Application Note

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

Arrowhead Client on ZynqBerry with SDSoC 2018.2 HW Accelerators Installation for Win7 and Win10

Jiri Kadlec, Lukáš Kohout
kadlec@utia.cas.cz kohoutl@utia.cas.c

Revision history

Rev.	Date	Author	Description
0	1.04.2019	J. Kadlec	Installation for Win7 and Win10
1			
2			

© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved.

Contents

Acknowledgement

This work has been partially supported from project Productive4.0, project number ECSEL 737459.

© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

1 Introduction

This application note describes an installation procedure of Arrowhead client on Zynq 7000 device with support for the Xilinx SDSoC 2018.2 HW accelerators. The concrete board is ZynqBerry TE0726-03M. It works with Xilinx XC77010-1C device with the dual core Arm A9 32 bit and programmable logic on single 28 nm chip. The ZynqBerry PCB has RaspberryPi 2 form factor. The ZynqBerry board is designed and manufactured by company Trenz Electronic [1]. The device runs Xilinx PetaLinux 2018.2 kernel with Debian 9.8 Stretch distribution (03.25.2019). The client SW acts as a *Producer* of a service or as a *Consumer* requesting the service from an Arrowhead framework. The base hardware platform for the Zynq device is compiled with Xilinx Vivado 2018.2 tool. The entire installation procedure has been tested on Win 7 Pro and Win 10 PC. The optional configuration of the Petalinux 2018.2 kerrnel and the optional generation of the Debian image are performed on the Ubuntu 16.04 LTS host. The Ubuntu OS can be executed on the same 64 bit Win7 or Win 10 PC in the VMware Workstation 14 Player. To run and test Arrowhead clients, it is required to have running Arrowhead-framework G4.0 light-weight installation running on a RaspberryPi 3B board (RPi3).

2 HW configuration with simple Arrowhead Client example

The targeted HW works with one RPi3 board (bottom) and two ZynqBerry boards (above). The RPi3 implements the Arrowhead framework. See [2] for the documentation. The Producer ZynqBerry on the top board hosts C++ provider capable to measure the actual temperature of the Xilinx XC77010-1C device. The Consumer ZynqBerry in the middle hosts C++ consumer capable to ask the Arrowhead framework about the temperature provided as service by the Producer ZynqBerry board. Zynq boards host HW accelerators of Matrix MultiplyAdd (20x20 int32 matrices), delivering approximately 4x shorter execution time in comparison to the optimized SW running on the 650 MHz Arm Cortex A9 processor.

Figure presents: RPi3 (bottom), and two ZynqBerry boards with the Arrowhead framework G4.0 compatible C++ clients running on ZynqBerry boards

© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

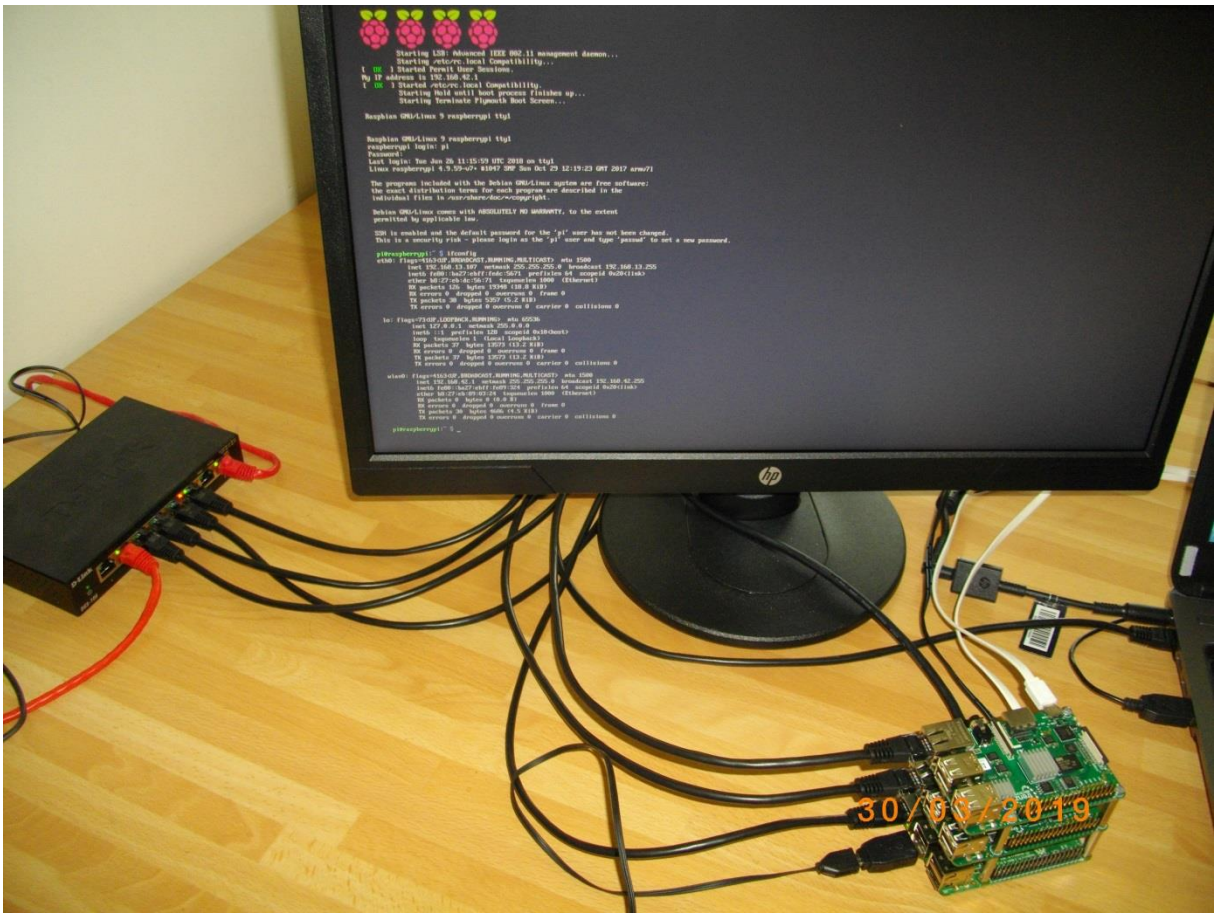
3 Installation of Arrowhead Framework Services on RPi3

Testing and running of the Arrowhead C++ clients on ZynqBerry boards requires Ethernet access to the Arrowhead framework services. It is recommended to use the precompiled image for the RPi3 board. It includes already installed and configured Arrowhead framework G4.0 lightweight implementation. The image is available as one of results of the work package WP1 of the running ECSEL JU project Productive4.0 https://productive40.eu/.

It is accessible for all consortium project partners from the project ownCloud repository https://productive4-cloud.automotive.oth-aw.de/index.php/login . Files are present in section WP1, task 1.4. Please contact coordinator of the consortium for further information about the access to the Arrowhead-framework G4.0 light-weight installation running on the RPi3 board. After receiving the access to the download, unzip the three downloaded files *Arrowhead-40-raspi.z01*, *Arrowhead-40-raspi.z02* and *Arrowhead-40-raspi.zip* into the final image file *image_180626.img* (size 3.711.959.040 Bytes).

Copy the RPi3 image *image_180626.img* to (at least) 4GB SD card (speed grade 10). You can use the *Win32DiskImager* utility from: https://sourceforge.net/projects/win32diskimager/ .

Connect the RPi3 to USB keyboard, HDMI monitor with inserted SD card. Connect it to Ethernet with the DHCP server. Power ON the board by connecting the 5V power supply via micro USB cable. Power can be provided from the PC via the USB port or, preferably, from the dedicated 5V power supply.



The RaspberryPi 3 will boot from the SD card image with text output to the HDMI monitor.

© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

Login as user:

```
pi
```

Password:

```
raspberry
```

Find and write down the assigned Ethernet IP address for IP V4 and IP V6 by typing on the RPi3 keyboard:

```
ifconfig
```

To shutdown properly the RPi3 type on the RPi3 keyboard:

```
sudo halt
```

The OS will shutdown and all possibly open R/W operations to the SD card are closed. Remove temporarily the SD card and disconnect the 5V power to switch OFF the board. Return the SD card to RPi3 slot.

4 Create SDSoC platform for Arrowhead compatible ZynqBerry boards

The Xilinx SDSoC 2018.2 compiler requires preparation of SDSoC platform. It is specific Vivado 2018.2 design with metadata, enabling to the SDSoC LLVM system level compiler to add additional HW accelerator blocks and data movers on top of the initial Vivado design. These HW blocs are defined as C/C++ user defined functions. These functions can be compiled, debugged and executed in Petalinux user space on ARM A9. But in addition, the selected C/C++ functions can be compiled also to form of Vivado HLS HW accelerators, compiled by the Vivado HLS compiler and automatically interfaced with dedicated data movers like DMA or SG DMA. The resulting compiled system remains compatible with the Debian OS and C++ Arrowhead Clients.

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 ZynqBerry board. You have to have the Xilinx SDSoC 2018.2 installed on your PC. Use the SDSoC 2018.2 web installer for Windows 64 (EXE - 50.58 MB) 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: "ZynqBerry 512 MByte DDR3L and SDSoC Voucher". See [1]:
https://shop.trenz-electronic.de/en/27229-Bundle-ZynqBerry-512-MByte-DDR3L-and-SDSoC-Voucher?c=350
We will use the ZynqBerry 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 HW bit-stream for the programmable part of the design 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 ZynqBerry boards.

Image files included in this evaluation package can be used for quick first evaluation of the development flow of the SDSoC platform.

The complete configuration/compilation of Petalinux kernel and Debian image is skipped at this stage, but it is described in the second part of this application note (Chapters 13 and 14).

To prepare the ZynqBerry SDSoC board support package follow these steps:

© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

1. Unpack the enclosed evaluation package *TE0726_zsys_SDSoC.zip* to Win 7 or Win10 directory of your choice. We will use:

   ```
   c:\TS82\TE0726_Debian_Arrowhead_Client\
   ```

   It will create *TE0726_zsys_SDSoC* folder.

2. On Win 7 or Win10, open dos terminal window, go to the *TE0726_zsys_SDSoC* folder and create an initial setup:

   ```
   cd c:\TS82\TE0726_Debian_Arrowhead_Client\TE0726_zsys_SDSoC
   _create_win_setup.cmd
   ```

   Select option (1) to create maximum setup of CMD-Files and to exit.
   Set of scripts is created in the *TE0726_zsys_SDSoC* 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:\TE0726_zsys_SDSoC* 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 TE0726_zsys_SDSoC
   ```

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=3
   ```

   The selected number corresponds to the number defined in file
   X:*TE0726_zsys_SDSoC\board_files/TE0726_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:\TE0726_zsys_SDSoC\vivado_create_project_guimode.sh
   ```

   Next figure shows block design of the created system. It includes 4 HW reset IPs for future HW accelerators with system clocks 50 MHz, 64 MHz, 74 MHz or 100 MHz.

   The DDR3 interface and the connections to the USB ports for keyboard, mouse and 100Mbit Ethernet are all pre-configured inside of the Vivado Zynq block.

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 *TE0726_zsys_SDSoC.hdf* is generated in folder:

   ```
   X:\TE0726_zsys_SDSoC\prebuilt\hardware\m\TE0726_zsys_SDSoC.hdf
   ```

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

   ```
   X:\TE0726_zsys_SDSoC\prebuilt\os\petalinux\default\image.ub
   X:\TE0726_zsys_SDSoC\prebuilt\os\petalinux\default\u-boot.elf
   ```

We skip the optional Petalinux and Debian configuration and compilation steps at this stage.

Steps will be described in Chapters 13 and 14.

© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

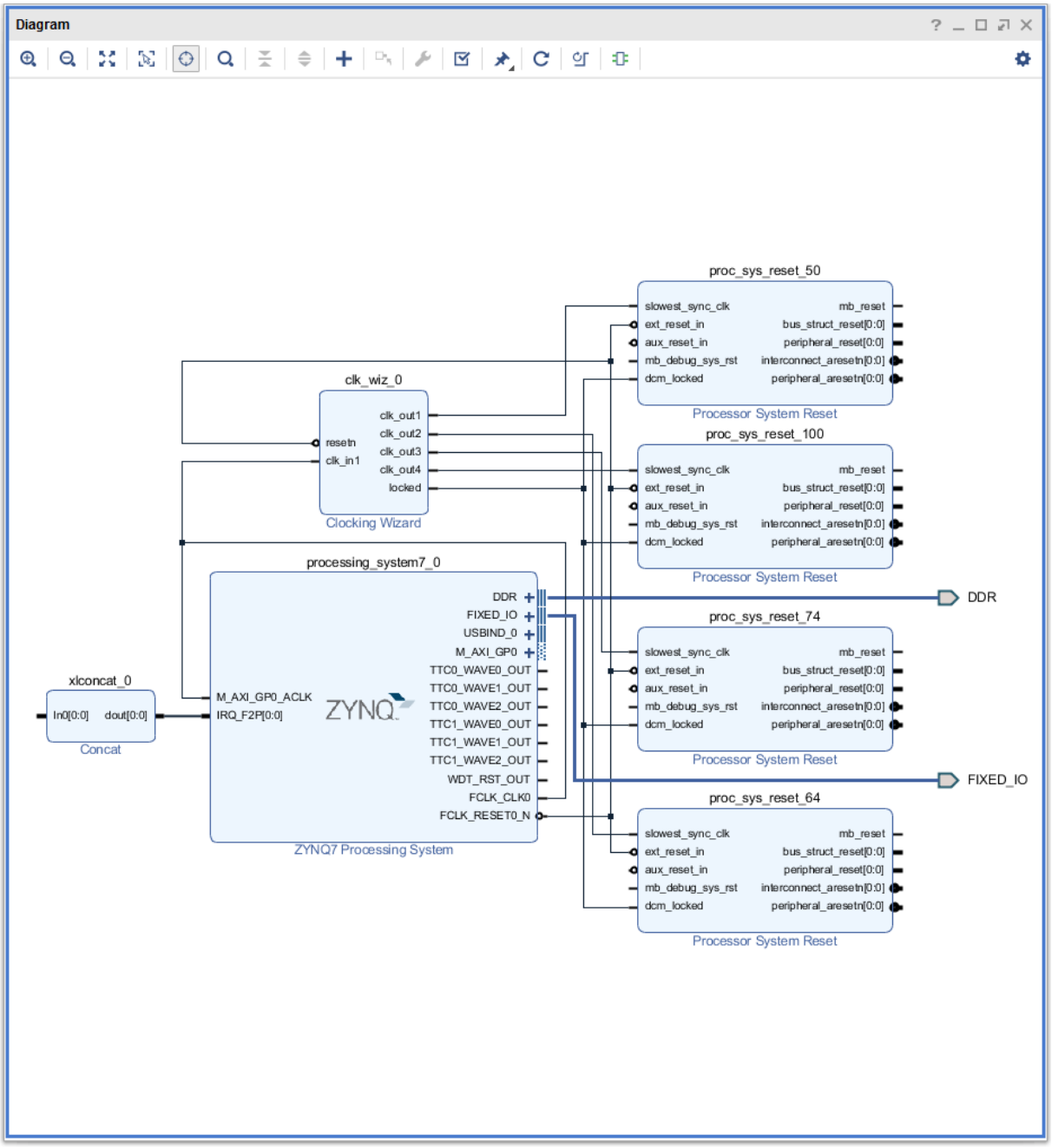


Figure describes the initial Vivado design. It defines the SDSoC 2018.2 platform.

5 Create SDSoC 2018.2 platform

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:\TE0726_zsys_SDSoC\prebuilt\boot_images\m\u-boot\BOOT.bin
   ```

2. To prepare for the SDSoC 2018.2 platform generation, move the created support file

   ```
   X:\TE0726_zsys_SDSoC\prebuilt\software\m\zynq_fsbl_flash.elf
   ```

   to

   ```
   X:\zynq_fsbl_flash.elf
   ```


© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

3. 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\te0726\03m\
   ```
   and it is also packed into the ZIP file like:
   ```
   X:\SDSoC_PFM\te0726\SDSoC_PFM_te0726-03m_20190330103933.zip
   ```

6 Compile HW accelerator to new BOOT.bin by the SDSoC 2018.2 compiler

4. Open SDSoC project in directory
   ```
   X:\SDSoC_PFM\te0726\03m\
   ```
5. In SDSoC select platform:
   ```
   X:\SDSoC_PFM\te0726\03m\TE0726_zsys_SDSoC
   ```
6. Create new project named
   ```
   te06_1
   ```
7. Select template project
   ```
   X:\SDSoC_PFM\te0726\03m\TE0726_zsys_SDSoC\samples\direct_connect
   ```
   and compile it for the *Release* target with all clocks set to 100 MHz.
8. The SDSoC compiler will create these relevant results in the *sd_card* directory
   ```
   X:\SDSoC_PFM\te0726\03m\te06_1\Release\sd_card\BOOT.BIN
   X:\SDSoC_PFM\te0726\03m\te06_1\Release\sd_card\te06_1.elf
   ```
9. To prepare for the programming of generated BOOT.BIN to the Qspi flash on the ZynqBerry board move back the temporarily moved file from the directory
   ```
   X:\zynq_fsbl_flash.elf
   ```
   back to the folder
   ```
   X:\TE0726_zsys_SDSoC\prebuilt\software\m\zynq_fsbl_flash.elf
   ```
   It will be needed fo programming of the Qspi FLASH of the ZynqBerry board.
10. Copy the created *BOOT.BIN* file it to a newly created *NA* directory (New Application):
    X:*TE0726_zsys_SDSoC\prebuilt\boot_images\m\NA\BOOT.bin*
11. Connect the ZynqBerry board to the Ethernet.
12. From the open Win 7 or Win 10 console execute command:
    ```
    design_clear_design_folders.cmd
    ```
    This script cleans all created Vivado 2018.2 design subfolders.
13. Unzip the preconfigured and precompiled Debian image for the ZynqBerry board from from this evaluation package file: *te0726-debian.zip* (598.913.412 Bytes) to the file *te0726-debian.img* (7.516.192.768 Bytes).
14. Use again the *Win32DiskImager* tool for creation of the image *te0726-debian.img* on the SD card. Use 8GB SD with speed grade 10.
15. Copy to the root of the SD card the HW accelerated matrix multiplication demo executable *te06_l.elf* from the directory:
    ```
    X:\SDSoC_PFM\te0726\03m\te06_1\Release\sd_card\te06_1.elf
    ```
16. Insert created SD card to the ZynqBerry board.
17. Connect the ZynqBerry board with your Win7 or Win 10 PC via micro USB cable. The USB cable provides the 5V power supply, the programming interface and console terminal. Use *putty* or similar terminal client with *speed (baud) 115200bps, data bits*

© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

8, stop bits 1, parity none and flow control none. The actual COM port number associated with your connection can be found in the windows *Device manager*.

18. You have to write the X:*TE0726_zsys_SDSoC\prebuilt\boot_images\m\NA\BOOT.bin* file to the ZynqBerry on-board Qspi FLASH. It is needed for the initial stage of the booting procedure of the Xilinx xc7z010 device present on the ZynqBerry board. From the open Win 7 or Win 10 console execute this command:
    ```
    program_flash_binfile.cmd
    ```
19. The programming of the Qspi will start. It will be followed by automatic reset of the Zynq board.
20. You can install and use *putty* terminal https://www.putty.org/
21. The ZynqBerry board will automatically boot from the newly programmed on board Qspi flash. The first stage boot loader (fsbl) program is executed first. It loads to DDR3 and starts the u-boot program. The u-boot program will download the bitstream, 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 the asci output to the serial terminal. The preconfigured Debian file system is present on the separate partition of the SDcard.
22. Login as user:
    ```
    root
    ```
    Password:
    ```
    root
    ```
23. Find and write down the assigned Ethernet IP address for IP V4 and IP V6 by typing command:
    ```
    ifconfig
    ```
    The HW accelerated matrix multiplication demo can be executed on both Zynqberry boards from the automatically mounted SD by executing:
    ```
    /boot/te06_1.elf
    ```
    See the HW acceleration measured by the number of Arm A9 clock cycles.
24. To shutdown properly the ZynqBerry 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 5V power to switch OFF the board. Return back the SD card.

The SDSoC compiler have created and compiled new HW accelerator to the programmable logic part of the device from the C++ source code *mmult.cpp.* 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
               )
{
```


© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

```
    // 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=400
        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=324 max=400
        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=20
        mmult2: for (int j = 0; j < dim ; j++) {
        #pragma HLS PIPELINE
        #pragma HLS LOOP_TRIPCOUNT min=1 max=20
           int result = 0;
           mmult3: for (int k = 0; k < DATA_SIZE; k++) {
           #pragma HLS LOOP_TRIPCOUNT min=1 max=20
               result += A[i][k] * B[k][j];
           }
           out[i * dim + j] = result;
        }
    }
}
```


© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

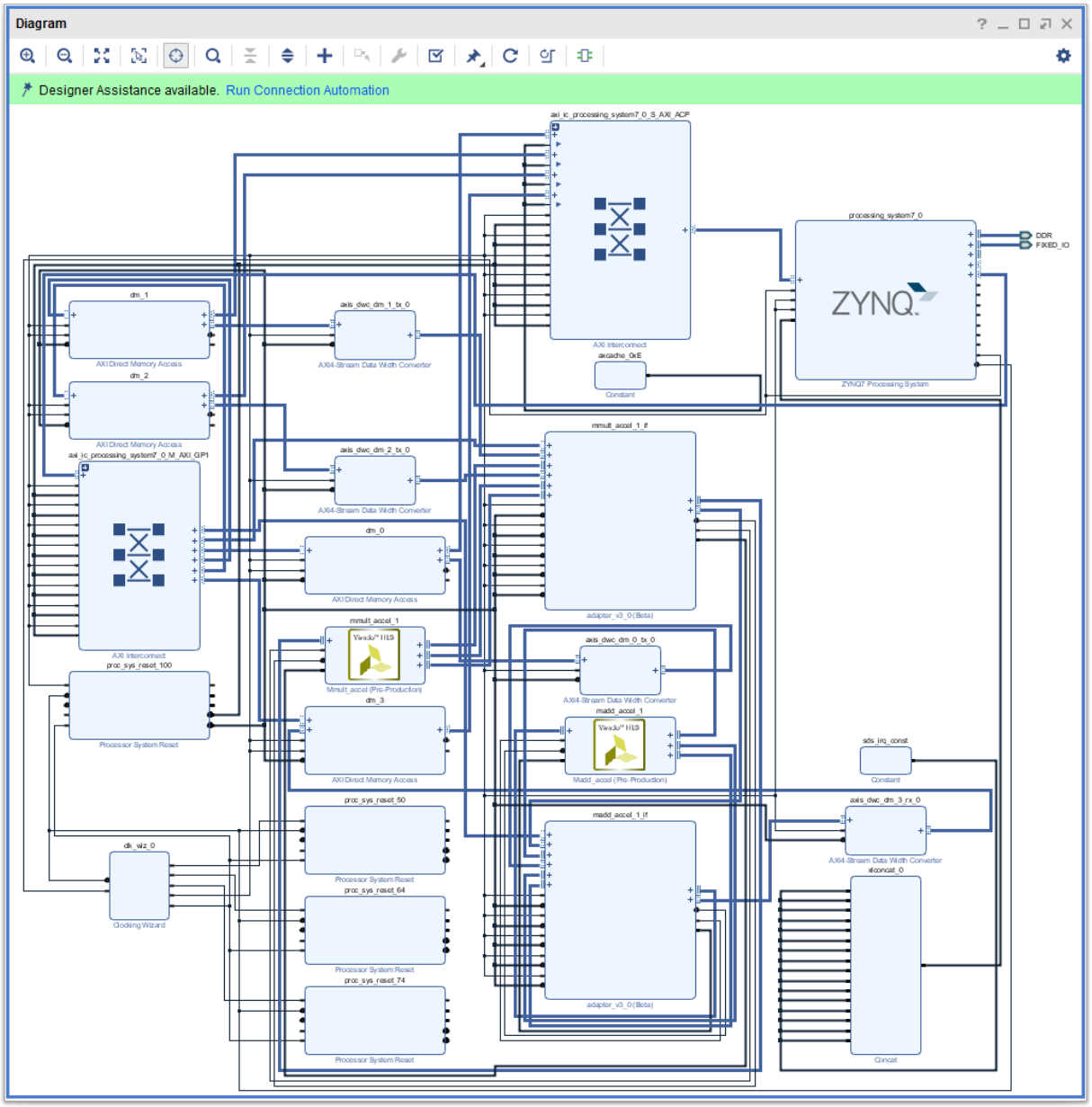


Figure describes the new SDSoC 2018.2 generated Vivado project with HW accelerator.

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 ZynqBerry board. The design includes the two Vivado HLS HW accelerators for matrix (20x20 int32) multiplication and for matrix (20x20 int32) addition. Both accelerators operate at 100 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 ZynqBerry board.

Repeat steps 1-24 also for the second ZynqBerry board.

© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

7 Install Arrowhead-f support on ZynqBerry boards

At this stage, the Debian OS present on both Zynqberry boards can be upgraded to become compatible with the Arrowhead framework G4.0 client and provider C++ demo applications.

1. Start the RPi3 board, both ZynqBerry boards and Win7 or Win 10 PC.
2. Identify and write down the Ethernet addresses set by the HDCP server. The network has to support access to the external Ethernet to get access to the repositories.

   In Win7 or Win 10 PC use WinSCP or similar tool to copy the arrowhead installation script *install-arrohead-cli-dep.sh* from this evaluation package to this folder of each of the two ZynqBerry boards:

   ```
   /root/install-arrohead-cli-dep.sh
   ```
3. To control the ZynqBerry boards, use two SSH (preferred) or serial terminals of your Win7 or Win 10 PC. Use again: user *root* pswd *root*
4. To upgrade the Debian installations and to install the dependencies required by the Arrowhead C++ clients, execute on each ZynqBerry board these commands:

   ```
   cd /root
   chmod ugo+x install-arrohead-cli-dep.sh
   ./install-arrohead-cli-dep.sh
   ```

8 Install Arrowhead-f C++ Provider on ZynqBerry

To control the ZynqBerry device, use SSH (preferred) or serial terminal.

1. Get the Arrowhead client source code. The sources include C++ version of the Arrowhead *Provider* and *Client* skeletons.

   ```
   cd /root
   git clone https://github.com/arrowhead-f/client-cpp
   ```
2. Compile Arrowhead *ProviderExample*.

   ```
   cd client-cpp/ProviderExample
   make
   ```
3. Modify the *ProviderExample* configuration file *ApplicationServiceInterface.ini*

   ```
   mcedit ApplicationServiceInterface.ini
   ```

   The configuration file consists of the following items.

   - sr_base_uri – an address of the Arrowhead registration service running in insecure mode, in our case it is the RPi3 IP address with port 8440.
   - sr_base_uri_https – an address of the Arrowhead registration service running in secure mode, in our case it is the RPi3 IP address with port 8441.
   - port – a port number where the *Provider* will be available on, set 8000.
   - address – *Provider* IP address, ZynqBerry IP.
   - Address6 - *Provider* IP address in IPV6

   The *ProviderExample* configuration file example:

   ```
   [Server]
   sr_base_uri="http://10.42.0.141:8440/serviceregistry/"
   sr_base_uri_https="https://10.42.0.141:8441/serviceregistry/"
   port="8000"
   address="10.42.0.103"
   address6="[fe80::483b:e5ff:fe7f:610d]"
   ```

   Safe the file (F2) and exit the editor (F10).
4. Start the *ProviderExample*

© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

```
./ProviderExample
```

The *ProvidedExample* registers itself in the Arrowhead framework database. On *Consumer* request, it returns an artificial temperature, fixed to value 26 degrees Celsius.

9 Install Arrowhead-f C++ Consumer on ZynqBerry

The Arrowhead *ConsumerExample* can be compiled and run on the second ZynqBerry board. Alternatively, the *ConsumerExample* can be compiled and tested on the same ZynqBerry board as the *ProviderExample*.

1. Compile Arrowhead *ConsumerExample*.

```
cd /root/client-cpp/ConsumerExample
make
```

2. Configure the *ConsumerExample*. There are two configuration files: *OrchestratorInterface.ini* and *consumedServices.json*.
   a. *OrchestratorInterface.ini*

```
mcedit OrchestratorInterface.ini
```

The configuration file consists of the following items.

- or_base_uri – an address of the Arrowhead orchestrator service running in insecure mode, in our case it is the RPi3 IP address with port 8440.
- sr_base_uri_https – an address of the Arrowhead orchestrator service running in secure mode, in our case it is the RPi3 IP address with port 8441.
- port – a port number where the *Consumer* will be available on, set 8002.
- address – *Consumer* IP address, ZynqBerry IP.
- address6 - *Consumer* IP address in IPV6

The configuration file example:

```
[Server]
or_base_uri="http://10.42.0.141:8440/orchestrator/orchestration"
or_base_uri_https="https://10.42.0.141:8441/orchestrator/orchestration"
port="8002"
address="10.42.0.103"
address6="[fe80::483b:e5ff:fe7f:610d]"
```

Safe the file (F2) and exit the editor (F10).

   b. *consumedServices.json*

```
mcedit consumedServices.json
```

Modify the following items in the file:

- requestForm/requesterSystem/port – Number of the *Consumer* port.
- Modify line

```
"security" : ""
```

- preferredProviders/providerSystem/address – Preferred *Provider* IP address.
- preferredProviders/providerSystem/port – Port number, where the preferred *Provider* listen on.

This configuration file should look like this:

© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

```
{
  "consumerID": "TestconsumerID",
  "requestForm": {
    "requesterSystem": {
      "systemName": "client1",
      "address": "dontcare",
      "port": 8002,
      "authenticationInfo": "null"
    },
    "requestedService": {
      "serviceDefinition": "IndoorTemperature_ProviderExample",
      "interfaces": ["REST-JSON-SENML"],
      "serviceMetadata":{
        "security" : ""
      }
    },
    "orchestrationFlags": {
      "overrideStore" : true,
      "matchmaking" : true,
      "metadataSearch" : false,
      "pingProviders" : false,
      "onlyPreferred" : true,
      "externalServiceRequest" : false
    },
    "preferredProviders": [{
      "providerSystem":{
        "systemName": "SecureTemperatureSensor",
        "address": "10.42.0.103",
        "port":"8000"
      }
    }]
  }
}
```

Save the file (F2) and exit the mcedit editor (F10).

The Debian midnight commander tool can be started from the command line by typing:

```
mc -s
```

The two putty console programs connect via USB to the two ZynqBerry boards and display the Ethernet address automatically assigned by the DHCP server.

Run the *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"}
```


© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

The *ConsumerExample* will fail in the first instance. 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. This is described next.

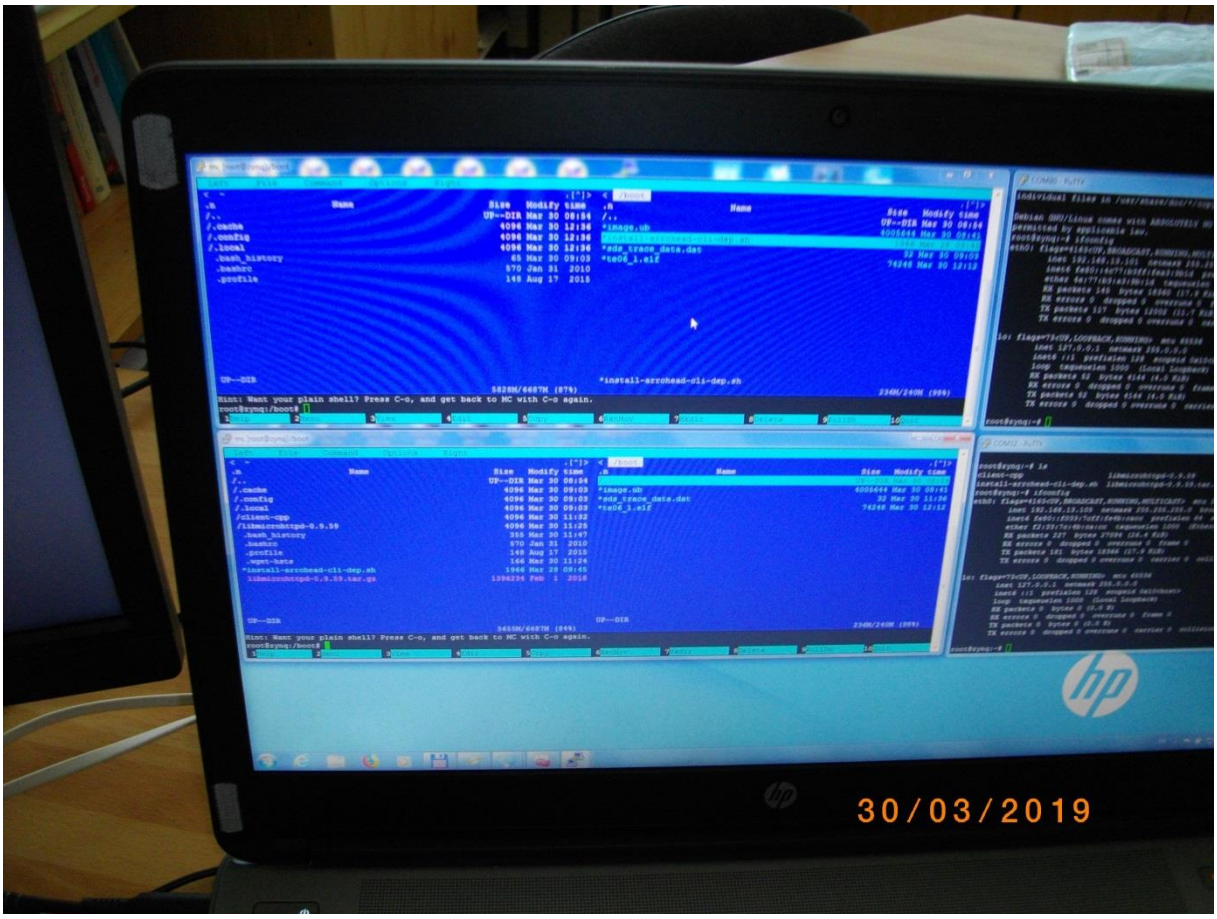
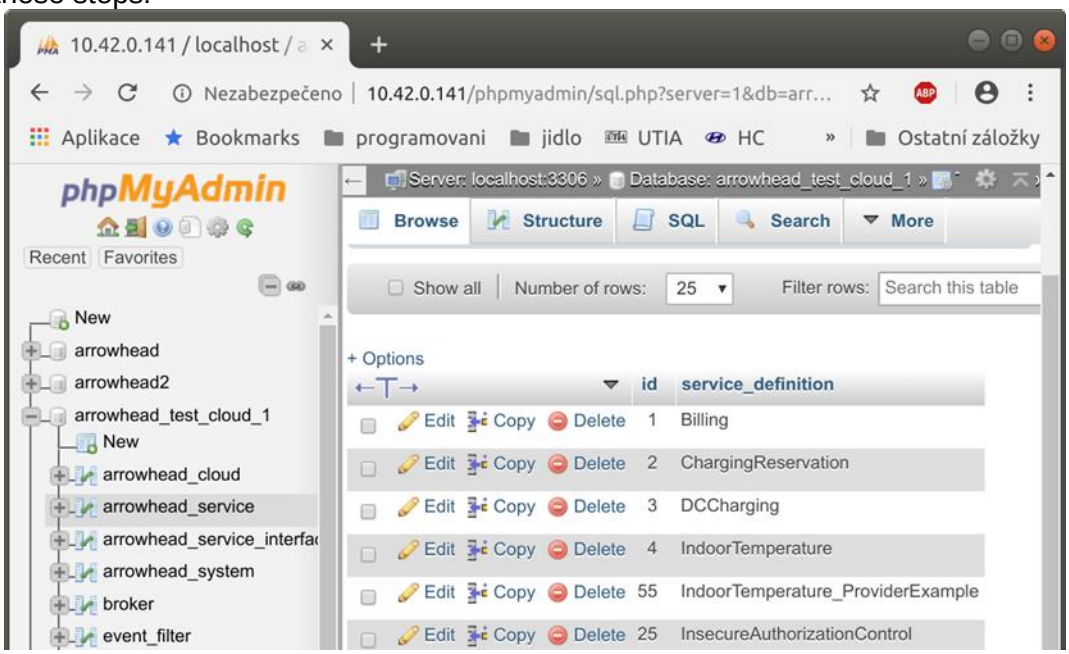


Figure presents win7 laptop screen. There are with two *midnight commander* programs running on two ZynqBerry boards.

© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

10 Modification of Arrowhead Database

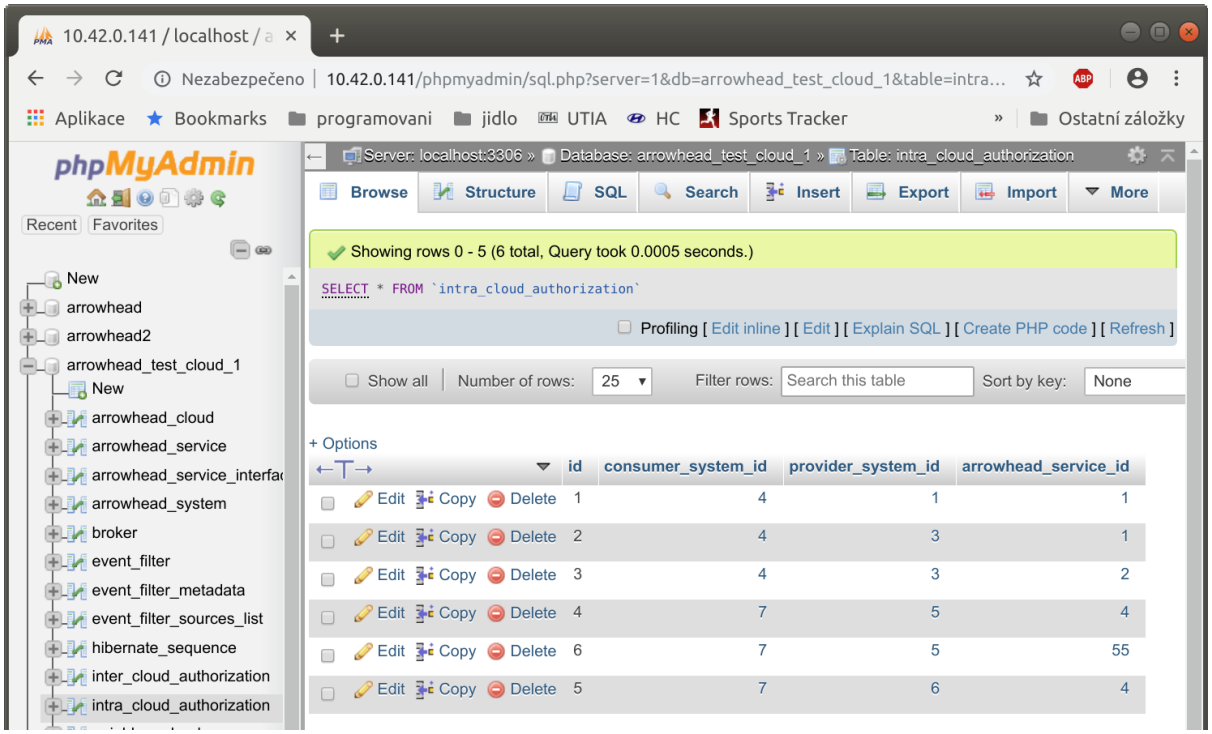
The Arrowhead framework running on RPi3 provides *phpMyAdmin* interface to control its database. To allow the *ConsumerExample* to get the *ProducerExample* service response, follow these steps:



1. On your Win7 or Win 10 PC, start web browser and go to the RPi3 *phpMyAdmin* web page, *http://10.42.0.141/phpmyadmin* (use the IP address of your RPi3).
   User name: *root* password: *root*
2. Get an ID of the *ProducerExample*.
   Select table *arrowhead_test_cloud_1→arrowhead_system*
   and locate the line containing the IP address of the ZynqBerry with system_name *SecureTemperatureSensor*.
   In our case the ID is 5.
3. Get an ID of the *ConsumerExample*.
   Select table *arrowhead_test_cloud_1→ arrowhead_system*
   Locate the line containing system_name:
   client1.
   In our case it is 7.
4. Get an ID of the *ProducerExample* service.
   Select table *arrowhead_test_cloud_1→ arrowhead_service*
   Locate the line containing service_definition called:
   IndoorTemperature_ProviderExample.
   In our case the ID is 55.
5. In table *service_registry, check* if the *ProviderExample* is linked with its service.
   Link the *ProviderExample*, its service and the *ConsumerExample* together. In table *intra_cloud_authorization,* add a new line containing: *consumer_system_id* 7, *provider_system_id* 5 and *arrowhead_service_id 55*.

© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

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



11 Test the ZynqBerry Consumer and Producer

The *ProducerExample* server is running on the “Producer” ZynqBerry board, now.

Execute the *ConsumerExample* client example on the “Consumer” ZynqBerry 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"}
```

This concludes the complete demo of Producer and Consumer on two ZynqBerry boards omplemented as C++ SW code compatible with the Arrowhead framework G4.0 lite-installation on the RPi3 board.

Producer service and Consumer client can run on a single Zynqbeery board or two different ZynqBerry boards. The configuration files and the configuration of the Arrowhead framework database described in Chapter 6 - Chapter 10 provides setup for single ZynqBerry board.

Change of the setup for two ZynqBerry boards involves only modification of the corresponding Ethernet addresses assigned by the DHCP server.

The HW accelerated matrix multiplication demo can be executed on both ZynqBerry boards by executing:

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

See the HW acceleration measured by the number of Arm A9 clock cycles.

© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

12 Producer with real temperature measurement on ZynqBerry

Real temperature of the Xilinx chip of the “producer” ZynqBerry board can be measured by modified *ProviderExample.cpp* code. All other files remain identical. Recompile the *ProviderExample* project by *make.* Test it on the the “Provider” ZynqBerry board.

This is source code of the *ProviderExample.cpp* code measures the temperature of the chip:

```
#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/platform/drivers/xadc/f8007100.adc/iio\:device0/in_temp0_raw"
#define TEMP_OFFSET_FILE
"/sys/bus/platform/drivers/xadc/f8007100.adc/iio\:device0/in_temp0_offset"
#define TEMP_SCALE_FILE
"/sys/bus/platform/drivers/xadc/f8007100.adc/iio\:device0/in_temp0_scale"
const std::string version = "4.0";
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 ) {
```


© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

```
        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);
```


© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

```
    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;
}
```


© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

13 Configuration of PetaLinux and Debian (optional)

The configuration/compilation of the Petalinux 2018.2 kernel and Debian 9.8 Stretch image is described in the second part of this application note.

The configuration/compilation requires Ubuntu 2016-04 installation on 64bit virtual machine. We have used as virtual machine the *VMware Workstation 14 Player* on Win7 or Win10 PC based on the Intel i7 CPU (8 processors, 16 GB RAM).

For the Ubuntu image we use the configuration of the VM machine with allocated 6 processors and 8 processors and 8 GB of RAM.

The Petalinux 2018.2 distribution is downloaded to the Ubuntu 2016-04 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/settings.sh
```

To target the Debian OS, and the PetaLinux 2018.2 distribution provided by the Trenz Electronic requires modification helping to configure the Petalinux kernel image and its file system on two separate partitions of the SD card.

1. On PC Win7 or Win10 execute all steps as described in chapter 3.
2. Copy to the Ubuntu OS all content of these to Win7 or Win 10 directories:

   ```
   X:\TE0726_zsys_SDSoC\prebuilt
   X:\TE0726_zsys_SDSoC\os
   ```

   to Ubuntu directories:

   ```
   /home/devel/work/TV0726/TE0726_zsys_SDSoC/os
   /home/devel/work/TV0726//TE0726_zsys_SDSoC/prebuilt
   ```

   Copy the Debian configuration script *install-arrohead-cli-dep.sh* from this evaluation package to

   ```
   cd /home/devel/work/TV0726/TE0726_zsys_SDSoC/os/petalinux/ install-arrohead-cli-dep.sh
   ```
3. In Ubuntu, open linux terminal window and set path to PetaLinux 2018.2 tool (modify the path if necessary):

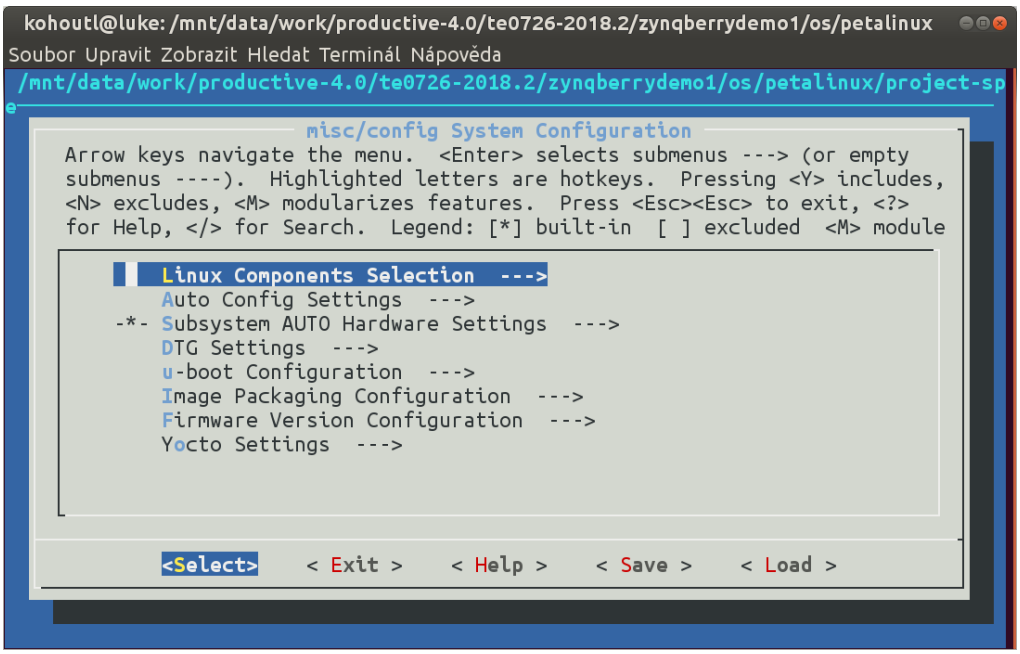
   ```
   source /opt/petalinux/petalinux-v2018.2-final/settings.sh
   ```
4. Go to the folder with PetaLinux, it already contains a prepared configuration according to ZynqBerry board requirements.

   ```
   cd /home/devel/work/TV0726/TE0726_zsys_SDSoC/os/petalinux
   ```
5. The HDF file created (see chapter 3) in Win7 or Win 10 in Vivado 2018.2 tool is therefore in the Ubuntu folder:

   ```
   /home/devel/work/TV0726/TE0726_zsys_SDSoC/prebuilt/hardware/m/TE0726_zsys_SDSoC.hdf
   ```
6. Load the HDF to current PetaLinux configuration.

   ```
   petalinux-config
   --get-hw-description=/home/devel/work/TV82/prebuilt/hardware/m
   ```


© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved



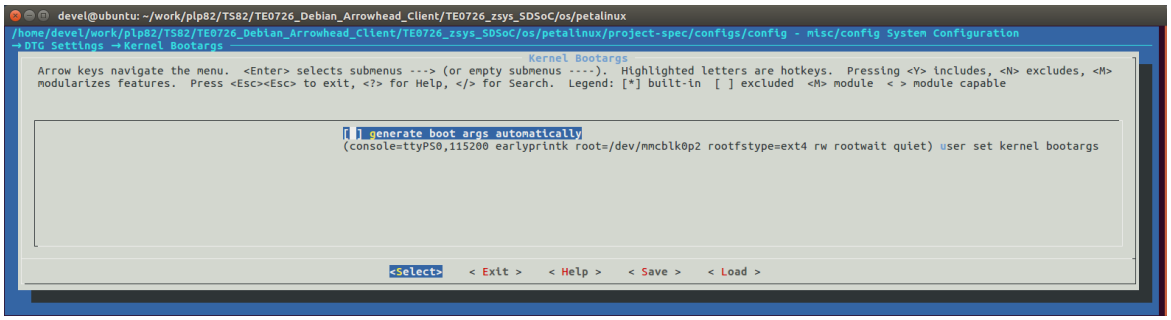
7. Test if the PetaLinux filesystem location is changed from the ramdisk to the extra partition on the SD card, select:

```
Image Packaging Configuration  --->
      Root filesystem type (SD card)  --->
```

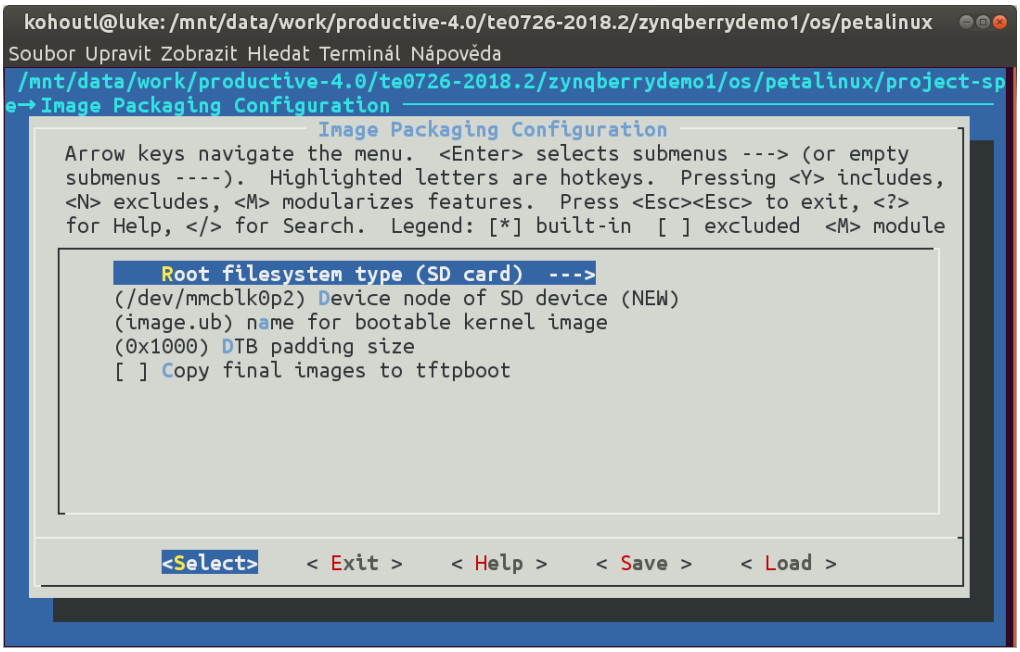
8. Test if option to generate boot args automatically is disabled and if user defined arguments are set to

```
console=ttyPS0,115200 earlyprintk root=/dev/mmcblk0p2 rootfstype=ext4 rw
rootwait quiet
```

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



© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved



9. Build PetaLinux, from the bash terminal execute

```
petalinux-build
```

10. Files *image.ub* and *u-boot.elf* are created in Ubuntu folder

```
/home/devel/work/TV0726/TE0726_zsys_SDSoC/os/petalinux/images/linux/image.ub
/home/devel/work/TV0726/TE0726_zsys_SDSoC/os/petalinux/images/linux/u-boot.elf
```

14 Configuration and compilation of Debian for ARM (optional)

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/TV0726/TE0726_zsys_SDSoC/os/petalinux/mkdebian.sh
```

2. Go to the folder with PetaLinux:

```
cd /home/devel/work/TV0726/TE0726_zsys_SDSoC/os/petalinux
```

3. 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.

```
sudo apt install package_of_the_missing_tool
```


© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

Next table summarizes all the 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 image with Debian. 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.

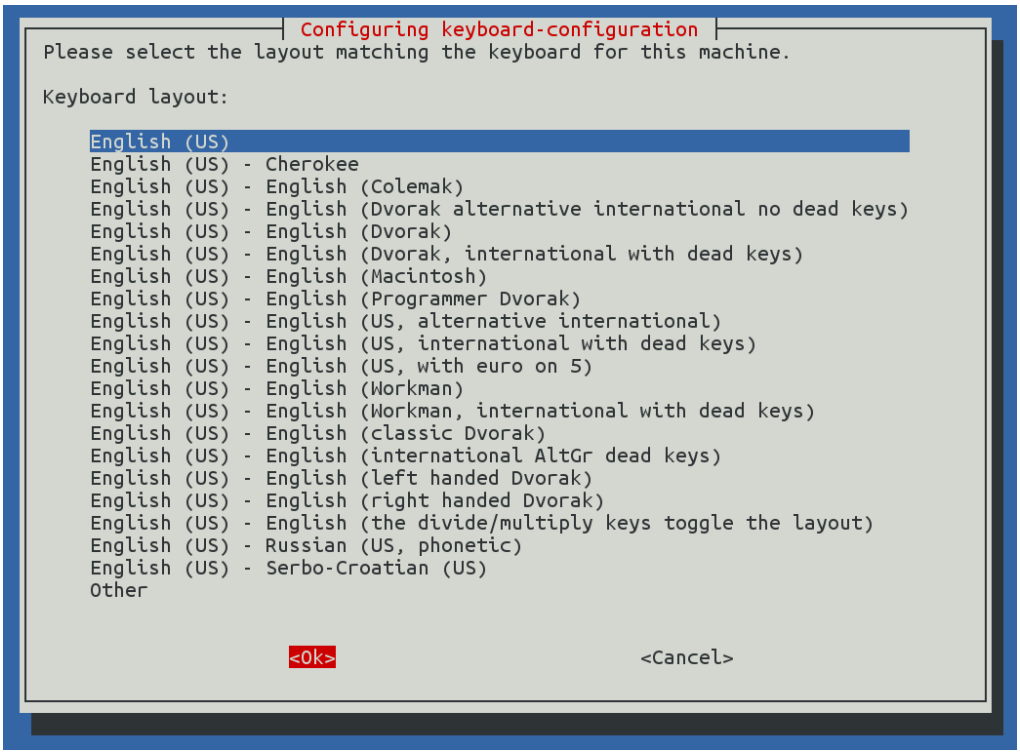
   Create the Debian image from the external Ethernet repositories by this command:

```
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 *te0726-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. Precompiled image can be found in the *te0726-arrohead-client/debian/te0726-debian.img.zip* file.

© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved



5. Compress the created image to file te0726-debian.zip:

```
zip te0726-debian te0726-debian.img
```

6. Copy from Ubuntu

```
/home/devel/work/TV0726/TE0726_zsys_SDSoC/os/petalinux/te0726-debian.zip
```

to Win7 or Win 10 file:

```
X:\TE0726_zsys_SDSoC\prebuilt\os\petalinux\default\te0726-debian.zip
```

7. Copy from Ubuntu

```
/home/devel/work/TV0726/TE0726_zsys_SDSoC/os/petalinux/images/linux/image.ub
```

to Win7 or Win 10 file:

```
X:\TE0726_zsys_SDSoC\prebuilt\os\petalinux\default\image.ub
```

8. Copy from Ubuntu

```
/home/devel/work/TV0726/TE0726_zsys_SDSoC/os/petalinux/images/linux/u-boot.elf
```

to Win7 or Win 10 file:

```
X:\TE0726_zsys_SDSoC\prebuilt\os\petalinux\default\u-boot.elf
```

9. In Ubuntu, clean Petalinux project files

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

10. In Ubuntu, delete files

```
/home/devel/work/TV0726/TE0726_zsys_SDSoC/os/petalinux/te0726-debian.zip
/home/devel/work/TV0726/TE0726_zsys_SDSoC/os/petalinux/te0726-debian.img
```

11. Shut down the Ubuntu 2016-04 operating system.
12. In Win7 or Win 10, close the VMware Workstation Player 14.

© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

You can continue with preparation of the ZynqBerry board (as described in chapters 5 and 6) and use these re-created files:

- Petalinux kernel image *image.ub*
- Compressed Debian image *te0726-debian.zip*
- U-boot program *u-boot.elf*

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

15 Package content

```
├── debian
│   ├── mkdebian.sh
│   ├── image.ub
│   ├── u-boot.elf
│   └── te0726-debian.zip
└── zynq
    ├── TE0726_zsys_SDSoC.zip
    └── install-arrohead-cli-dep.sh
```

References

[1 Trenz Electronic, "TE0726 TRM," [Online]. https://shop.trenz-electronic.de/en/27229-Bundle-ZynqBerry-512-MByte-DDR3L-and-SDSoC-Voucher?c=350 .

[2]] Documents for Arrowhead Framework Available:https://forge.soa4d.org/docman/?group_id=58

© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved

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.

© 2019 ÚTIA AV ČR, v.v.i.
All disclosure and/or reproduction rights reserved