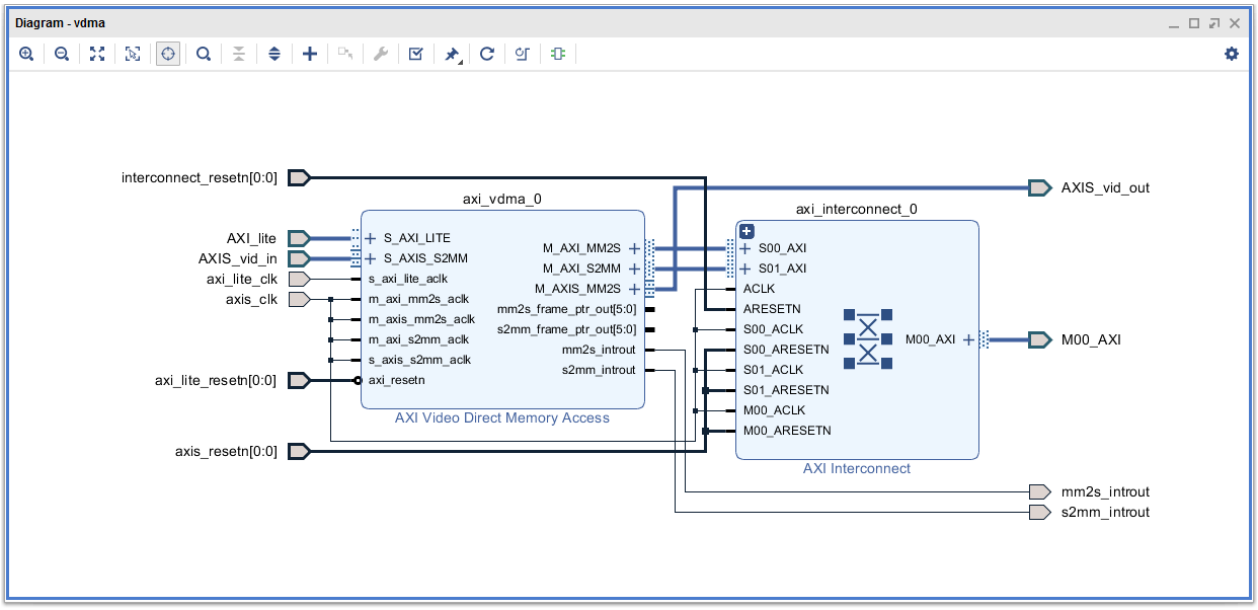


Figure 7: vdma serves for video dma in/out to/from 8 Full HD video frame buffers in DDR4

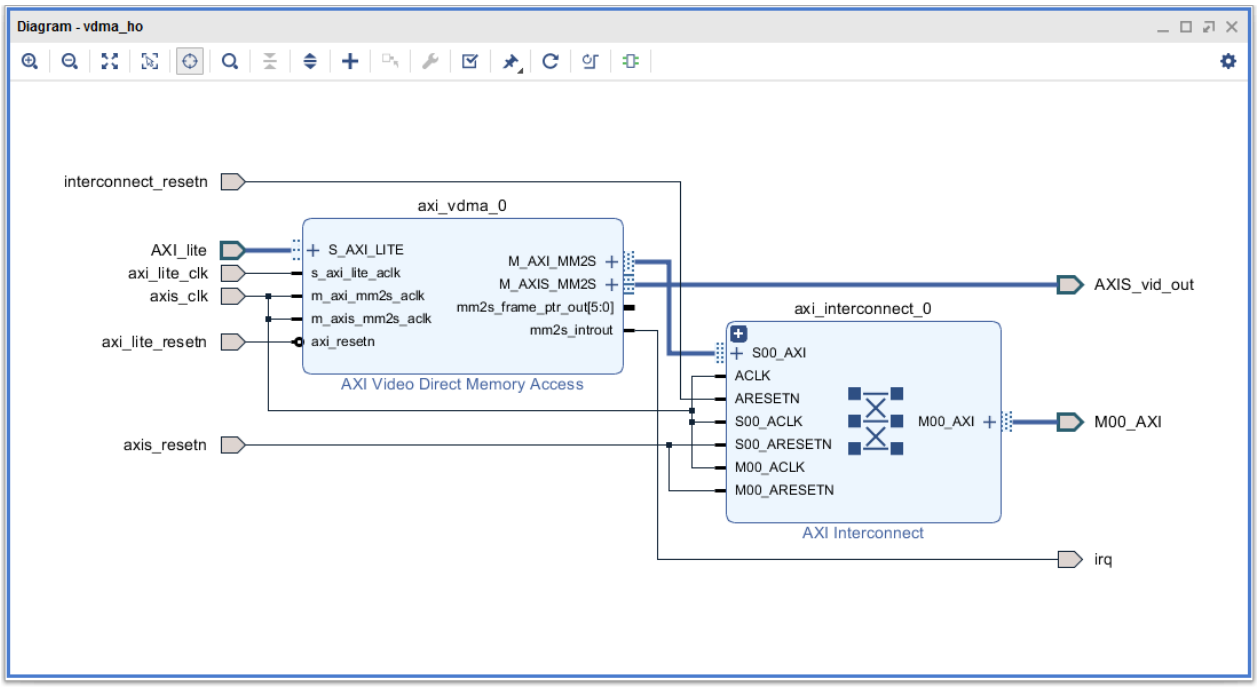


Figure 8: vdma_ho serves for video dma out from one Full HD video frame buffer in DDR4

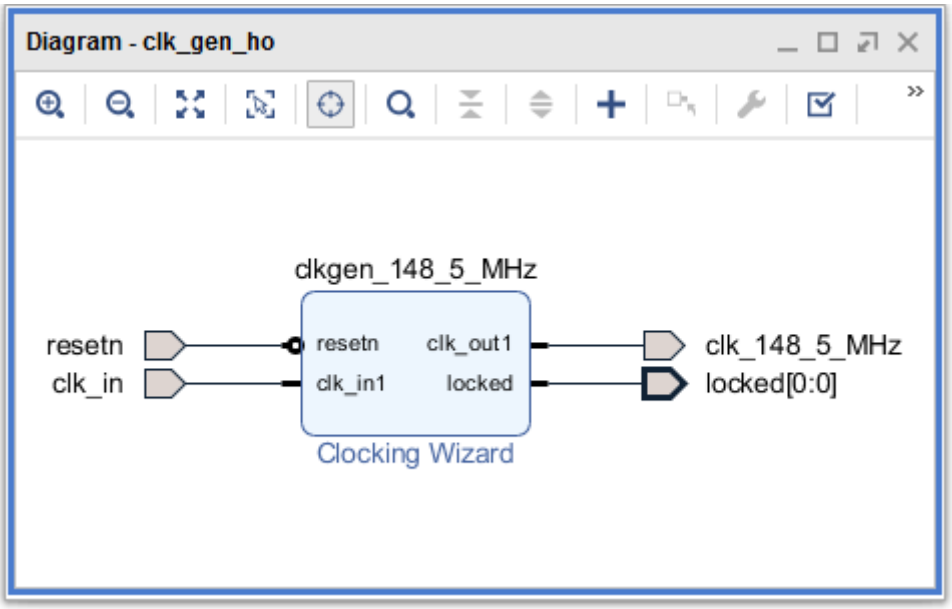


Figure 9: clk_gen_ho serves for generation of fixed clock for the Full HD video output.

The hierarchical blocks of *Figure 3* described in *Figure 4* - *Figure 9* 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 *PetalLinux 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 TE0820-03-4EV-1E 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/TE0820/zusys/os
   /home/devel/work/TS82/TE0820/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 TE0820-03-4EV-1E:

   ```
   cd /home/devel/work/TS82/TE0820/zusys/os/petalinux
   ```

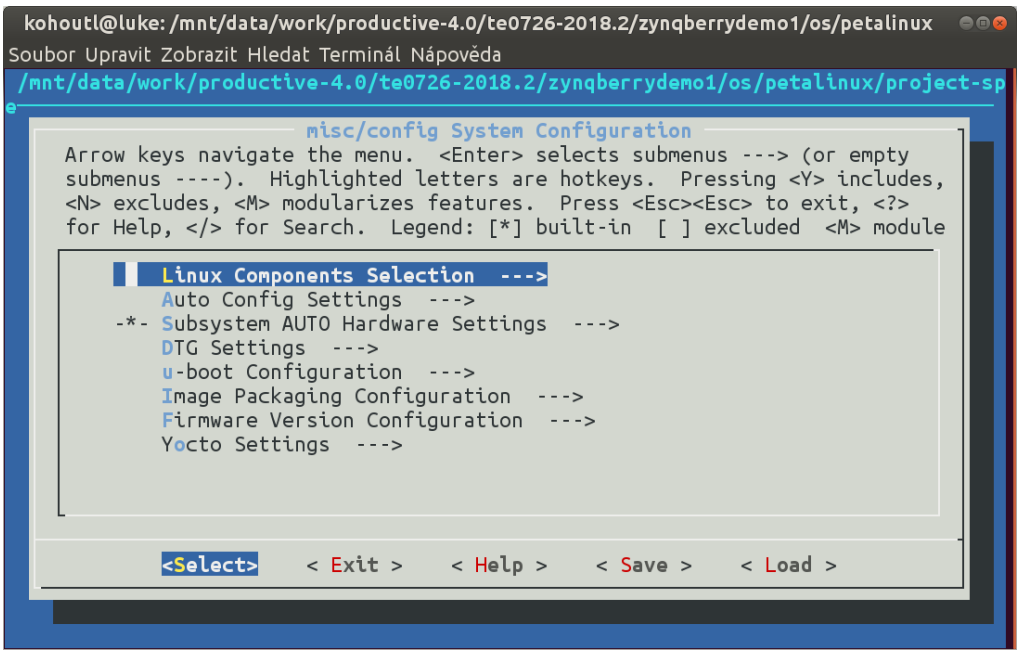
   It contains a predefined configuration according to Zynq Ultrascale+ board requirements.

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/TE0820/zusys/prebuilt/hardware/4ev_1e/zusys.hdf
   ```

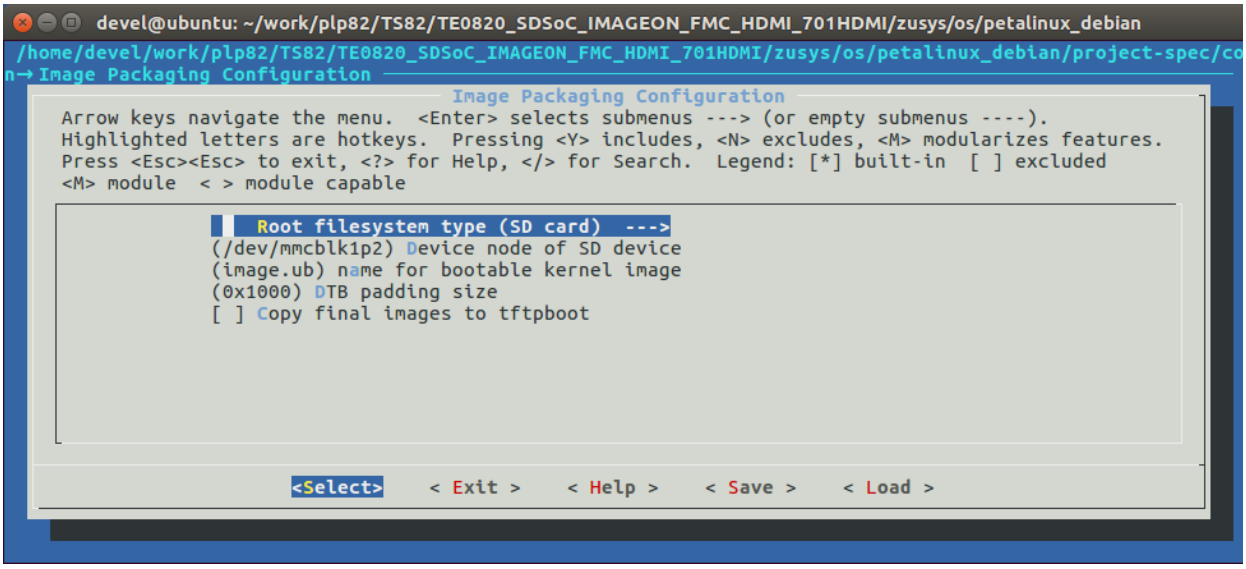
5. Load the HDF to current PetaLinux configuration by command (on single line)

   ```
   petalinux-config --get-hw-description=/home/devel/work/TS82/TE0820/
   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. Build PetaLinux, from the bash terminal execute

```
petalinux-build
```

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

```
/home/devel/work/TS82/TE0820/zusys/os/petalinux/images/linux/image.ub
/home/devel/work/TS82/TE0820/zusys/os/petalinux/images/linux/u-boot.elf
/home/devel/work/TS82/TE0820/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/TE0820/zusys/os/petalinux/mkdebian.sh
   ```
2. Go to the folder with PetaLinux:
   ```
   cd /home/devel/work/TS82/TE0820/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.

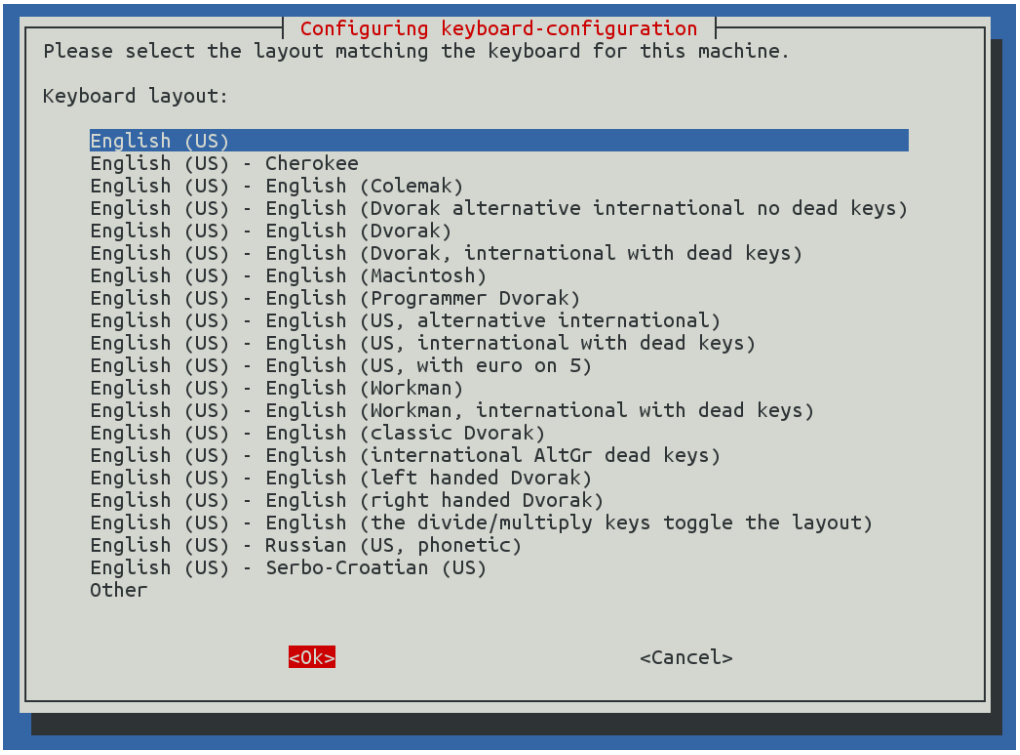
   When some of them are missing, install them by:
   ```
   sudo apt install Package
   ```

   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 *te0820-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 te0820-debian.zip:

```
zip te0820-debian te0820-debian.img
```

6. Copy compressed image file from Ubuntu

```
/home/devel/work/TS82/TE0820/zusys/os/petalinux/te0820-debian.zip
```

to Win7 or Win 10 file:

```
X:\zusys\prebuilt\os\petalinux\default\te0820-debian.zip
```

7. Copy from Ubuntu

```
/home/devel/work/TS82/TE0820/zusys/os/petalinux/images/linux/image.ub
/home/devel/work/TS82/TE0820/zusys/os/petalinux/images/linux/u-boot.elf
/home/devel/work/TS82/TE0820/zusys/os/petalinux/images/linux/bl31.elf
```

to Win7 or Win 10 file:

```
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/TE0820/zusys/os/petalinux/te0820-debian.zip
/home/devel/work/TS82/TE0820/zusys/os/petalinux/te0820-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 *te0726-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\4ev_1e\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\TE0820-03\4EV-1EA`
   and it is also packed into the ZIP file.

This ends the configuration and compilation steps needed for the initial generation of the SDSoC 2018.2 platform for the TE0820-03-4EV-1EA module on the TE0701-06 carrier.

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

6 Compile HW accelerator by the SDSoC 2018.2 compiler

Compilation and test of simple matrix multiplication and addition SDSoC 2018.2 sample application is described in this section.

1. On Win 7 or Win10, i*n the open dos terminal window, cancel the current virtual drive X: by executing from* the command line
   ```
   _use_virtual_drive.cmd
   ```
   and response (1)
2. Change directory to
   c:\TS82\TE0820\TE0820_SDSoC_IMAGEON_FMC_HDMI_701HDMI\SDSoC_PFM\TE0820-03\4EV-1EA\
3. On Win 7 or Win10, open dos terminal window and use the copy of the script *_use_virtual_drive.cmd* to create a new virtual short path to get short SDSoC directory *X:\4EV-1EA*
   ```
   _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 4EV-1EA
   ```

4. Open SDSoC project in directory
   ```
   X:\4EV-1EA
   ```
5. In SDSoC select platform:
   ```
   X:\SDSoC_PFM\te0726\03m\zusys
   ```
6. Create new project named
   ```
   te30_l
   ```
7. Select template project
   ```
   X:\4EV-1EA\zusys\samples\z_is_a_times_b_direct_connect\
   ```
   and compile it for the *Release* target with all clocks set to 200 MHz.

. This example will accelerates int32 matrix operation:

   D[75,75] = A[75,75] * B[75,75] + C[75,75]

   in the programmable logic of the Zynq Ultrascale+ module.

8. The SDSoC compiler will create these relevant results in the *sd_card* directory:
   ```
   X:\4EV-1EA\te30_l\Release\sd_card\BOOT.BIN
   X:\4EV-1EA\te30_l\Release\sd_card\te30_l.elf
   ```
9. Unzip the preconfigured and precompiled Debian image for the Zynq Ultrascale+ board from this evaluation package file: *te0820-debian.zip* to the file *te0820-debian.img*.
10. Use the *Win32DiskImager* https://sourceforge.net/projects/win32diskimager/ for creation of the image *te0820-debian.img* on the SD card. Use 8GB SD with speed grade 10.
11. Copy to the root of the SD card the HW accelerated matrix multiplication demo executable *te30_l.elf* from the directory:
    ```
    X:\SDSoC_PFM\TE0820-03\4EV-1EA\te30_l\Release\sd_card\te30_l.elf
    ```
12. Insert created SD card to the Zynq Ultrascale+ board.
13. Connect the Zynq Ultrascale+ board to the Ethernet cable.
14. On PC, you can use the *putty* terminal (see https://www.putty.org/).
15. Connect the Zynq Ultrascale+ board with your PC via mini USB cable. The mini USB cable provides the programming interface and 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 in the Win7 or Win10 *Device manager* utility.
16. Connect the 12V power supply.
17. The Zynq Ultrascale+ board will automatically boot from 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 (size 3.926.136 bytes) from the SD card with text output to the serial terminal. The Debian file system is present on the separate partition of the SDcard.
18. Login as user:
    ```
    root
    ```
    Password:
    ```
    root
    ```
19. Find and write down the assigned Ethernet IP address for IP V4 and IP V6 by typing command:
    ```
    ifconfig
    ```

20. Start output to the Full HD HDMI monitor connected to the TE0701 carrier board by executing this support command from the SD card:
    ```
    /boot/hdmi_start.elf
    ```
21. The full screen text console is open on the Full HD HDMI monitor connected to the DE0701 carrier board. Use the USB keyboard and login as:
    ```
    root
    ```
    Password:
    ```
    root
    ```
    The Debian top will open automatically on the Full HD HDMI monitor connected to the TE0701 carrier board. The USB keyboard and the USB mouse can be used.
22. The HW accelerated matrix multiplication demo can be executed on both Zynq Ultrascale+ boards from the automatically mounted SD by executing. See *Figure 12*:
    ```
    /boot/te06_l.elf
    ```
23. See *Figure 12*. The HW acceleration measured by the number of Arm A9 clock cycles.
24. To shut down properly the Zynq Ultrascale+ board type:
    ```
    halt
    ```
    The Debian OS is properly shut down and all possibly open R/W to the SD card are closed. Remove temporarily the SD card and disconnect the 12V power to switch OFF the board. Return back the SD card.

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[75,75] = A[75,75] * B[75,75] + C[75,75].

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=5625
        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=5625 max=5625
        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=75
        mmult2: for (int j = 0; j < dim ; j++) {
        #pragma HLS PIPELINE
        #pragma HLS LOOP_TRIPCOUNT min=1 max=75
           int result = 0;
           mmult3: for (int k = 0; k < DATA_SIZE; k++) {
           #pragma HLS LOOP_TRIPCOUNT min=1 max=75
               result += A[i][k] * B[k][j];
           }
           out[i * dim + j] = result;
        }
    }
}
```

Figure 10: The SW source code

The generated HW design is interfaced to the modified user C++ source code. SW is compiled into *te30_l.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. One for matrix (75x75 int32) multiplication and one for matrix (75x75 int32) addition. Both accelerators operate at 200 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_l.elf* file. The generated HW respects the initial board support package constrains and fits to the Zynq Ultrascale+ TE0820-03-4EV-1E module.

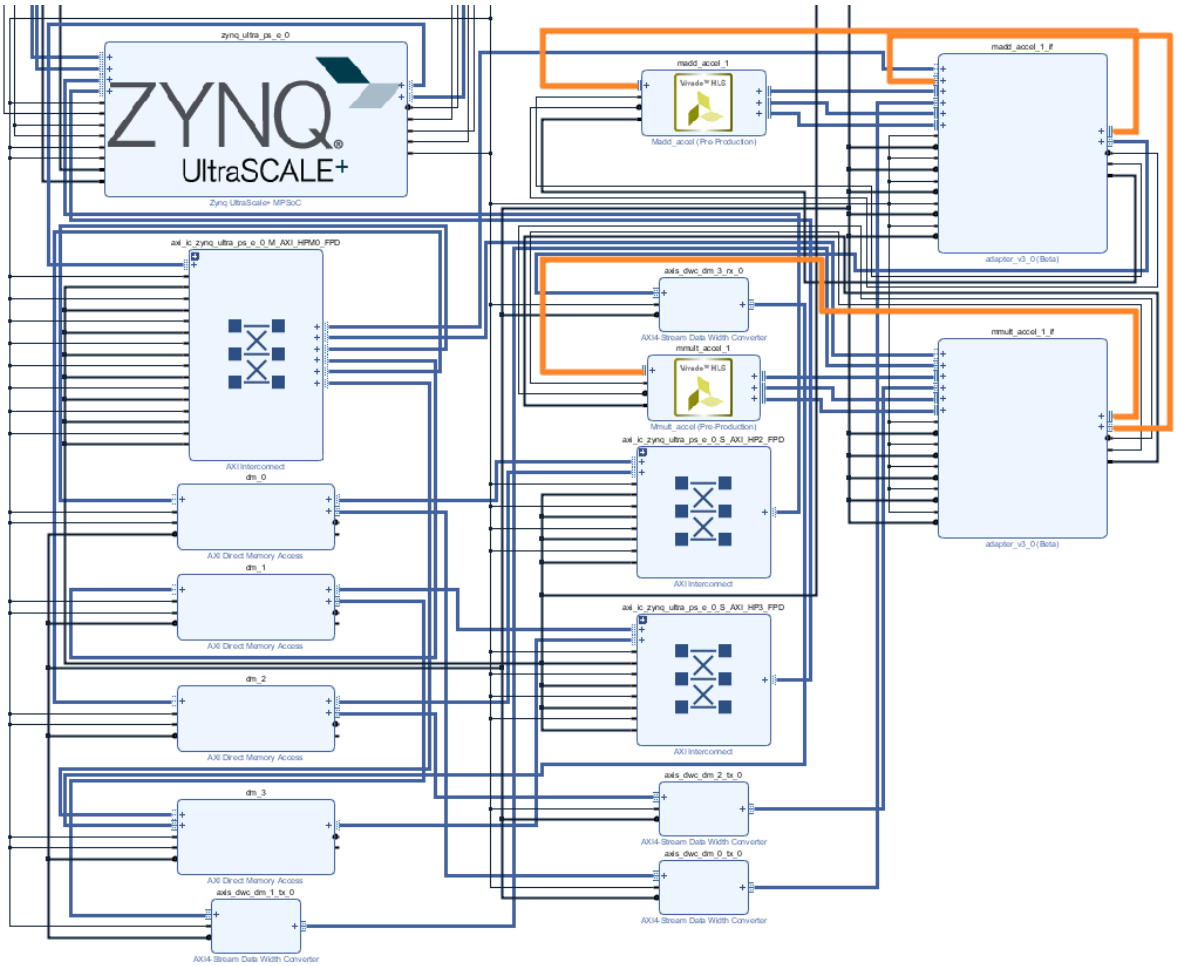


Figure 11: The additional HW generated by the SDSoC 2018.2 compiler.

The measured HW acceleration is **33x** in comparison to the optimized SW computation on the 1.2 GHz Arm A53 processor. See *Figure 12*.

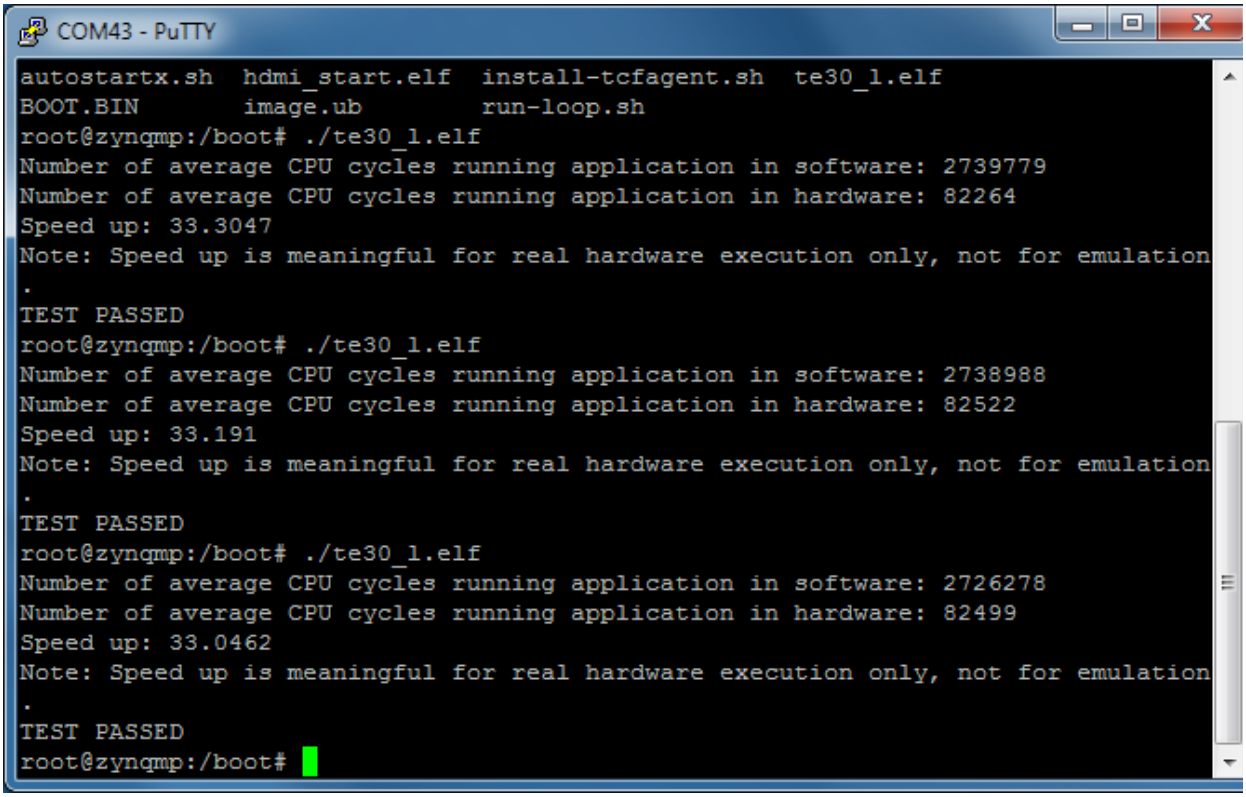


Figure 12: HW Accelerated matrix multiplication and add

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 of 45x45 pixels for each pixel.

Figure 13 presents HW implementation generated by the SDSoC 2018.2 compiler from C++ algorithm definition SW with SG DMA engines for video In/Out data transfer. Two SG DMA engines serve for parallel read of two subsequent video frames from the DDR4 video frame buffers.

The third SG DMA serves for writing of resulting frames with movement vectors to the DDR4 video frame buffer for the display of results. All three SG DMA engines use interrupt based drivers and therefore the used Arm 53 (one of 4 cores) is not 100% busy by the pooling of the result flags during the SG DMA transfers. Interrupt lines are highlighted. See *Figure 13*.

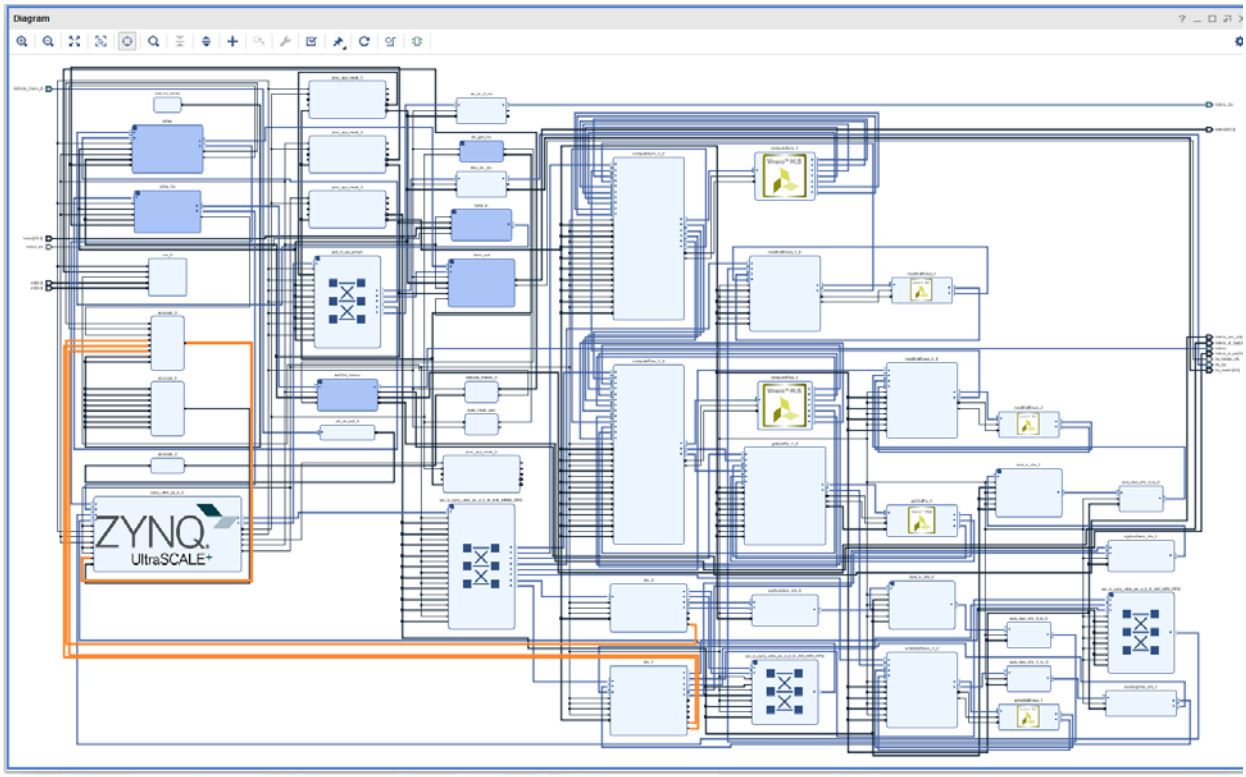


Figure 13: LK Dense Optical Flow in HW with Full HD HDMI video I/O

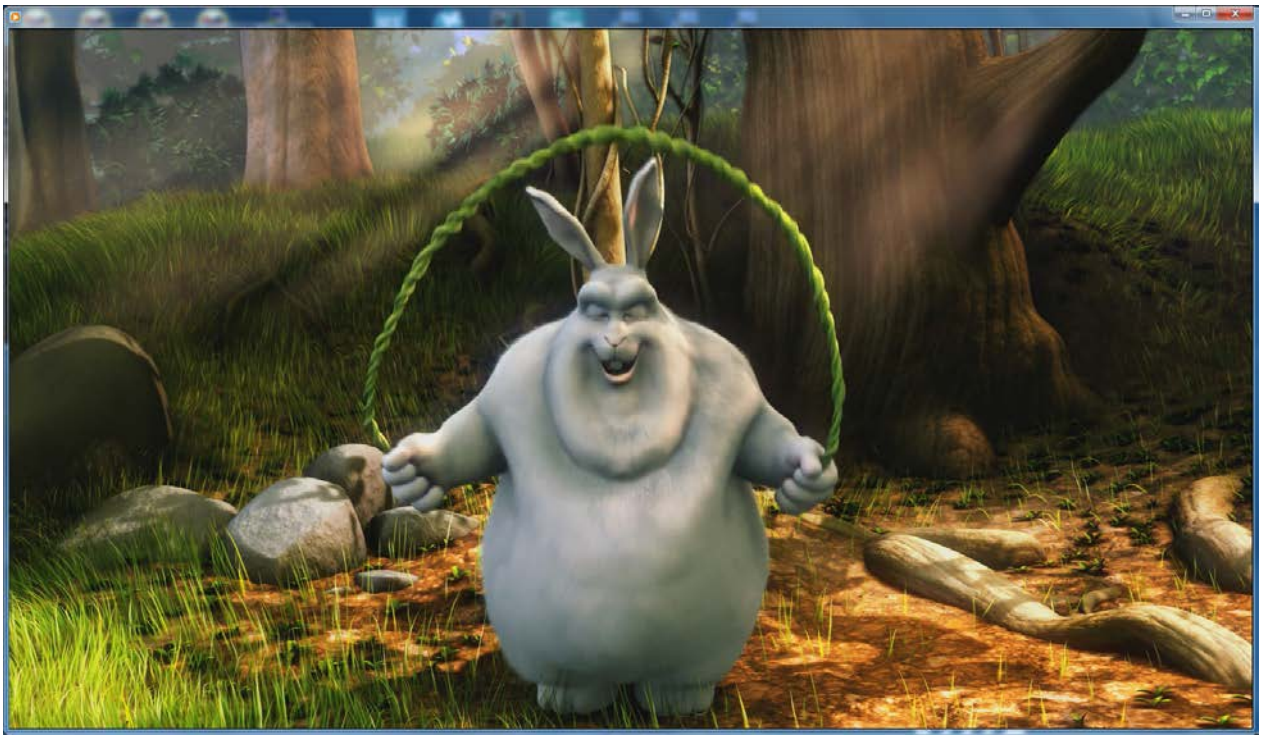


Figure 14: LK Dense Optical Flow input movie Full HD HDMI video 60fps

Figure 15: HW accelerated LK DOF input/output Full HD HDMI 60fps

Figure 14 and *Figure 15* 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 DOF and the load of Arm A53 processors.

LK DOF algorithm with per pixel integral tile size [45x45]	Frames per second	A53 SDSoC CPU load	A53 DeskTop CPU load	Acceleration of LK DOF
In SW	0.121	100%	3%	**1x**
In HW DMA	60	100%	80%	**495x**
In HW SG DMA	60	30%	80%	**495x**

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 **495x** over the same algorithm implemented on 1.2 GHz Arm A53 processor.

8 Inter-cloud connectivity based on the Arrowhead framework

The FitOptiVis (WP4) run-time resources are supported for the Zynq Ultrascale+ module TE0820-03-4EV-1E 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 Arowhead famework 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 TE0820-03-4EV-1E 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.

Installation is described in chapter 8 of App note [4].

10 Install Arrowhead-f support on Zynq Ultrascale+ module

At this stage, the Debian OS configured for the Zynq Ultrascale+ module TE0820-03-4EV-1E can be upgraded to become compatible with the Arrowhead framework G4.0 client and provider C++ demo applications. The installation is described in chapter 9 of App note [4].

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

To control the Zynq Ultrascale+ module, use SSH (preferred) or serial terminal. The installation is described in chapter 10 of App note [4].

Start the compiled *PrividerExample* template.

```
./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 as the *ProviderExample*. The installation is described in chapter 11 of App note [4].

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 will 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 Databaze.

13 Modification of 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. The configuration is described in chapter 12 of App note [4].

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

14 Test the Zynq Ultrascale+ Consumer and Producer

The *ProducerExample* server is running on the “Producer” Zynq Ultrascale+ board, now.

Execute the *ConsumerExample* client example on the “Consumer” Zynq Ultrascale+ board:

```
./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"}
```

15 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 =
          "{"
              "\"e\":[{"
                   "\"n\": \"this_is_the_sensor_id\","
                   "\"v\":" + sValue +","
                   "\"t\": \"" + sLinuxEpoch + "\""
                   "}],"
              "\"bn\": \"this_is_the_sensor_id\","
              "\"bu\": \"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 =
              "{"
                  "\"e\":[{"
                      "\"n\": \"this_is_the_sensor_id\","
                      "\"v\":" + sValue +","
                      "\"t\": \"" + 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 runs on the same module as separate Debian application. Alternatively it can run on another Zynq Ultrascale+ module or another ZynqBerry board connected to the local cloud as described in the App. note [6].

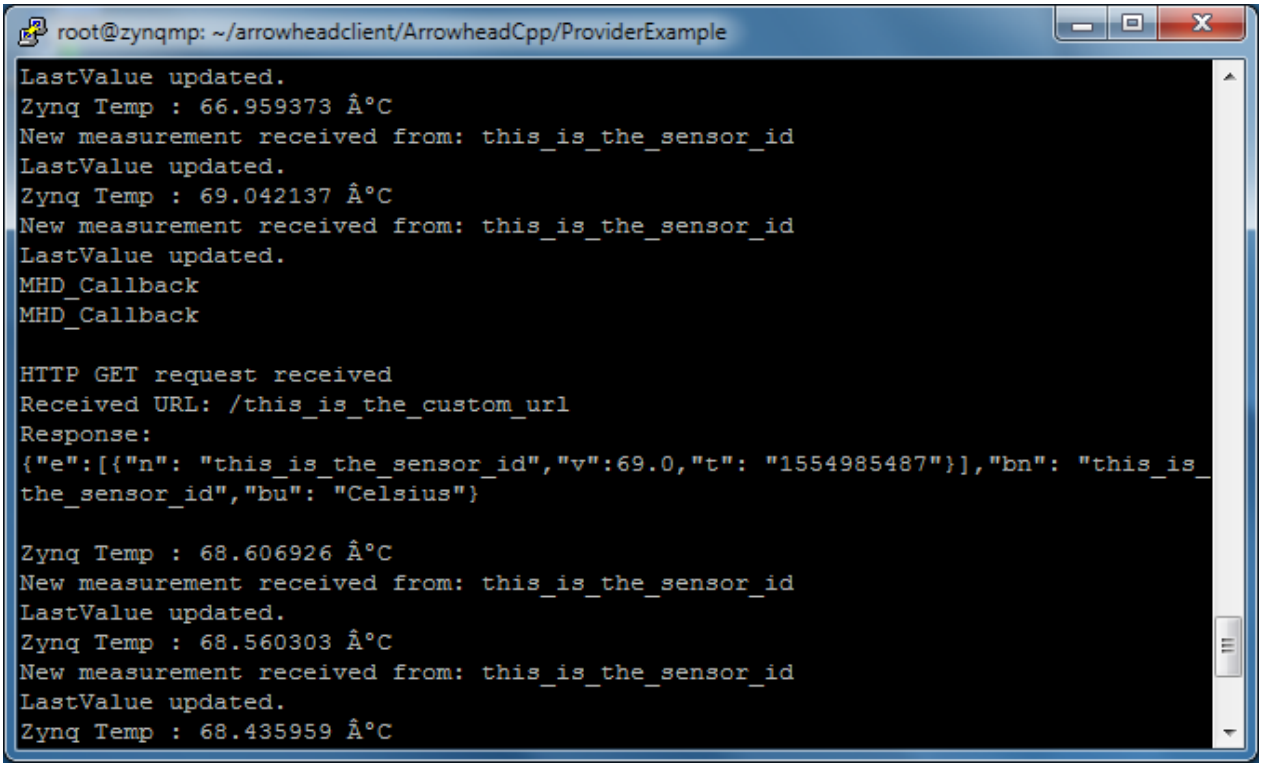


Figure 16: Provider of the chip temperature, response to request, (LK DOW running)

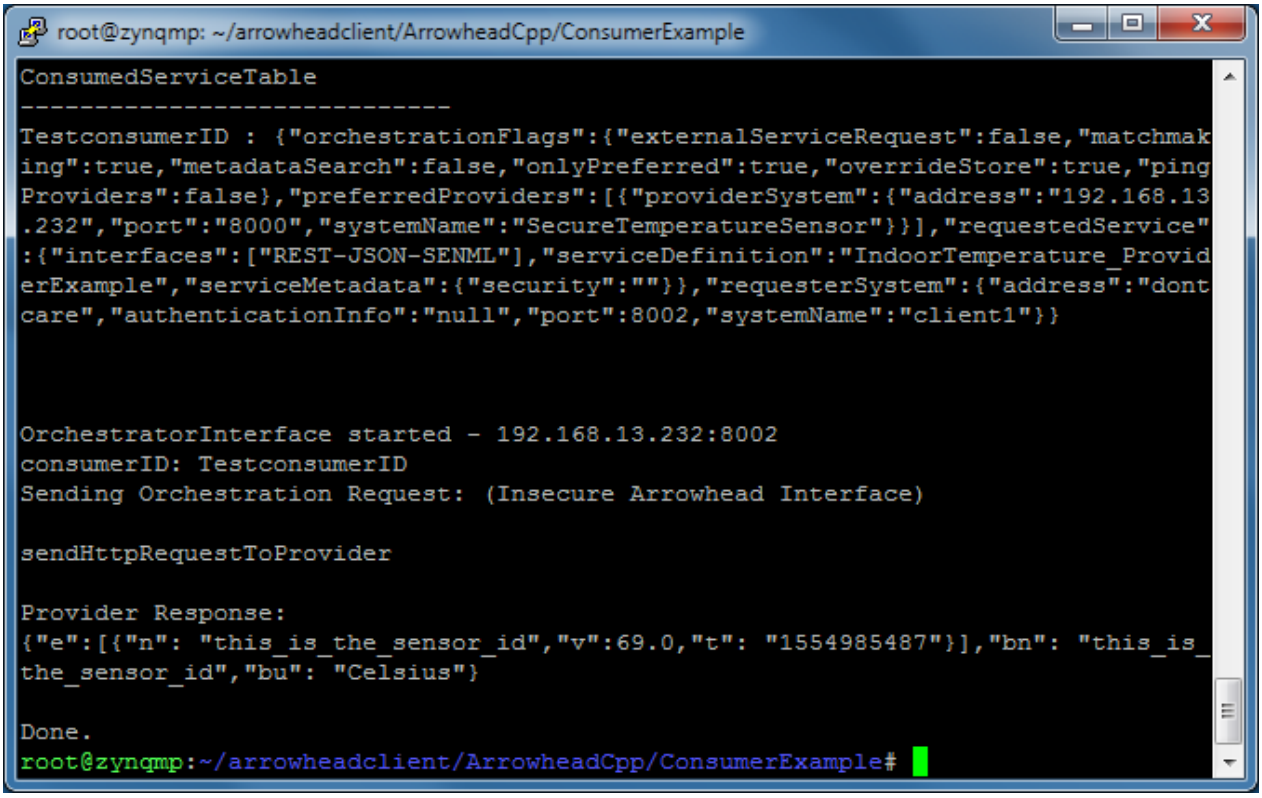


Figure 17: Consumer got the chip temperature (LK DOW is running)

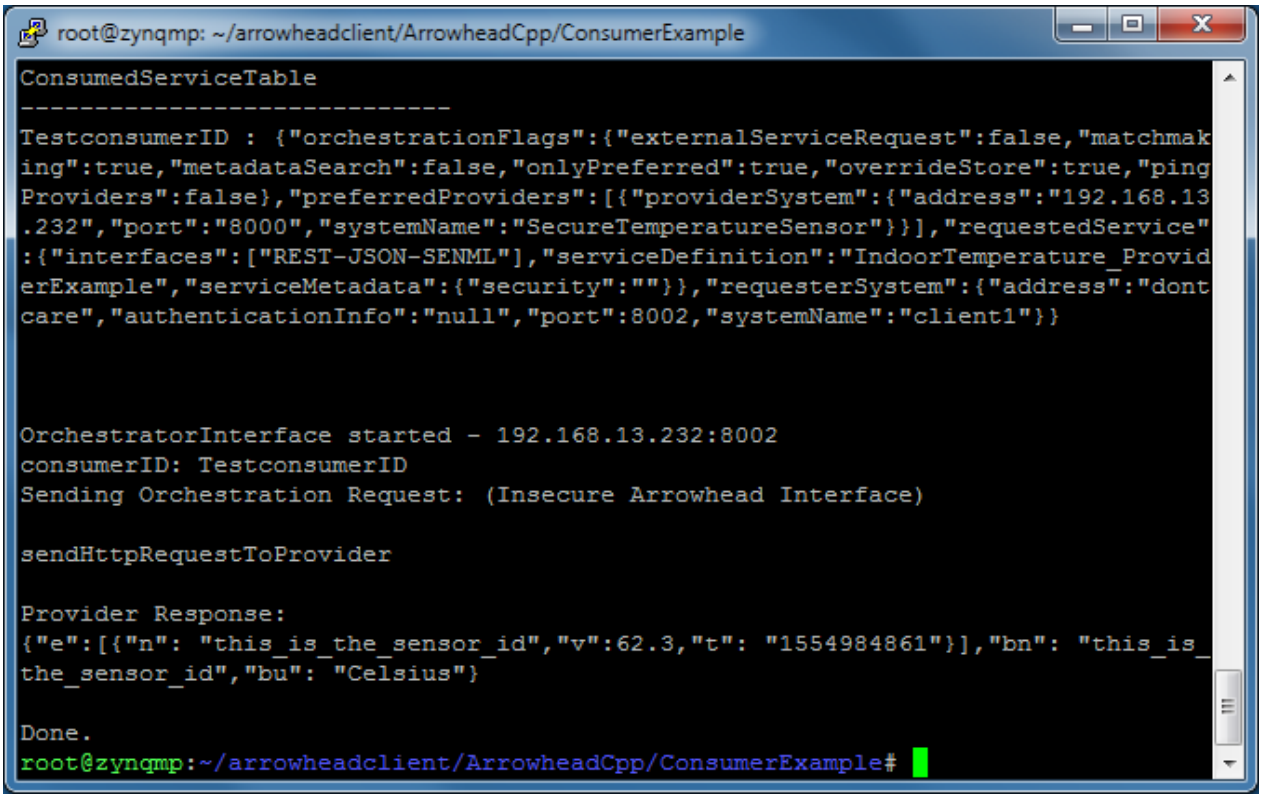


Figure 18: Consumer got the chip temperature (LK DOW is NOT running)

16 Package content

```
├── debian
│   ├── mkdebian.sh
│   ├── image.ub
│   ├── u-boot.elf
│   └── bl31.elf
└── zynq
    ├── TE0820_SDSoC_IMAGEON_FMC_HDMI_701HDMI.zip
    └── install-arrohead-cli-dep.sh
```

References

[1] Trenz Electronic, "MPSoC Module with Xilinx Zynq UltraScale+ ZU4EV-1E, 2 GByte DDR4 SDRAM, 4x5cm", [Online]. https://shop.trenz-electronic.de/en/TE0820-03-04EV-1EA-MPSoC-Module-with-Xilinx-Zynq-UltraScale-ZU4EV-1E-2-GByte-DDR4-SDRAM-4-x-5-cm

[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

[4] Jiři 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] https://shop.trenz-electronic.de/en/TE0701-06-Carrier-Board-for-Trenz-Electronic-7-Series?c=261

Disclaimer