signal processing department of

http://sp.utia.cz

Application Note

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

Arrowhead Compatible Zynq Ultrascale+ Systems with Xilinx 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	24.04.2019	J.Kadlec	Description of nine Zynq Ultrascale+ systems. Three TE0820-03 modules on three carriers.
2	04.05.2019	J Kadlec	Updated figures; file ProviderExample.cpp; Updated Appendix 19; Updated Appendix 20.

Table of Contents

Table of Figures

Table of Tables

Acknowledgement

This work has been partially supported by ECSEL JU project Productive4.0 No. 737459.

1 Introduction to nine supported Zynq Ultrascale+ Systems

This application note describes nine Arrowhead framework compatible Zynq Ultrascale+ systems with support for the Xilinx SDSoC 2018.2 system level compiler.

- Zynq Ultrascale+ systems can accelerate computing by HW acceleration of algorithms in the programmable logic. See example of acceleration 50 x in chapter 6.
- Shorter computing latency can be achieved in comparison to pure SW solution.
- The total system energy consumption associated with single iteration of an HW accelerated algorithm can be significantly reduced in comparison to the pure SW implementation.
- The nine supported systems use identical Debian configuration and Arrowhead framework support
- The described board support package generation project is starting point for creation, configuration and compilation of user specific board support packages with custom I/O data interfaces and user specific extensions.

Three Zynq Ultrascale+ modules are supported. See Table 1.

Table 1: Supported Zynq Ultrascale+ modules

TE0820-03-2CG-1EA	MPSoC Module with Xilinx Zynq UltraScale+ ZU2CG-1E, 2 GByte DDR4, 4x5cm. 2x Arm Cortex A53 64bit, 2x Arm Cortex R5. https://shop.trenz-electronic.de/en/TE0820-03-02CG-1EA-MPSoC-Module-with-Xilinx-Zynq-UltraScale-ZU2CG-1E-2-GByte-DDR4-SDRAM-4-x-5-cm
TE0820-03-2EG-1EA	MPSoC Module with Xilinx Zynq UltraScale+ ZU2EG-1E, 2 GByte DDR4, 4x5cm. 4x Arm Cortex A53 64bit, 2x Arm Cortex R5, Mali GPU. https://shop.trenz-electronic.de/en/TE0820-03-02EG-1EA-MPSoC-Module-with-Xilinx-Zynq-UltraScale-ZU2EG-1E-2-GByte-DDR4-SDRAM-4-x-5-cm
TE0820-03-4EV-1EA	MPSoC Module with Xilinx Zynq UltraScale+ ZU4EV-1E, 2 GByte DDR4, 4x5cm. 4x Arm Cortex A53 64bit, 2x Arm Cortex R5, Mali GPU, Video Codec, larger programmable logic area. 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

Three carrier boards are supported. See Table 2.

Table 2: Supported carrier boards

TE0701-06	Carrier Board targets extensions with FMC card and PMODs. https://shop.trenz-electronic.de/en/TE0701-06-Carrier-Board-for-Trenz-Electronic-7-Series?c=261
TE0703-06	Carrier Board targets wide I/O with pre-processing in a Lattice FPGA. https://shop.trenz-electronic.de/en/TE0703-06-TE0703-Carrier-board-for-Trenz-Electronic-modules-with-4-x-5-cm-form-factor?c=261
TE0706-03 **TE0790-02**	Carrier Board targets extensions with second Ethernet in the Zynq PL. https://shop.trenz-electronic.de/en/TE0706-03-TE0706-Carrierboard-for-Trenz-Electronic-Modules-with-4-x-5-cm-Form-Factor?c=261 XMOD FTDI JTAG Adapter- Xilinx compatible. Console and Jtag for TE0706-03. https://shop.trenz-electronic.de/en/TE0790-02-XMOD-FTDI-JTAG-Adapter-Xilinx-compatible

The evaluation package described in this application note supports in total nine Zynq Ultrascale+ systems with common procedure for generation of the board support package for Petalinux 2018.2 kernel, Debian OS and for the Arrowhead framework clients.

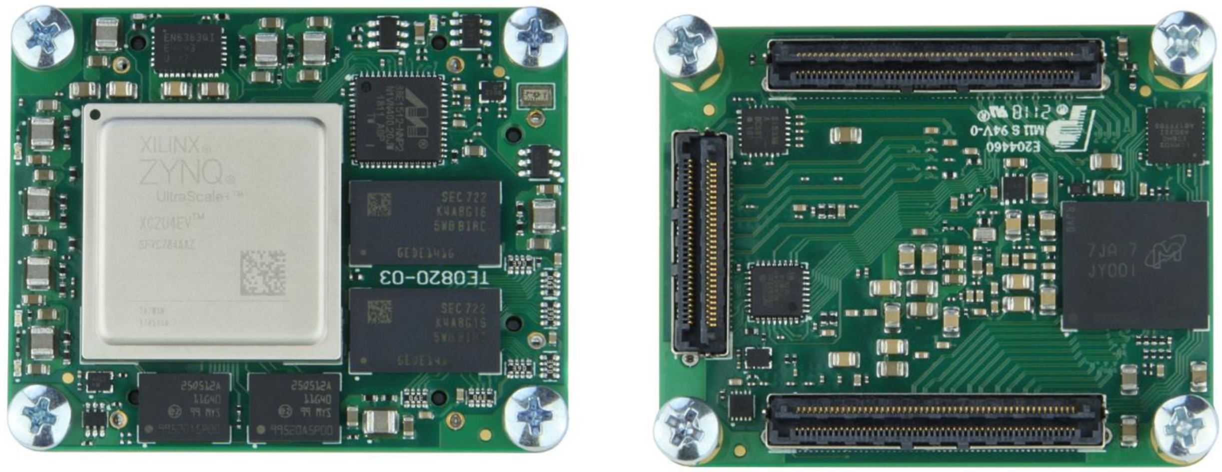


Figure 1: Zynq Ultrascale+ module TE0820. Size 4x5cm

Table 3: Parameters of supported modules

Trenz Electronic Module	**TE0820-03-2CG-1EA**	**TE0820-03-2EG-1EA**	**TE0820-03-4EV-1EA**
Xilinx Device	**XCZU2CG-1SFVC784E**	**XCZU2EG-1SFVC784E**	**XCZU4EV-1SFVC784E**
Application Processing Unit	Dual-core ARM® Cortex™-A53	Quad-core ARM® Cortex™-A53	Quad-core ARM® Cortex™-A53
Real Time Processing Unit	Dual-core ARM Cortex R5	Dual-core ARM Cortex R5	Dual-core ARM Cortex R5
Graphics Processing Unit	-	ARM Mali™-400 MP2	ARM Mali™-400 MP2
Video Codec Unit	-	-	Supports H.264/H.265
Dynamic Memory Interf.	DDR4, LPDDR4, DDR3, DDR3L, LPDDR3		
High-Speed Peripherals	PCIe® Gen2, USB3.0, SATA 3.1, DisplayPort, Gigabit Ethernet		
Programmable logic Features			
System Logic Cells (K)	103	103	192
Block RAM Memory (Mb)	5.3	5.3	4.5
UltraRAM Memory (Mb)	-	-	13.5
DSP Slices	240	240	728
Video Codec Unit (VCU)	-	-	1
GTH 16.3 Gb/s Transceivers	-	-	16
DDR4 on module (GB)	2	2	2
ARM® Cortex™-A53 clock	1.2 GHz	1.2 GHz	1.2 GHz

Unit cost of each of the nine supported Zynq Ultrascale+ systems is presented in *Figure 2*

Modules and carrier boards are designed and manufactured by company Trenz Electronic and the indicated price is based on the assumption of 1000 units without VAT.
It is cost without power supply and heatsink. Data from: https://www.trenz-electronic.de/

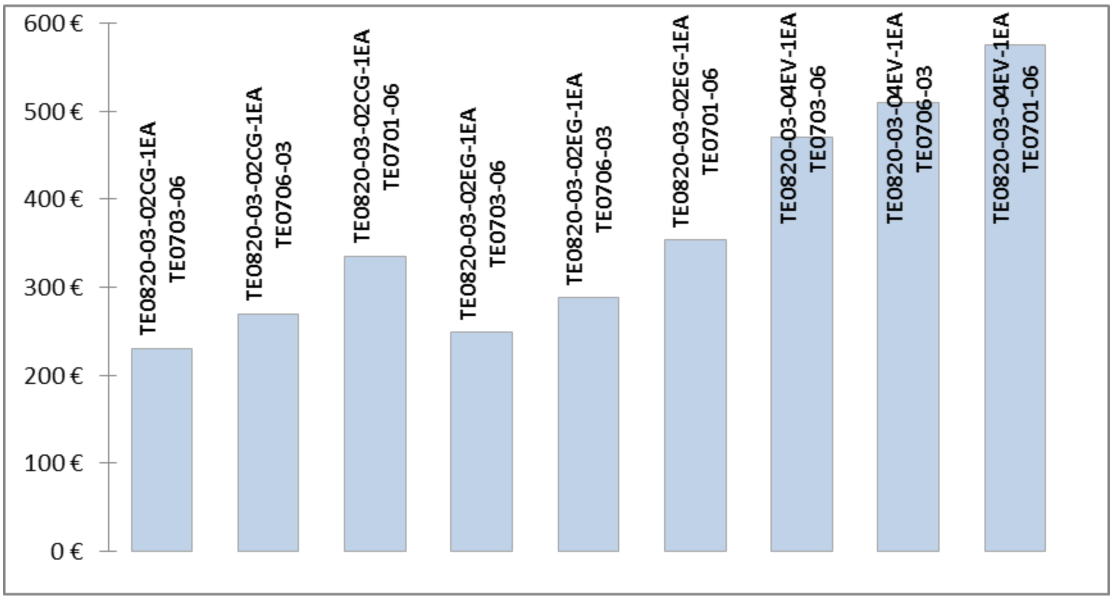


Figure 2: Cost of each supported Zynq Ultrascale+ system (1000 units, without VAT)

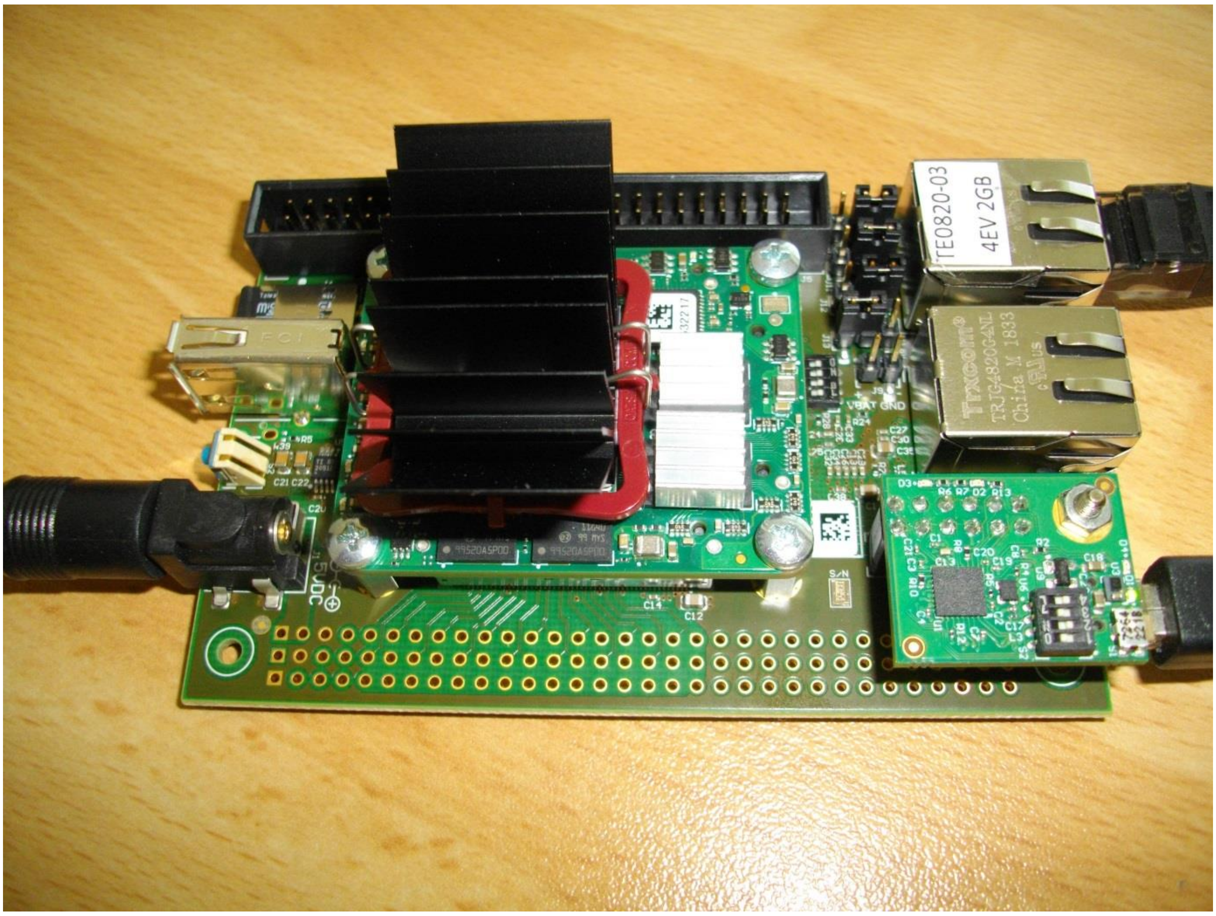


Figure 3: TE0820-03-4EV-1EA module on carrier board TE0706-03

Figure 4: Zynq Ultrascale+ module on TE0701-06 carrier board

Figure 5: The Zynq Ultrascale+ system and RaspberreyPi 3B with Arrowhead framework

2 Create SDSoC platform for supported Zynq Ultrascale+ system

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 6*. 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 8*. The resulting compiled system remains compatible with 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.

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+ modules.

We will use the Zynq Ultrascale+ board support package generation project included in the evaluation package accompanying this application note. 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 Zynq Ultrascale+ module on the carrier board.

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 module (15 for TE0820-03-4EV-1EA, 16 for TE0820-03-2CG-1EA or 17 for TE0820-03-2EG-1EA). 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 6 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
```

Configurations and compilation steps for Petalinux 2018.2 and and Debian 9.8 are described in next two sections.

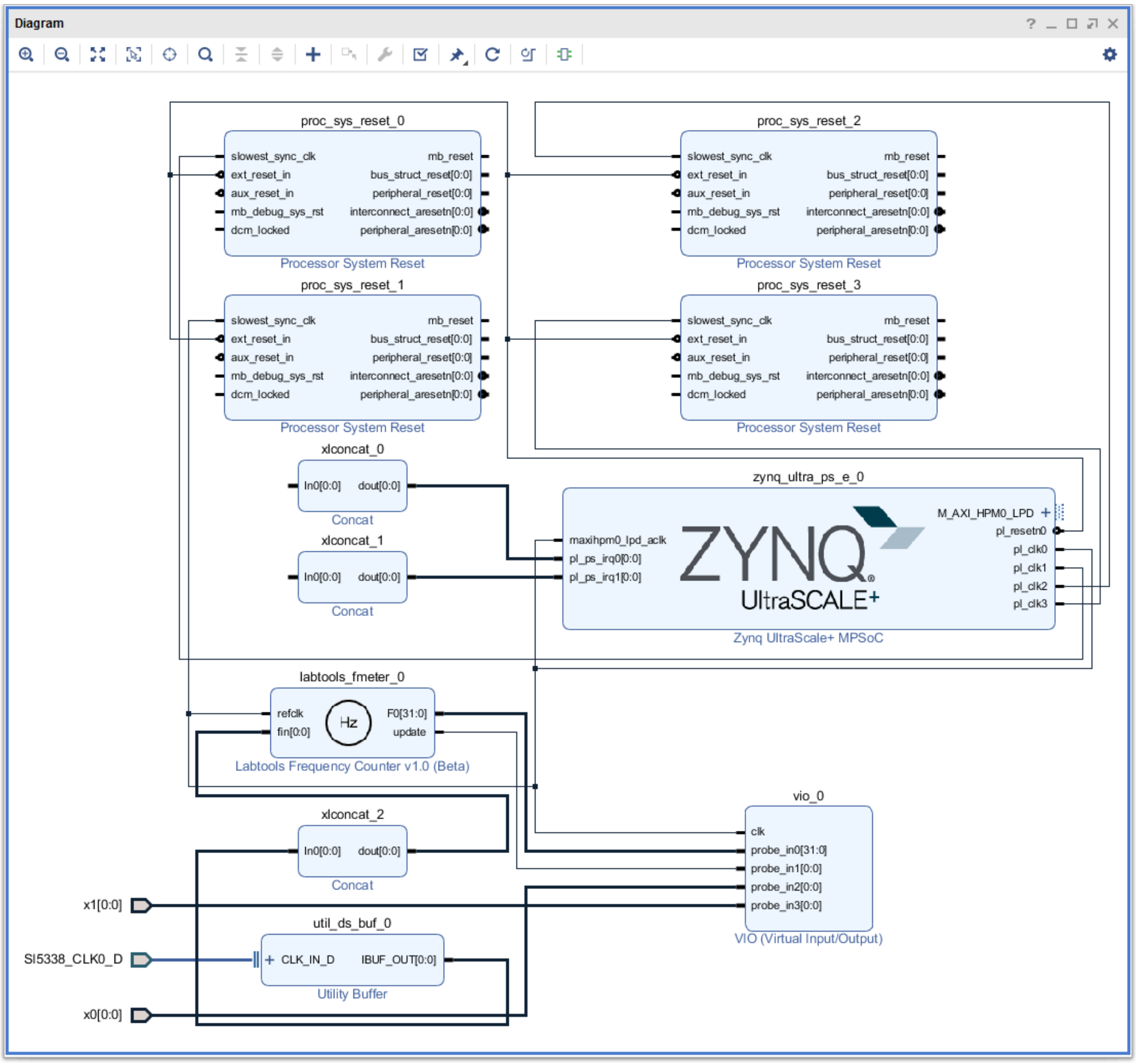


Figure 6: The initial Vivado design. It defines the SDSoC 2018.2 platform.

3 Configuration of the PetaLinux 2018.2

The configuration and compilation of the *Petalinux 2018.2* kernel and *Debian 9.8 Stretch* image for the Zynq Ultrascale+ module is described now. The configuration is performed on the Ubuntu 16.04 LTS.

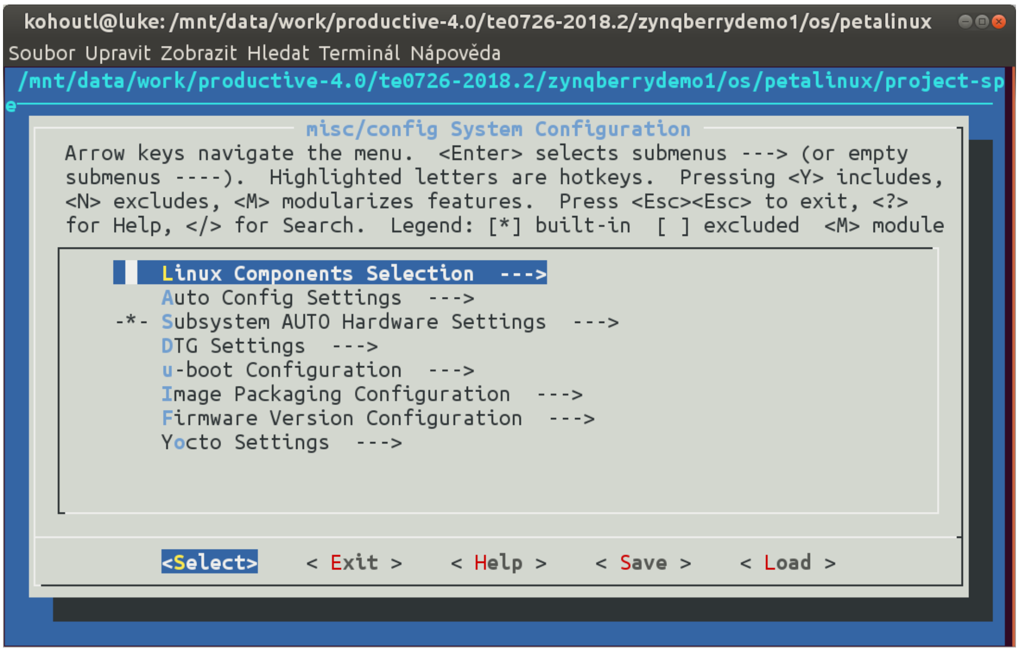
We used the *VMware Workstation 14 Player* on Win7 or Win10 PC with Intel i7 CPU (8 processors, 16 GB RAM). We use configuration of the VM machine with allocated 6 processors and 8 GB of RAM for the Ubuntu 16.04 LTS. It results in fast compilation of the PetaLinux 2018.2 kernel.

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:

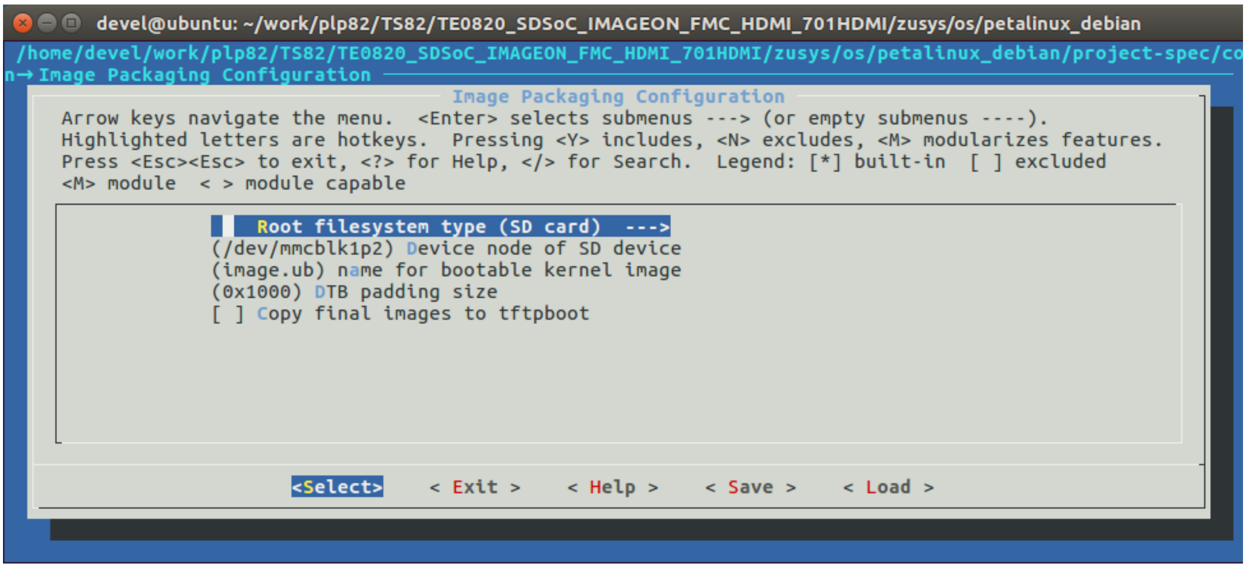
```
/opt/petalinux/petalinux-v2018.2-final
```

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 4: Needed tools with a corresponding package name.

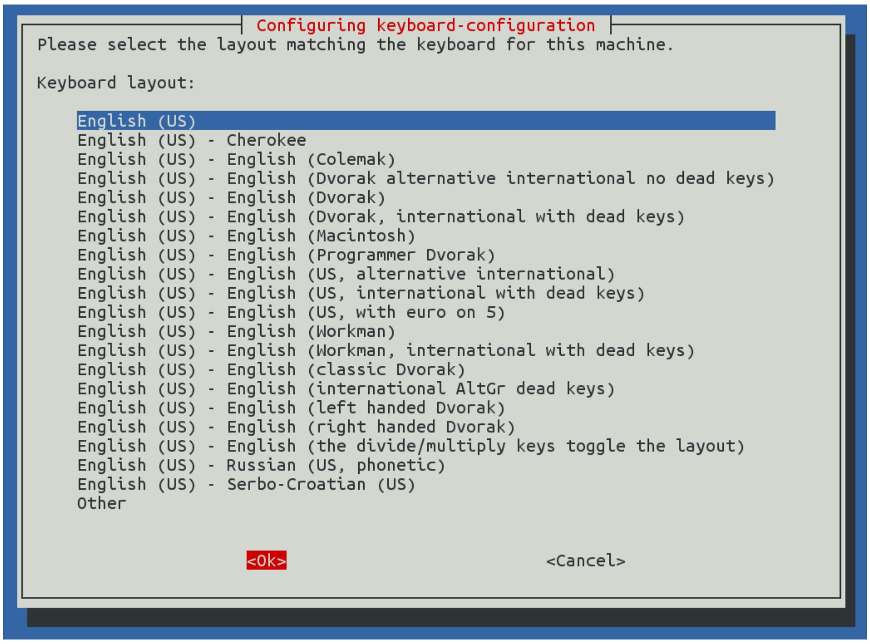
Tool	***Package***
dd	coreutils
losetup	mount
parted	parted
lsblk	util-linux
mkfs.vfat	dosfstools
mkfs.ext4	e2fsprogs
debootstrap	debootstrap
gzip	gzip
cpio	cpio
chroot	coreutils
apt-get	apt
dpkg-reconfigure	debconf
sed	sed
locale-gen	locales
update-locale	locales
qemu-arm-static	qemu-user-static

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. It can be repeated for the other two modules TE0820-03-2CG-1EA and TE0820-03-2EG-1EA. The PetaLinux 2018.2 kernel and the Debian 9.8 image is identical for all three modules. It can be re-used by the platforms for the other two modules without repeating the configuration and compilation steps

All three platforms created in chapters 1 – 5 are 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 application is described in this section. We will describe it for TE0820-03-4EV-1EA module.

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_1
   ```
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[100,100] = A[100,100] * B[100,100] + C[100,100]
   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_1\Release\sd_card\BOOT.BIN
   X:\4EV-1EA\te30_1\Release\sd_card\te30_1.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. Use the USB keyboard and login as:

```
root
```

Password:

```
root
```

21. The HW accelerated matrix multiplication demo can be executed on Zynq Ultrascale+ board from the automatically mounted SD by executing:

```
/boot/te06_l.elf
```

22. See *Figure 9*. The HW acceleration measured by the number of Arm A53 clock cycles.
23. 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 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[100,100] = A[100,100] * B[100,100] + C[100,100].

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=10000
        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=10000 max=10000
        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=100
        mmult2: for (int j = 0; j < dim ; j++) {
        #pragma HLS PIPELINE
        #pragma HLS LOOP_TRIPCOUNT min=1 max=100
           int result = 0;
           mmult3: for (int k = 0; k < DATA_SIZE; k++) {
           #pragma HLS LOOP_TRIPCOUNT min=1 max=100
               result += A[i][k] * B[k][j];
           }
           out[i * dim + j] = result;
        }
    }
}
```

Figure 7: The SW source code for Matrix mult and add

The generated HW design is interfaced to the modified user C++ source code. SW is compiled into *te06_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 two Vivado HLS HW accelerators. One for matrix 100x100 int32 multiplication and one for matrix 100x100 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 *te06_l.elf* file. The generated HW respects the initial board support package constrains and fits to the Zynq Ultrascale+ module.

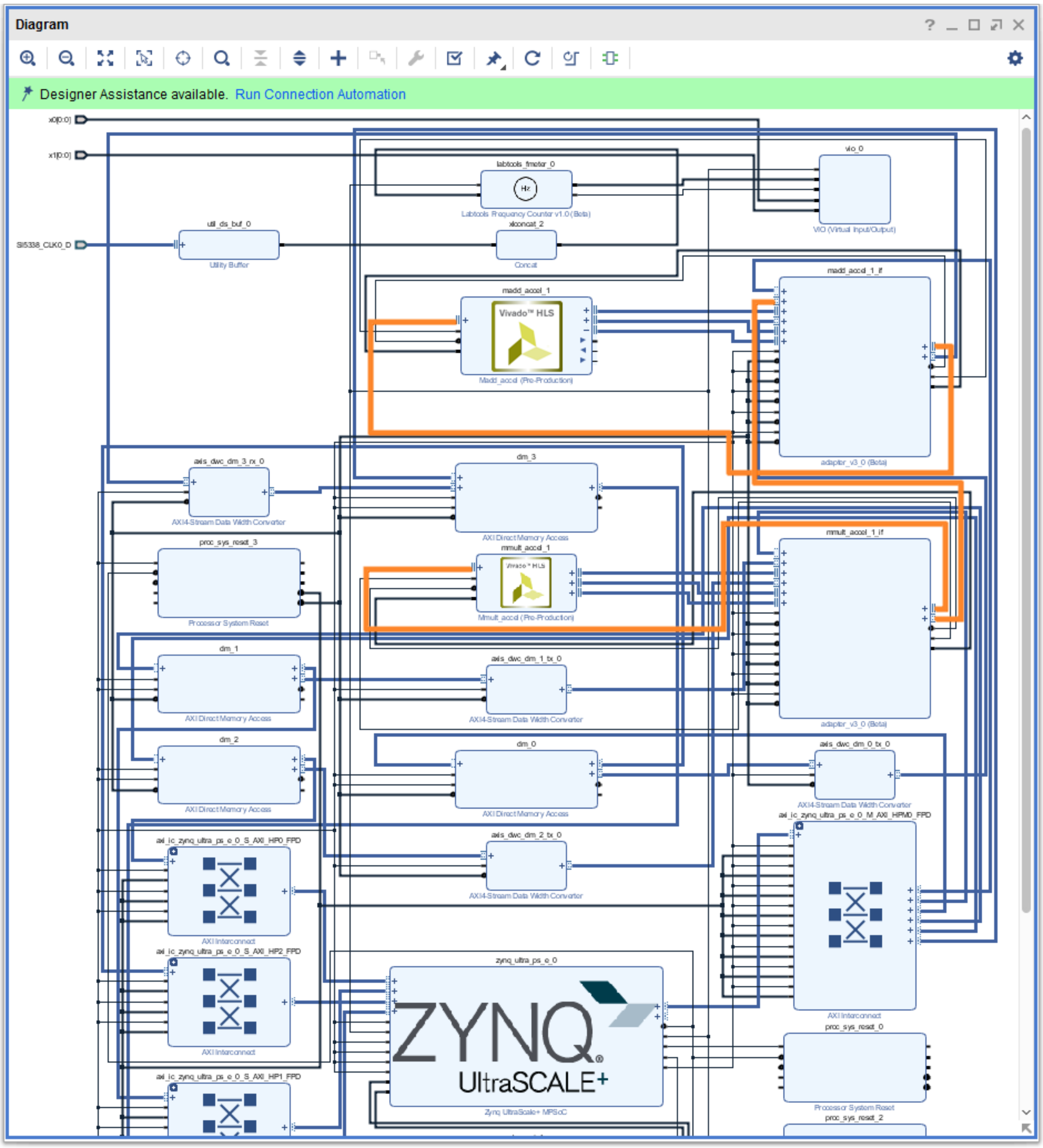


Figure 8: The additional HW generated by the SDSoC 2018.2 compiler.

The measured HW acceleration is **51.7x** in comparison to the optimized SW computation on the 1.2 GHz Arm A53 processor. See *Figure 9*.

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

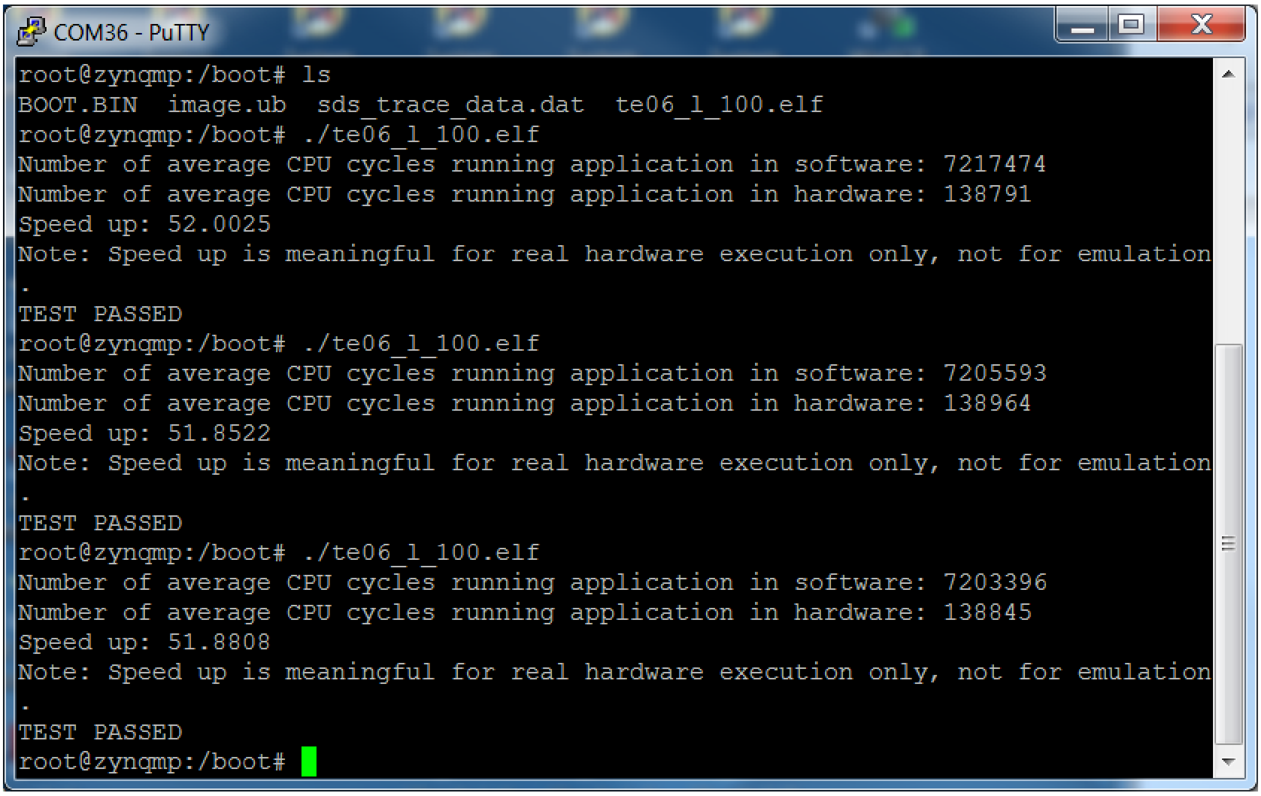


Figure 9: HW Accelerated matrix multiplication and add

7 Board to board connectivity based on the Arrowhead framework

The Arrowhead framework compatible clients on the 64 bit Arm Cortex A53 processor are supported for the Zynq Ultrascale+ module. SW implementation of the Arrowhead framework [3] has been developed within WP1 of the ECSEL Productive4.0 project https://productive40.eu/.

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 hosts C++ provider capable to measure the actual temperature of silicon in the SFVC784E package. 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.

8 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].

9 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].

10 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.

11 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.

12 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, perform the configuration as described in chapter 12 of the application note [4].

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+ 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"}
```

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"


const std::string version = "4.1";

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;

//SenML format
//todo:
//generate own measured value into "measuredValue"
//"value" should be periodically updated
//"sLinuxEpoch" should be periodically updated

     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;
    }


//      double value = 26.0;

    // (raw + offset) * scale ... in milidegree Celsius
    float value = ((float)(t_raw + t_off) * t_scale) / 1000.00f;

//convert double to string
     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\""
          "}";

//do not modify below this

    oSensorHandler.processProvider(measuredValue, bSecureProviderInterface,
bSecureArrowheadInterface);

    while (true) {

        linuxEpochTime = std::time(0);
        sLinuxEpoch = std::to_string((uint64_t)linuxEpochTime);

//         if (value < 30.0) value += 0.1;
//         else              value = 26.0;

        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 Â°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 application note [4].

```
root@zynqmp: ~/arrowheadclient/ArrowheadCpp/ProviderExample
LastValue updated.
Zynq Temp : 66.959373 Â°C
New measurement received from: this_is_the_sensor_id
LastValue updated.
Zynq Temp : 69.042137 Â°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":69.0,"t": "1554985487"}],"bn": "this_is_
the_sensor_id","bu": "Celsius"}

Zynq Temp : 68.606926 Â°C
New measurement received from: this_is_the_sensor_id
LastValue updated.
Zynq Temp : 68.560303 Â°C
New measurement received from: this_is_the_sensor_id
LastValue updated.
Zynq Temp : 68.435959 Â°C
```

Figure 10: Provider of the chip temperature, response to request, (LK DOW running)

```
root@zynqmp: ~/arrowheadclient/ArrowheadCpp/ConsumerExample
ConsumedServiceTable
-----------------------------
TestconsumerID : {"orchestrationFlags":{"externalServiceRequest":false,"matchmak
ing":true,"metadataSearch":false,"onlyPreferred":true,"overrideStore":true,"ping
Providers":false},"preferredProviders":[{"providerSystem":{"address":"192.168.13
.232","port":"8000","systemName":"SecureTemperatureSensor"}}],"requestedService"
:{"interfaces":["REST-JSON-SENML"],"serviceDefinition":"IndoorTemperature_Provid
erExample","serviceMetadata":{"security":""}},"requesterSystem":{"address":"dont
care","authenticationInfo":"null","port":8002,"systemName":"client1"}}



OrchestratorInterface started - 192.168.13.232:8002
consumerID: TestconsumerID
Sending Orchestration Request: (Insecure Arrowhead Interface)

sendHttpRequestToProvider

Provider Response:
{"e":[{"n": "this_is_the_sensor_id","v":69.0,"t": "1554985487"}],"bn": "this_is_
the_sensor_id","bu": "Celsius"}

Done.
root@zynqmp:~/arrowheadclient/ArrowheadCpp/ConsumerExample#
```

Figure 11: Consumer got the chip temperature (LK DOW is running)

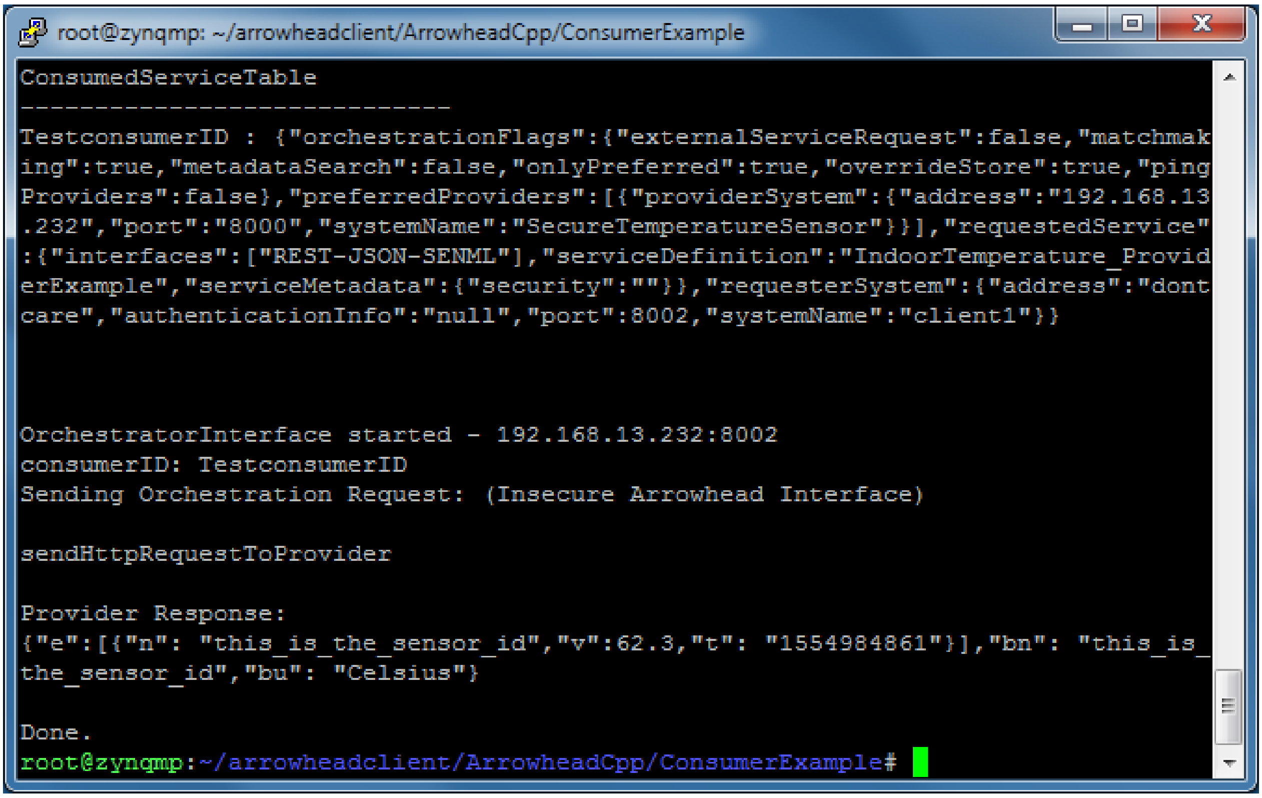
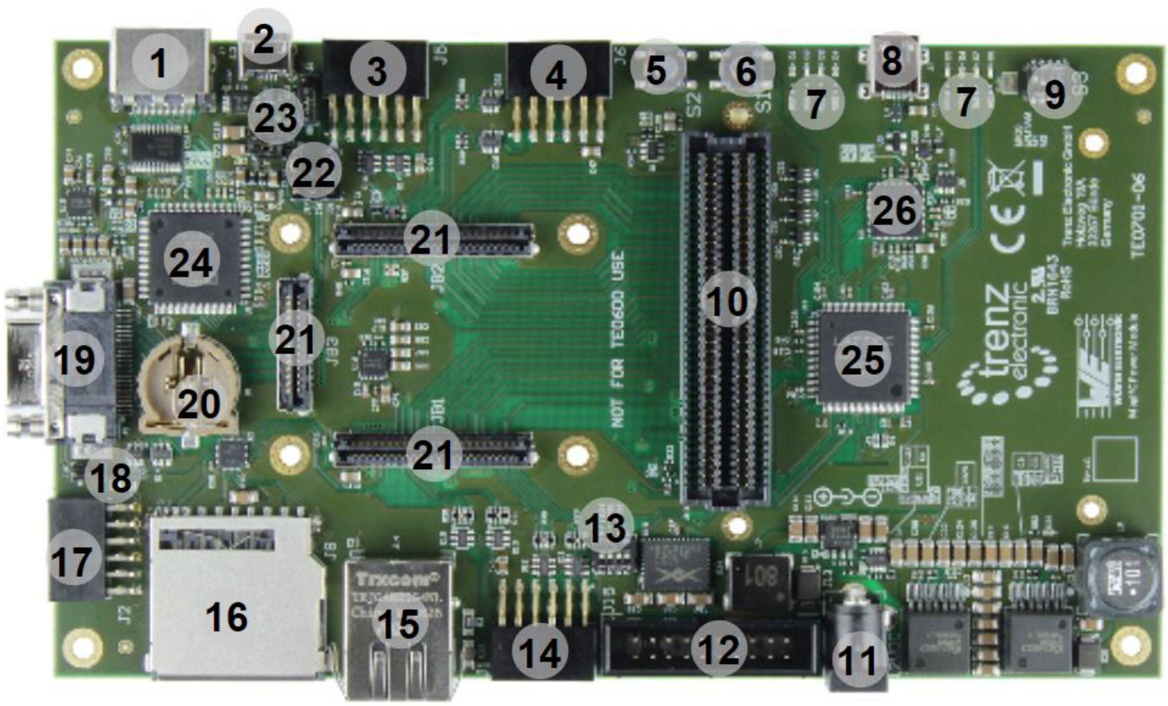


Figure 12: Consumer got the chip temperature (LK DOW is NOT running)

15 Appendix: TE0701-06 carrier board details



1.HDMI connector (1.4 HEAC support), J4
2.Micro-USB2 connector, J12
3.Pmod connector, J5
4.Pmod connector, J6
5.User push-button ("RESTART" button by default), S2
6.User push-button ("RESET" button by default), S1
7.8x red user LEDs, D1 ... D8
8.Mini-USB2 connector, J7
9.User 4-bit DIP switch, S3
10.VITA 57.1 compliant LPC FMC connector, J10
11.Barrel jack for 12V power supply, J13
12.ARM JTAG connector (DS-5 D-Stream), J15
13.User 4-bit DIP switch, S4
14.Pmod connector, J1
15.RJ45 Gigabit Ethernet connector, J14
16.SD Card connector, J8
17.Pmod connector, J2
18.Jumper, J18
19.Mini CameraLink connector, J3
20.CR1220 Backup-Battery holder, B1
21.Trenz Electronic 4 x 5 modules B2B connectors, JB1 ... JB3
22.Jumper J16, J17, J21
23.Jumper J9, J19, J20
24.Analog Devices ADV7511 HDMI Transmitter, U1
25.Lattice Semiconductor MachXO2 1200 HC System Controller CPLD, U14
26.FTDI FT2232H USB2 to JTAG/UART Bridge, U3

Figure 13: TE0701-06 Carrier Board.

Following table gives an overview of the Pmod connectors and the signals routed to the attached module and to the System Controller CPLD U14:

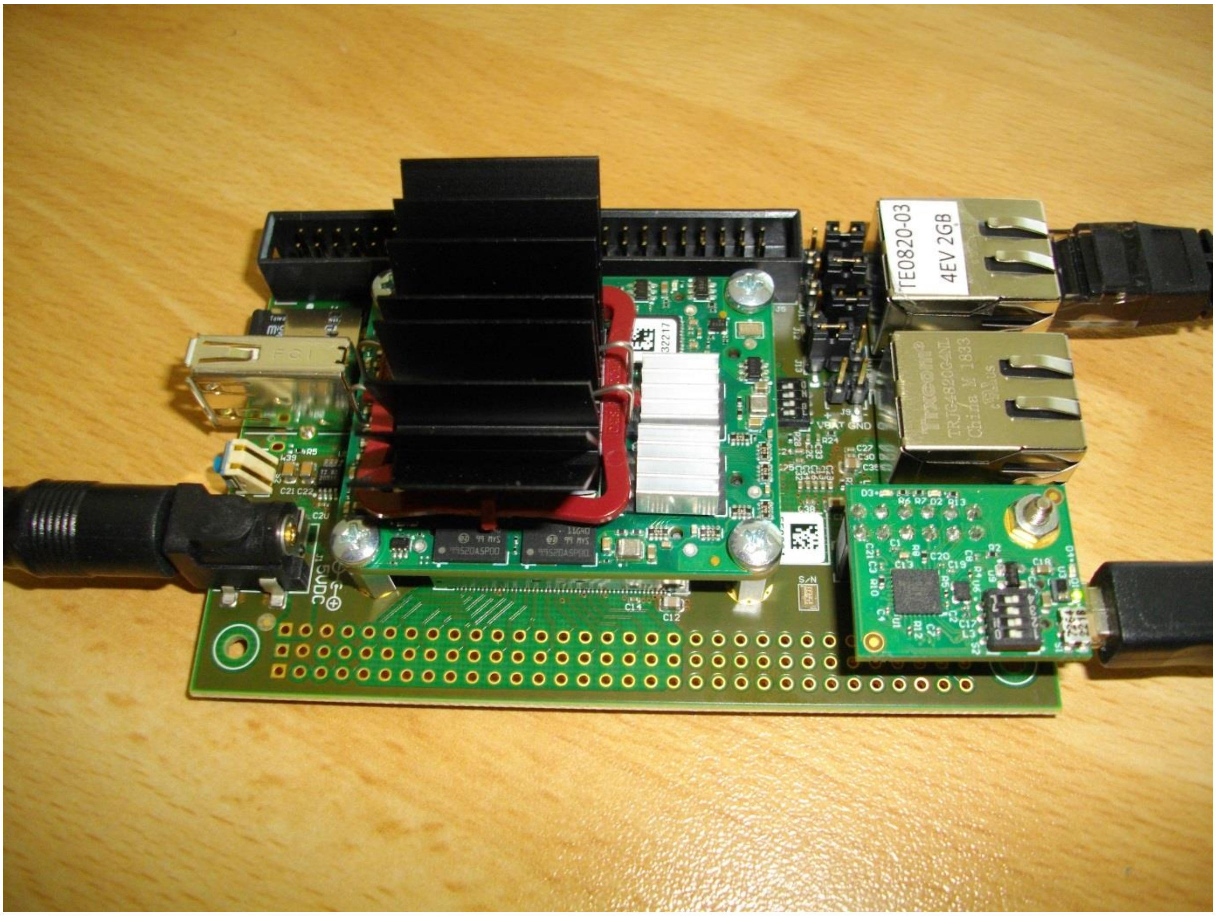
Table 5: Connections of Zynq Ultrascale+ pins to Pmod connectors of TE0701-06

Pmod J1 pin	Signal Schematic Name /(Zynq Ultrascale+ SFVC784 MIO name)	Connected to	Zynq Ultrascale+ SFVC784 package pin
1	MIO0 / (MIO33)	B2B connector JB1, pin 88; DIP switch S3-4	L16
2	MIO9 / (MIO32)	B2B connector JB1, pin 92	J16
3	MIO14 / (MIO30)	B2B connector JB1, pin 91; SC CPLD U14, pin 37	F16
4	MIO15 / (MIO31)	B2B connector JB1, pin 86; SC CPLD U14, pin 18	H16

7	MIO13 / (MIO29)	B2B connector JB1, pin 98; SC CPLD U14, pin 30	G16
8	MIO10 / (MIO26)	B2B connector JB1, pin 96; SC CPLD U14, pin 29	L15
9	MIO11 / (MIO27)	B2B connector JB1, pin 94; SC CPLD U14, pin 19	J15
10	MIO12 / (MIO28)	B2B connector JB1, pin 100; SC CPLD U14, pin 36	K15
Pmod J2 pin	**Signal Schematic Name**	**Connected to**	**Notes**
1	PX3	SDIO Port Expander U2, pin 10	muxed to signal 'SD_DAT3' (B2B JB1, pin 18)
2	PX4	SDIO Port Expander U2, pin 12	muxed to signal 'SD_CMD' (B2B JB1, pin 26)
3	PX0	SDIO Port Expander U2, pin 14	muxed to signal 'SD_DAT0' (B2B JB1, pin 24)
4	PX5	SDIO Port Expander U2, pin 13	muxed to signal 'SD_CLK' (B2B JB1, pin 28)
7	PX1	SDIO Port Expander U2, pin 15	muxed to signal 'SD_DAT1' (B2B JB1, pin 22)
8	PX2	SDIO Port Expander U2, pin 8	muxed to signal 'SD_DAT2' (B2B JB1, pin 20)
9	PX6	SC CPLD U14, pin 49	-
10	PX7	SC CPLD U14, pin 48	-
Pmod J5 pin	**Signal Schematic Name**	**Connected to**	**Zynq Ultrascale+ `SFVC784 pin`**
1	PA1_P	B2B connector JB2, pin 27	N6 usable as LVDS pair
2	PA1_N	B2B connector JB2, pin 25	N7
3	PA2_P	B2B connector JB2, pin 26	L3 usable as LVDS pair
4	PA2_N	B2B connector JB2, pin 28	L2

7	PA0_P	B2B connector JB2, pin 23	K1 usable as LVDS pair
8	PA0_N	B2B connector JB2, pin 21	L1
9	PA3_P	B2B connector JB2, pin 22	M6 usable as LVDS pair
10	PA3_N	B2B connector JB2, pin 24	L5
Pmod J6 pin	**Signal Schematic Name**	**Connected to**	**Zynq Ultrascale+ SFVC784 pin**
1	PB2_N	B2B connector JB2, pin 51	AE5 usable as LVDS pair
2	PB2_P	B2B connector JB2, pin 53	AF5
3	PB0_N	B2B connector JB2, pin 33	AC8 usable as LVDS pair
4	PB0_P	B2B connector JB2, pin 31	AB8
7	PB3_N	B2B connector JB2, pin 47	AF6 usable as LVDS pair
8	PB3_P	B2B connector JB2, pin 45	AF7
9	PB1_N	B2B connector JB2, pin 43	AE8 usable as LVDS pair
10	PB1_P	B2B connector JB2, pin 41	AE9

16 Appendix: TE0706-03 carrier board details



1. 5V power connector jack, J1
2. Reset switch, S2
3. USB2.0 type A receptacle, J7
4. Micro SD card socket with Card Detect, J4
5. 50 pin IDC male connector, J5
6. 1000Base-T Gigabit RJ45 Ethernet MagJack, J3, Arm A53 1Gb Ethernet line
7. 1000Base-T Gigabit RJ45 Ethernet MagJack, J2, Ethernet line for designs in PL
8. XMOD JTAG- / UART-header, JX1
9. User DIP-switch, S1
10. VCCIO selection jumper block, J10 - J12
11. External connector (VG96) placeholder, J6
12. Samtec Razor Beam™ LSHM-150 B2B connector, JB1
13. Samtec Razor Beam™ LSHM-150 B2B connector, JB2
14. Samtec Razor Beam™ LSHM-130 B2B connector, JB3

Figure 14: TE0706-03 Carrier Board.

Figure 14 presents main components and connector locations of the TE0706-03 Carrier Board. The evaluation package released together with this application note supports single 1000Base-T Gigabit RJ45 Ethernet MagJack, J3 as Arm A53 PetaLinux eth0. See Figure 14. See https://wiki.trenz-electronic.de/display/PD/TE0706+TRM for source of the photo and for detailed description of the TE0706-03 carrier board.

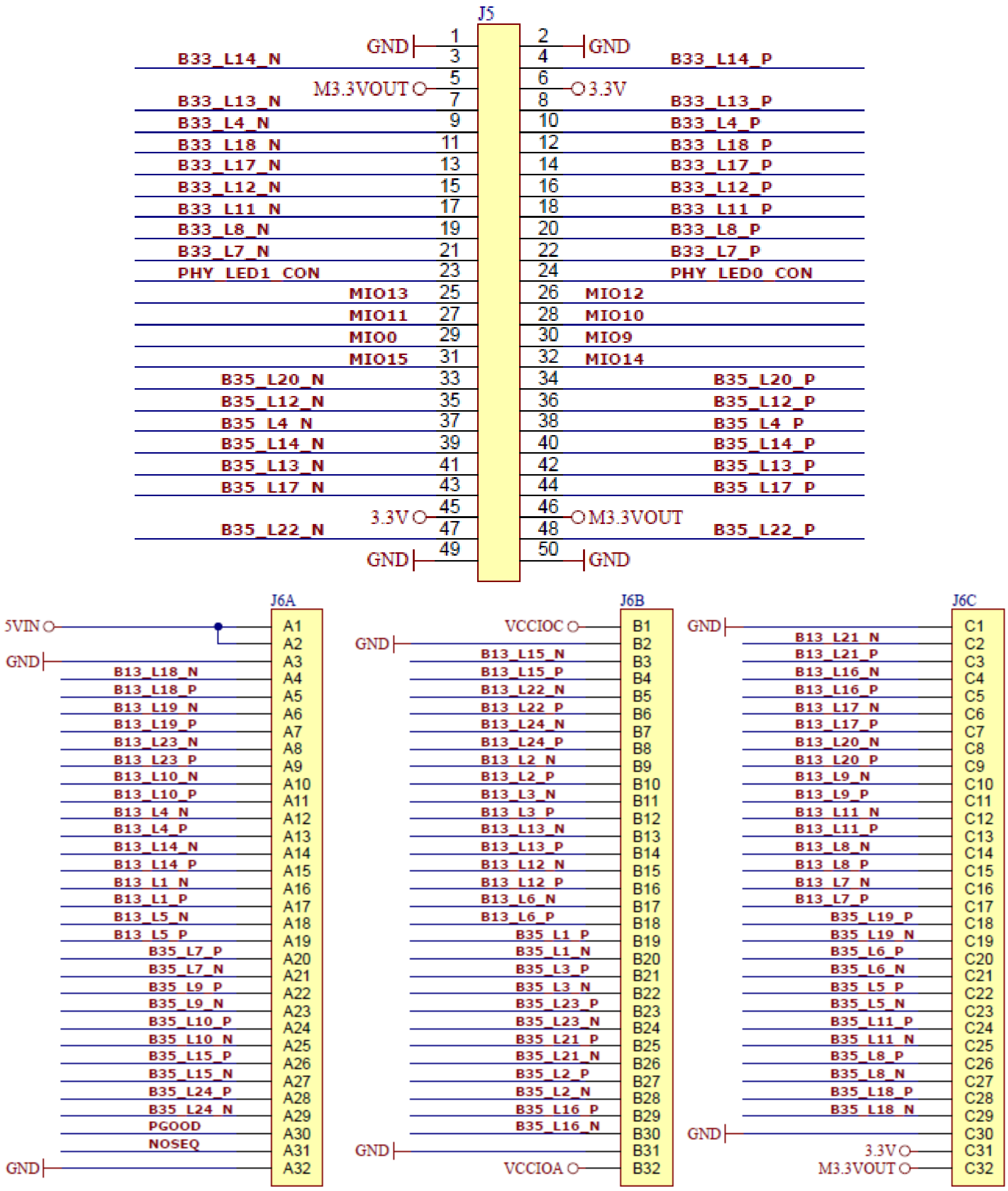
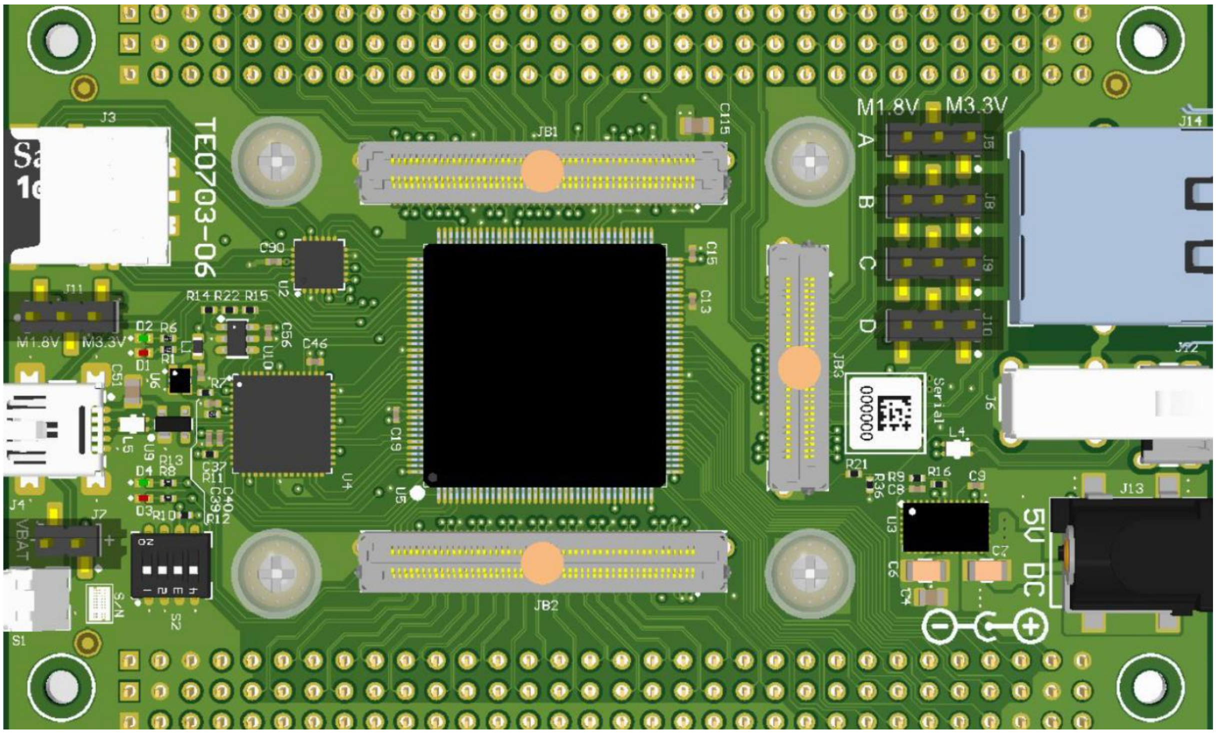


Figure 15: Connection of PCBs data lines to connectors on TE0706-03 carrier board

17 Appendix: TE0703-06 carrier board



1. Samtec Razor Beam™ LSHM-150 B2B connector, JB1
2. Samtec Razor Beam™ LSHM-150 B2B connector, JB2
3. Samtec Razor Beam™ LSHM-130 B2B connector, JB3
4. Micro SD card socket with detect switch, J3
5. LED indicators D1 and D2
6. Mini-USB type B connector, J4
7. LED indicators D3 and D4
8. Configuration DIP switches, S2
9. User push button (Reset), S1
10. External connector (VG96) placeholder, J1
11. External connector (VG96) placeholder, J2
12. VCCIO voltage selection jumper block, J5, J8, J9 and J10
13. Trxcom 1000Base-T Gigabit RJ45 Magjack, J14
14. USB type A receptacle, J6 (optional micro USB 2.0 type B receptacle available, J12)
15. 5V power connector jack, J13

Figure 16: TE0703-06 Carrier Board.

Figure 16 presents main components and connector locations of the TE0703-06 Carrier Board [3]. The precompiled designs can be used without modification on the TE0703-06. See https://wiki.trenz-electronic.de/display/PD/TE0703+TRM for source of the photo and for description of the TE0703-06 carrier board.

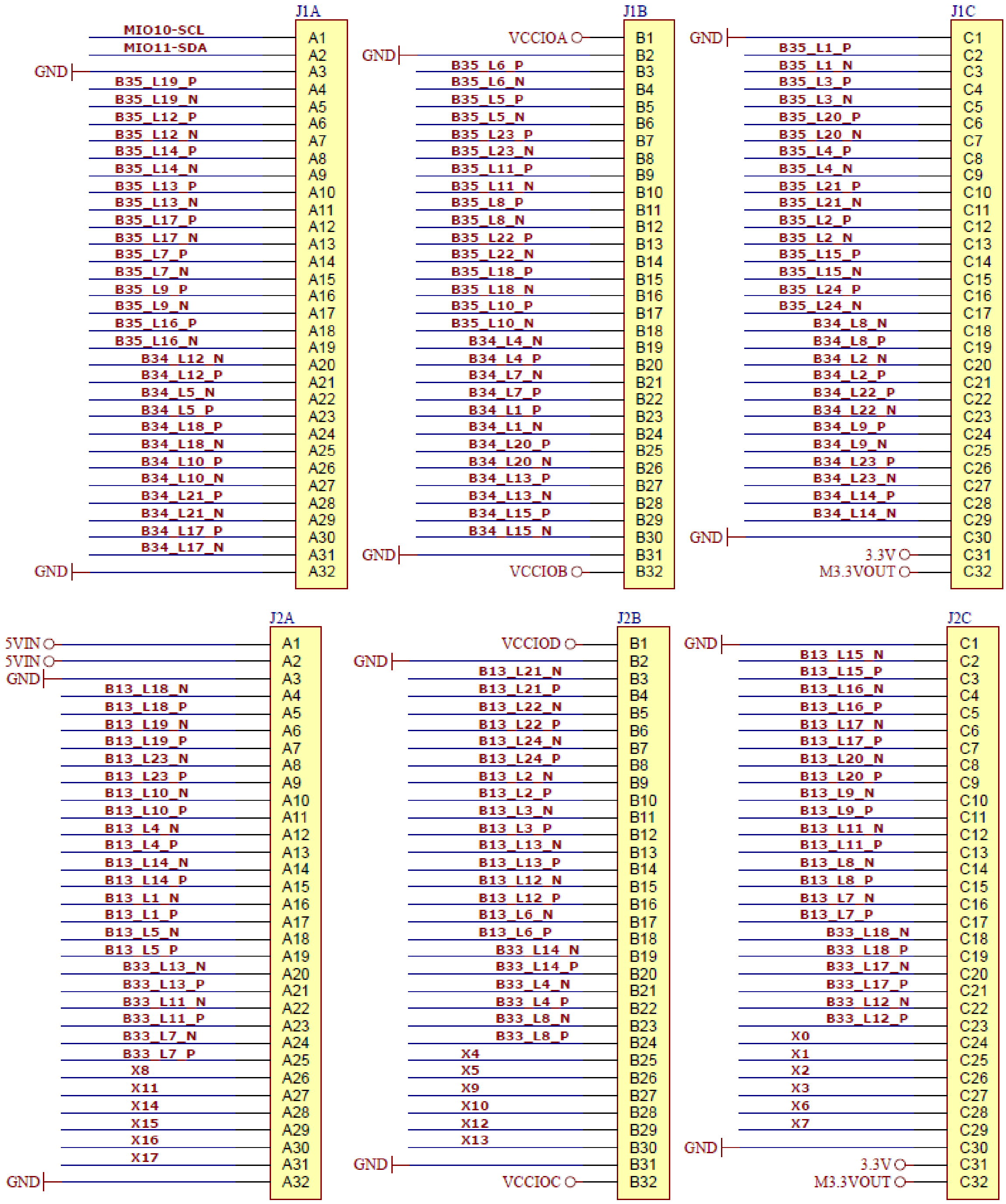


Figure 17: Connection of PCBs data lines to connectors on TE0703-06 carrier board

Figure 17 describes connection of PCBs data lines to the connectors on the TE0703-06 carrier board and Figure 15 for the TE0703-06 carrier board. Table 6 describes the common connections of Zynq pins to TE0703-06 and TE0706-03 PCB data lines. Users of the development package can use these data for creation of own user constrains and extend the Vivado 2018.2 HW projects generated by the SDSoC 2018.2 design environment.

18 Appendix: FPGA pins on TE0706-03 and TE0703-06

Table 6: Connections of Zynq Ultrascale+ pins TE0703-06 and TE0706-03

SFVC784 B66 Pin	Carrier TE0703-06 TE0706-03	SFVC784 B65 Pin	Carrier TE0703-06 TE0706-03	SFVC784 B64 Pin	Carrier TE0703-06 TE0706-03	SFVC784 B65&B505 Pin	Carrier TE0703-06
F5	B35_L16_N	H9	B33_L7_P	AB3	B13_L7_P	B23	B34_L7_P
G5	B35_L16_P	H8	B33_L7_N	AB4	B13_L7_N	B24	B34_L7_N
C8	B35_L24_N	R8	B33_L8_P	AB2	B13_L8_P	C25	B34_L2_P
B8	B35_L24_P	T8	B33_L8_N	AC2	B13_L8_N	C26	B34_L2_N
G3	B35_L18_N	M6	B33_L11_P	AC4	B13_L11_P	D23	B34_L4_P
F3	B35_L18_P	L5	B33_L11_N	AC3	B13_L11_N	D24	B34_L4_N
B6	B35_L15_N	L3	B33_L12_P	AB1	B13_L9_P	E25	B34_L5_P
C6	B35_L15_P	L2	B33_L12_N	AC1	B13_L9_N	E26	B34_L5_N
B1	B35_L22_N	L8	B33_L17_P	AD2	B13_L20_P	F23	B34_L12_P
C1	B35_L22_P	M8	B33_L17_N	AD1	B13_L20_N	F24	B34_L12_N
D1	B35_L17_N	J7	B33_L18_P	AE3	B13_L17_P	U8	B34_L8_P
E1	B35_L17_P	H7	B33_L18_N	AF3	B13_L17_N	V8	B34_L8_N
D5	B35_L13_N	P7	B33_L4_P	AE2	B13_L16_P	N9	B34_L9_P
E5	B35_L13_P	P6	B33_L4_N	AF2	B13_L16_N	N8	B34_L9_N
C4	B35_L14_N	L1	B33_L13_P	AG6	B13_L18_P	K4	B34_L22_P
D4	B35_L14_P	K1	B33_L13_N	AG5	B13_L18_N	K3	B34_L22_N
G1	B35_L4_N	N7	B33_L14_P	AG4	B13_L15_P	A25	B34_L1_P
F1	B35_L4_P	N6	B33_L14_N	AH4	B13_L15_N	A26	B34_L1_N
C2	B35_L12_N			AG3	B13_L21_P	B27	B34_L18_P
C3	B35_L12_P			AH3	B13_L21_N	B28	B34_L18_N
A2	B35_L20_N			AB8	B13_L5_P	D27	B34_L20_P
A1	B35_L20_P			AC8	B13_L5_N	D28	B34_L20_N
E9	B35_L10_N			AC9	B13_L6_P	F27	B34_L10_P
D9	B35_L10_P			AD9	B13_L6_N	F28	B34_L10_N
G8	B35_L9_N			AE9	B13_L1_P	W8	B34_L21_P
F7	B35_L9_P			AE8	B13_L1_N	Y8	B34_L21_N
E8	B35_L7_N			AF7	B13_L12_P	R7	B34_L15_P
F8	B35_L7_P			AF6	B13_L12_N	T7	B34_L15_N
G6	B35_L2_N			AE5	B13_L14_P	U9	B34_L17_P
F6	B35_L2_P			AF5	B13_L14_N	V9	B34_L17_N
F2	B35_L8_N			AD5	B13_L13_P	R6	B34_L23_P
E2	B35_L8_P			AD4	B13_L13_N	T9	B34_L23_N
E4	B35_L21_N			AG9	B13_L4_P	L7	B34_L14_P
E3	B35_L21_P			AH9	B13_L4_N	L6	B34_L14_N
D6	B35_L11_N			AF8	B13_L3_P		
D7	B35_L11_P			AG8	B13_L3_N		
C9	B35_L23_N			AH8	B13_L10_P		
B9	B35_L23_P			AH7	B13_L10_N		
A8	B35_L5_N			AE7	B13_L2_P		
A9	B35_L5_P			AD7	B13_L2_N		
A7	B35_L3_N			AB7	B13_L23_P		
A6	B35_L3_P			AC7	B13_L23_N		
A5	B35_L6_N			AB6	B13_L24_P		
B5	B35_L6_P			AC6	B13_L24_N		
A4	B35_L1_N			AF1	B13_L19_P		
B4	B35_L1_P			AG1	B13_L19_N		
A3	B35_L19_N			AH2	B13_L22_P		
B3	B35_L19_P			AH1	B13_L22_N		

19 Appendix: Configuration of switches and jumpers

Configuration of the TE0701-06 board

Power supply: DC 12V/3A.

- Set User 4-bit DIP switch, S3 to: **1=OFF; 2=OFF; 3=ON; 4=OFF**
- User 4-bit DIP switch, S4 to: **1=ON; 2=OFF; 3=ON; 4=OFF**
- Set Jumpers (indicted by [ ]) to 1.8V to all Zynq Ultrascale+ pin banks and to FMC:
  - **J9: [1-2] 3**
  - **J16 [1-2]**
  - **J17: 1 2 3 (no jumper)**
  - **J18: 1 2 (no jumper)**
  - **J19: [1-2]**
  - **J20: [1-2]**
  - **J21: 1 [2-3]**

Configuration of the TE0703-06 board

Power supply: DC 5V/4A.

- Set jumpers of the **TE0703-06** board to(indicted by [ ]) to 1.8V
  - **J5: 1 [2-3] (1.8V) VCCIOA**
  - **J8: 1 [2-3] (1.8V) VCCIOB**
  - **J9: 1 [2-3] (1.8V) VCCIOC**
  - **J10: 1 [2-3] (1.8V) VCCIOD**
  - **J11: 1 [2 3] (3.3V)** FPGA side of the SD card level shifter for TE0820 module
- Set FPGA side of the SD card level shifter voltage by jumper J11 to 3.3V:
- Set switch **S1** of the **TE0706-03** board to: **1=OFF; 2=ON; 3=ON; 4=ON**

Configuration of TE0706-03 board

Power supply: DC 5V/4A.

- Set jumpers of the **TE0706-03** board to:
  - **J10: [1-2]-3 (1.8V) VCCIOA**
  - **J11: [1-2]-3 (1.8V) VCCIOB**
  - **J12: [1-2]-3 (1.8V) VCCIOC**
  - **J13: 1 [2-3] (3.3V)** FPGA side of the SD card level shifter for TE0820 module
- Set switch S1 of the **TE0706-03** board to: **1=ON; 2=ON; 3=ON; 4=OFF**

Configuration of TE0790-02 xmod adapter for the TE0706-03 board

The TE0706-03 board ARM serial terminal/JTAG is connected to the PC by a Mini USB (type B) cable via the **TE0790-02** XMOD FTDI JTAG adapter [5].

- Set switch in the XMOD module to: **1=ON; 2=OFF; 3=ON; 4=OFF;**

The TE0790-02 xmod adapter generates its required local 3.3V power supply from the PC 5V power supply (present in the USB cable) by an on-module DC2DC power converter.

20 Appendix: Package content

```
├── debian
│   ├── mkdebian.sh
│   ├── image.ub
│   ├── u-boot.elf
│   ├── bl31.elf
│   └── te0820-debian.zip
└── zynq
    ├── TE0820_zusys_SDSoC_TE0701_TE0703_TE0706.zip
    ├── ProviderExample.cpp
    └── 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

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.